ЃОРЃИ ЈОВАНЧЕВСКИ
СИЛВИЈА НИКОЛОВСКА

ОСНОВИ

НА

ПРОГРАМИРАЊЕ

ЗА II ГОДИНА
ЕЛЕКТРОТЕХНИЧКА СТРУКА

Скопје, 2022

Ѓорѓи Јованчевски
Силвија Николовска

ОСНОВИ НА ПРОГРАМИРАЊЕ

УЧЕБНИК ЗА II ГОДИНА

ЕЛЕКТРОТЕХНИЧКА СТРУКА

Скопје, 2022

ОСНОВИ НА ПРОГРАМИРАЊЕ

УЧЕБНИК ЗА II ГОДИНА
ЕЛЕКТРОТЕХНИЧКА СТРУКА

Автори:

Ѓорѓи Јованчевски
Силвија Николовскаѓи

Рецензенти:

Дејан Спасов
Билјана Пејовска
Викторија Доказа

Јазична редакција:

Виолета Јанушева
Јованка Јованчевска

Стручна редакција:

Цеваир Беќири

Уредници:

Елена Стефановска
Тамара Јовановиќ Нешовска

Графичко и техничко уредување:

Владанка Колева - АРС СТУДИО
Евгенија Павлова - АРС СТУДИО

Место и година на издавање:

Скопје,2022, 2022

Издавач:

Министерство за образование и наука на Република Северна македонија,ул.„Св. Кирил и Методиј“ бр.54, 1000 Скопје,Со одлука број
26-548/1 од 19.08.2022 година на Националната комисија за учебници, одобрува употребата на овој учебник

CIP - Каталогизација во публикација
Национална и универзитетска библиотека "Св. Климент Охридски", Скопје

004.42/.43(075.3)

ЈОВАНЧЕВСКИ, Ѓорѓи
Основи на програмирање [Електронски извор] : учебник за II година : електротехничка струка / Ѓорѓи Јованчевски, Силвија Николовска. - Скопје : Министерство за образование и наука на Република Северна Македонија, 2022

Начин на пристапување (URL): https://e-ucebnici.mon.gov.mk/pdf/Osnovi na programiranje 1 mak.pdf. - Текст во PDF формат, содржи 314 стр., илустр. - Наслов преземен од екранот. - Опис на изворот на ден 28.12.2022. - Библиографија: стр. 313. - Содржи и: Додаток

ISBN 978-608-273-134-6

1. Николовска, Силвија [автор]

COBISS.MK-ID 59066117

С О Д Р Ж И Н А

Вовед

Учебников претставува почетен курс за изучување на програмирањето. Тој е конципиран да ги опфати основите на алгоритамското изразување на проблемите, преку посебен псевдојазик на македонски, како и кодирањето на алгоритмите во програмскиот јазик C++.

Учебникот е усогласен со Наставната програма за предметот „Основи на програмирање“, за II година електротехничка струка. Тој е напишан според модуларната програма за овој предмет, која се состои од три модуларни единици.

Во првата модуларна единица „Вовед во програмскиот јазик C++“, ученикот се запознава со алгоритмите и нивното текстуално и графичко изразување, со програмските јазици и со околините за програмирање. Понатаму, се запознава со основните елементи на јазикот C++, како: променливи, типови на податоци, наредби за читање и за печатење и структура на програма во јазикот C++.

Во втората модуларна единица „Контрола на тек и функции во C++“, се објаснети основните алгоритамски контролни структури за контролирање на текот на извршувањето на алгоритмите: контролните структури за избор и контролните структури за повторување. Тие се доволни за да се изрази секој алгоритам и да се контролира текот на неговото извршување. Нивната имплементација е објаснета и илустрирана преку соодветните контролни наредби во програмскиот јазик C++. Исто така, во оваа модуларна единица, се обработени подалгоритмите, како и соодветната имплементација со функции во јазикот C++. Обработени се библиотечните функции во C++, а повеќе внимание е посветено на кориснички дефинираните функции.

Во третата модуларна единица „Сложени типови на податоци во C++“, ученикот се воведува во користење на бројните низи, на покажувачи и на стрингови при програмирање во C++.

Сметаме дека изложениот материјал е доволен за секој ученик почетник во програмирањето, а неговото совладување не е тешко бидејќи настојувавме да го изложиме на лесен и достапен начин, со многу примери, вежби и решени задачи. По секоја наставна единица, се поставени прашања за проверка на знаењето и задачи за самостојно решавање.

Авторите

***Забелешка**: Во соработка со лекторот, авторите се обидоа да изразат на македонски јазик некои информатички термини, но и термини од програмскиот јазик C++. Тие, можеби, ќе звучат невообичаено за некои читатели, но сметаме дека е крајно време да се поработи на проблемот на употребата на информатичките термини во македонскиот јазик.*

Конкретно, поради специфичноста на овој програмски јазик, но и на јазикот што се употребува во информатиката кај нас, воопшто, одредени термини само се транскрибираат на македонски јазик. Во учебникот се почитува мислењето на авторите дека некои поими, поради нивната многузначност, во овој програмски јазик треба да се напишат во форма која не соодветствува со правилата за нивната употреба во македонскиот јазик.

МОДУЛ

ВОВЕД ВО ПРОГРАМСКИОТ ЈАЗИК C++

1

Во оваа глава ќе се запознаете со следното:

- Програма, програмирање и програмски јазици.
- Софтверот и неговата поделба.
- Алгоритам, алгоритамски чекор, општ и детален алгоритам.
- Претставувањето на алгоритмите.
- Структурираното програмирање.
- Псевдојазикот, блок-дијаграмот, изворната програма и извршната програма.
- Интегрираните развојни околини за C++.
- Историјата на C++.
- Елементите на јазикот C++.
- Величина, податок, константа, променлива и тип на променлива.
- Типот на податок.
- Форматите за претставување на децималните броеви.
- Преименувањето на типот, автоматското одредување на типот.
- Читањето и печатењето податоци.

Клучни зборови

bool	Исказ
char	Клучен збор
cin	Кодирање
Code::Blocks	Коментар
cout	Корисничка програма
double	Литерал
endl	Логички израз
enum	Логички јазик
enum	Логички оператор
float	Логички тип на податок
int	Машински јазик
iomanip	Машински независен јазик
iostream	Модуларно програмирање
istream	Наброив тип на податок
long	Назабување
long double	Неподвижна точка
long long	Неструктуриран тип на податок
Microsoft Visual Studio	Објектно ориентиран јазик
namespace	Оператор за декрементирање – –
ostream	Оператор за инкрементирање ++
return	Оператори со битови
short	Оператор за вметнување (инсертирање)
string	Оператор за издвојување (екстрахирање)
Unified Modeling Languge – UML	Општ алгоритам
unsigned char	Основен тип на податок
unsigned int	Поврзувач
unsigned long	Податок
unsigned long long	Подвижна точка
unsigned short	Преведувач (компајлер)
Автоматско одредување на типот	Преименување на типот на податокот
Адресен тип на податок	Преполнување
Алгоритам	Проблемски ориентиран јазик
Алгоритамска контролна структура	Програмирање одгоре надолу
Алгоритамски чекор	Програмски јазик

Апликативна програма	Променлива
Апликација	Прост тип на податок
Асемблер	Процедурален програмски јазик
Блок-дијаграм	Псевдојазик
Булови функции	Реален тип на податок
Вграден тип на податок	Редоследна контролна структура (секвенца)
Величина	Резервиран збор
Виш програмски јазик	Релациски оператор
Влезен поток на податоци	Стандардна библиотека на C++
Главна функција	Семантика на програмски јазик
Граматика на програмски јазик	Симболички јазик
Декларативен програмски јзаик	Синтакса на програмски јазик
Декларација на променлива	Системски програми
Детален алгоритам	Скратена форма на оператор (сложен оператор)
Дефинирање на променлива	Скриптен јазик
Дибагер	Сложен тип на податок
Експлицитна типска конверзија (кастирање на типот)	Софтвер
Знак за нова линија	Спецификатор на типот
Знаковен тип на податок	Спојување стрингови
Идентификатор	Стринг
Изворна програма	Структуриран тип на податок
Извршна програма	Структурирано програмирање
Излезен поток на податоци	Текстуален податок
Излезна (ескејп) секвенца	Текстуален уредувач – едитор
Именувана константа	Тип
Императивен програмски јазик	Функција
Имплицитна конверзија (типска принуда)	Функциски јазик
Иницијализација на променлива	Хедер-датотека
Интегрирана развојна околина	Целоброен тип на податок
Интерпретатор	

1.1 Програми

Компјутерот не е способен самостојно да изврши никаква работа. За да може компјутерот да изврши каква било работа, потребно е да му се зададат соодветни наредби со кои се активира и се извршува некоја **програма** (англ. program). Ако програмата е правилно напишана, компјутерот ќе ја изврши и ќе ги даде соод-ветните резултати.

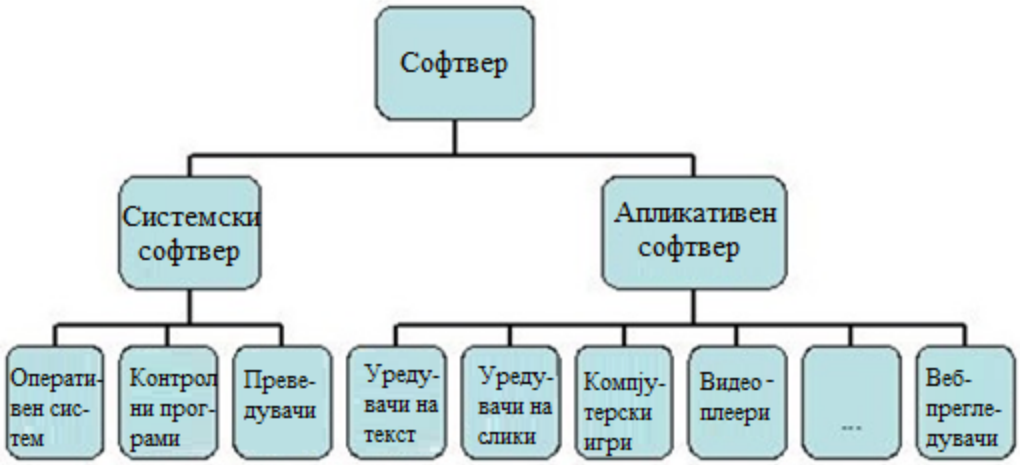
Целокупната работа на компјутерот се одвива под контрола на посебни т.н. **системски програми** (англ. system programs), кои може да се поделат во три групи:

- **Оперативни системи** (англ. operating systems).
- **Контролни програми** (англ. control programs).
- **Услужни програми** (англ. utility programs).

Програмите кои извршуваат одредени работи за корисниците се нарекуваат **апликативни програми** (англ. application programs) или **кориснички програми** (англ. user programs).

Сите апликативни програми кои ги имаме на нашиот компјутер решаваат одредени задачи, односно со нив извршуваме одредена работа. На пример, познати апликативни програми се програмите од пакетот Microsoft Office: MS Word, MS Excel, MS PowerPoint итн.

Сите програми кои може да ги извршува компјутерот, со едно име се наречени **софтвер** (англ. software). На ***слика 1.1.1*** е дадена една можна поделба на софтверот.



Слика 1.1.1

Програма[1] претставува текст кој компјутерот може да го разбере и изврши. Програмите се пишуваат во некој **програмски јазик** (англ. programming language), кој служи за комуникација меѓу човекот и компјутерот. Програмите се пишуваат со наредби кои компјутерот ги „разбира“ и ги извршува оние активности кои се „искажани“ со нив.

Програмите може да се уредуваат (пишуваат и менуваат) со некој уредувач на текст, да се паметат како датотеки и да се извршуваат.

Пишувањето програма е процес наречен **програмирање** (англ. programming), а луѓето што ги пишуваат програмите (програмираат) се нарекуваат **програмери** (англ. programmers).

1.2 Алгоритми

Секој човек точно знае што треба да направи кога треба да свари кафе, да подготви одредено јадење, да вози кола, да инсталира програма на компјутер, да си ја подготви чантата за следниот ден итн.

Човечките активности, во голем дел, може да се извршат како редослед од определен број на одделни помали (елементарни) активности, т. е. активности за кои не е потребно објаснување за да бидат извршени.

На пример, елементарните активностите при варење кафе може да бидат:

- Турање вода во ѓезвето.
- Ставање кафе во ѓезвето.
- Ставање на ѓезвето на рингла.
- Приклучување на рингла на која е ставено ѓезвето.
- Чекање кафето да зоврие.
- Тргање на ѓезвето од рингла.
- Исклучување на рингла.

За да може еден изведувач (како оној што вари кафе) да изврши дадена активност (варење кафе), тој треба да ги знае елементарните активности и постапката (редоследот) на нивното извршување. Таквата низа од елементарни активности се нарекува постапка или **алгоритам** (англ. algorithm).

Можеме да кажеме дека:

алгоритам ***е постапка од конечен број строго дефинирани активности и точно зададен редослед на нивното извршува е***.

Поимот алгоритам ќе го објасниме подетално на примерот за наоѓање на најголемиот број од три дадени броја, со следнава постапка:

- Споредување на кои било два броја и одредување на поголемиот од нив.
- Споредување на поголемиот број од првото споредување со третиот број и одредување на поголемиот од нив.

[1] Една програма напишана во програмскиот јазик C++ е дадена на ***слика 1. .4***.

Поголемиот број при второто споредување е најголемиот од трите броја.

Очигледно е дека постапката се состои од само две точно дефинирани активности.

За да се примени оваа постапка врз кои било три броја, потребно е тие да бидат претходно зададени. Резултатот од постапката е најголемиот број. Тоа значи дека решавањето на некоја задача се состои во одредување на *излезните резултати* со примена на одредена постапка – алгоритам врз *влезните податоци*.

Затоа, може да се рече дека:

алгоритам *претставува постапка која се состои од конечно мно ество точно дефинирани активности, применети врз влезните податоци по строго пропишан редослед, со кои се доаѓа до излезни резултати*.

Зборот *алгоритам* е земен од латинскиот јазик и претставува латински превод на презимето Al–Khwarizmi на арапскиот математичар од IX век Abu Ja'far Muhammad ibn Musa al–Khwarizmi, кој прв ги формулирал правилата за извршување на четирите основни аритметички операции со арапски цифри.

Алгоритамски чекори

Активностите од кои се состои еден алгоритам се нарекуваат **алгоритамски чекори** (англ. algorithms steps). Во зависност од тоа дали чекорите се поопшти или подетални, алгоритамот може да биде **општ** или **детален**.

На пример, општиот алгоритам за задачата од претходната точка (одредување на најголемиот од три дадени броја) можеме да го запишеме како на ***слика 1.2.1***.

чекор 1. Задавање на три броја.
чекор 2. Споредување на кои било два броја и наоѓање на поголемиот од нив.
чекор 3. Споредување на поголемиот број најден во чекор 2 со третиот број и наоѓање на поголемиот од нив.
чекор 4. Печатење на резултатот.

Слика 1.2.1

Ако ги означиме броевите со a, b и c, поголемиот од a и b со p, поголемиот од p и c со n, тогаш може да напишеме подетален алгоритам, како на ***слика 1.2.2***.

чекор 1. Задавање на броевите a, b и c.
чекор 2. Ако a е поголем од b, тогаш p = a, инаку p = b.
чекор 3. Ако p е поголем од c, тогаш n = p, инаку n = c.
чекор 4. Печатење на n.

Слика 1.2.2

Рековме дека во секој алгоритам има влезни податоци и излезни резултати. Излезните резултати се добиваат со обработка („трансформација“) на влез-

ните податоци при извршување на алгоритамот чекор по чекор. Притоа, обработката не се врши секогаш директно, туку со посредство на **меѓурезултати**. Така, во горниот алгоритам меѓурезултат е p, влезни податоци се a, b и c, а излезен резултат е n.

За да ја провериме исправноста на алгоритамот, се врши тестирање врз произволни влезни податоци. На пример, да го тестираме претходниот алгоритам за следните броеви: a = 37, b = 12 и c = 44.

чекор 1. a = 37, b = 12, c = 44.
чекор 2. a (= 37) е поголем од b (= 12), тогаш p = a (= 37).
чекор 3. p (= 37) не е поголем од c (= 44), затоа n = c (= 44).
чекор 4. печати n (= 44).

Задачи за вежбање

1. Напишете алгоритам за правење торта.
2. Напишете алгоритам за утринска подготовка пред да излезете од дома.
3. Напишете алгоритам за копирање датотеки од диск на уесбе.
4. Кои се можните чекори за одредување на средната вредност на три броја?
5. Обидете се да напишете алгоритам (со алгоритамски чекори) за издвојување на средната цифра од даден трицифрен број.
6. Напишете алгоритам за утврдување дали некој природен број n е парен или непарен. (Ако бројот е делив со 2, тогаш е парен, инаку е непарен).
7. Напишете алгоритам со кој ќе најдете кој ден по ред во годината е денес. На пример, ако денес е 01.09.2020 година, денес е 245-тиот ден во годината.
8. Напишете општ алгоритам за наоѓање на најголемиот заеднички делител (НЗД) за два природни броја.
9. Напишете општ алгоритам за наоѓање на најмалиот заеднички содржател (НЗС) за два природни броја.
10. Напишете алгоритам со кој ќе пресметате колку години, месеци и денови имате денес.

1.3 Претставување на алгоритмите

Алгоритмите може да се претстават на три начина:

- Текстуално.
- Графички.
- Со програмски јазик.

Текстуалното претставување на алгоритмите се врши со чекори, како што е покажано во потточката **1.2 Алгоритми**. Притоа може да се користи и т.н. **псевдојазик** за опишување на алгоритмите.

Ние ќе користиме псевдојазик (со зборови од македонскиот јазик), кој се состои од зборовите: **алгоритам, подалгоритам, почеток, крај, ако, тогаш, инаку, додека, извршувај, до, за, чекор, зголемувај, намалувај, читај, печати, прекин, продолжи, скок, излез**.

Овие зборови ќе ги пишуваме зацрнето (задебелено) за да потенцираме дека тие се зборови резервирани за псевдојазикот.

Пример за текстуално претставување на алгоритам со ваков псевдојазик е даден на ***слика 1. .1***, каде што е претставен алгоритамот за наоѓање на најголемиот од три броја од ***слика 1.2.2***. Во него се користи стрелка со која се изразува чекор за доделување. На пример, со стрелката p ← a се изразува дејството со кое на p ѝ се доделува вредноста на a.

```
алгоритам Најголем
почеток
        читај a, b, c;
        ако a > b
                тогаш
                        p ← a
                инаку
                        p ← b
        крај_ако {a > b}
        ако p > c
                тогаш
                        n ← p
                инаку
                        n ← c
        крај_ако {p > c}
        печати n;
крај {Најголем}
```

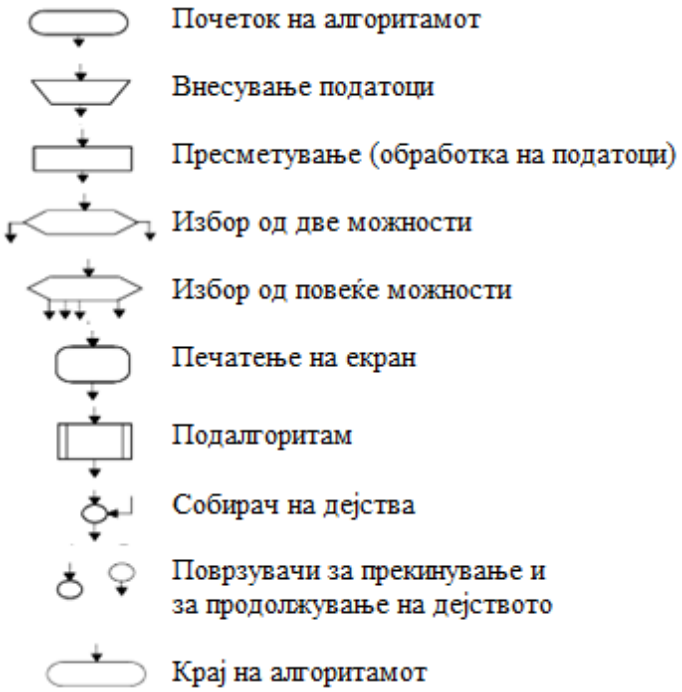
Слика 1. .1

Графичкото претставување на алгоритмите се врши со т.н. **блок-дијаграм** (англ. flowchart). Во блок-дијаграмот се користат посебни графички симболи (блокови) за одредени дејства (операции), дадени на ***слика 1. .2***. Претставувањето на алгоритмите со овие графички симболи се нарекува **стандардно графичко претставување**. Алгоритамот за задачата за одредување на најголемиот од трите дадени броја, претставен текстуално на ***слика 1. .1***, со блок-дијаграмот е даден на ***слика 1. .*** .

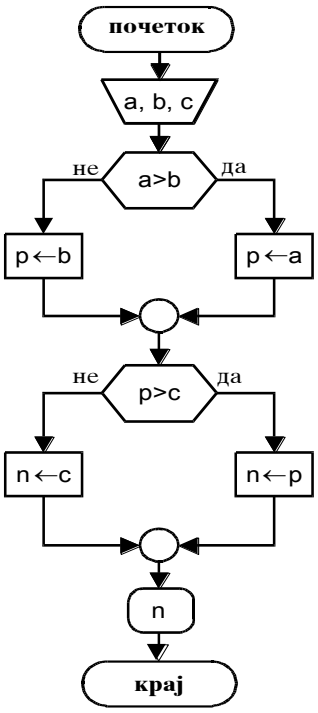
Графичкото претставување на алгоритмите има и свои предности и свои недостатоци. Предноста е во поголемата прегледност на текот на активностите во алгоритамот бидејќи човекот полесно перципира слика од текст. Од друга страна, блок-дијаграмот за поголем алгоритам може да зафаќа повеќе страници и алгоритамот да биде непрегледен.

Претставувањето на алгортимите може да биде и директно со програмски јазик. Притоа, алгоритамот го „изразуваме“, т. е. **кодираме** (англ. encoding) со наредбите од некој програмски јазик. Кодираниот алгоритам со наредби од некој

програмски јазик се нарекува **изворна програма** (англ. source program). За да може да се изврши изворната програма, таа мора да се преведе во **извршна програма** (англ. executive program).



Слика 1. .2



Слика 1. .

На пример, разгледаниот алгоритам за наоѓање на најголемиот број од 3 дадени броја, кодиран во програмскиот јазик C++ е даден на ***слика 1. .4.***

```
1   #include <iostream>
2   using namespace std;
3
4   int main() {    // Najgolemiot od tri broja.
5
6       double a, b, c, p, n;
7       cout << "Vnesete prv broj: ";
8       cin >> a;
9       cout << "Vnesete vtor broj: ";
10      cin >> b;
11      cout << "Vnesete tret broj: ";
12      cin >> c;
13      if (a > b)
14          p = a;
15      else
```

Слика 1. .4

```
16          p = b;
17      if (p > c)
18          n = p;
19      else
20          n = c;
21      cout << "Najgolem od broevite " << a << ", "
22          << b << " i " << c << " e " << n << endl;
23
24      cout << endl;
25      system("Color 17");
26      system("pause");
27      return 0;
28  }
```

Слика 1. .4 продол ение

По извршување на програмата, излезот е:

```
Vnesete prv broj: 123
Vnesete vtor broj: 321
Vnesete tret broj: 234
Najagolem od broevite 123, 321 i 234 e 321
```

Структурирани алгоритми

Алгоритмите за посложени задачи може да бидат многу долги и непрегледни. Затоа, и програмите што ќе се напишат според тие алгоритми може да бидат тешко разбирливи. Тоа е посебно важно кога ќе се јави потреба од некоја измена или надополнување во нив. За полесно снаоѓање во големите програми, денес тие се пишуваат со техника за програмирање позната под името **структурирано програмирање** (англ. structured programming).

Името структурирано програмирање е добиено според користењето на т.н. **алгоритамски контролни структури** (англ. algorithm control structures), со кои се контролира дејството на алгоритмите.

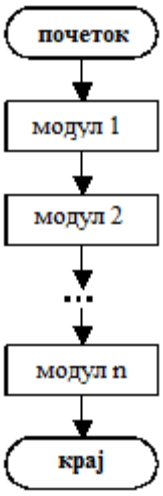
Во структурираното програмирање се користат две техники за програмирање:

- **Програмирање одгоре надолу** (англ. top-down programming).
- **Модуларно програмирање** (англ. modular programming).

Програмирање одгоре надолу се врши со разделување (расчленување) на задачата на помали и поедноставни задачи, т.н. **подзадачи**. Ако е потребно, и тие подзадачи понатаму се разделуваат на уште поедноставни подзадачи сè додека не се добијат задачи што лесно се програмираат.

Секоја подзадача од така расчленетата задача може да се разгледува како посебна задача, независно од другите подзадачи. За секоја подзадача може да се напише посебен алгоритам, а потоа да се напише и посебна програма, според алгоритамот. Таквите програми за подзадачите се нарекуваат **модули** (англ. modules).

Секој модул има само една влезна точка (почеток) и само една излезна точка (крај), ***слика 1. .5***. Сите модули во програмата се подредени секвенцијално (еден по друг) и не може ниту еден од нив да се прескокне. Функцијата што ја извршува еден модул не смее да се повторува во друг модул, односно не смее да има преклопување на модулите. Исто така, еден модул може да се состои од повеќе други модули, и, спротивно, може да биде и дел од поголем модул.



Слика 1. .5

Модуларното програмирање е посебно корисно при измена на програмите или при поправка на грешки. Ако треба да се измени програмата или да се поправи некоја грешка во неа, тогаш се менува само модулот во кој се јавила грешката, а не се менуваат другите модули.

Графичкото претставување на алгоритмите при структурираното програмирање, се разликува од графичкото претставување при неструктурираното програмирање, кое се нарекува **стандардно претставување**.

Структурираните алгоритми се претставуваат со правоаголни блокови. Секој правоаголник изразува посебна целина (модул), со свој почеток и крај. Ако некој модул се дели на поедноставни модули, тие се претставуваат со помали правоаголници во поголем. На пример, алгоритамот за наоѓање на најголемиот од три дадени броја, претставен стандардно на ***слика 1. .*** , структурирано е претставен на ***слика 1. .6***.

Прашања за проверка на знаењето

1. Што е програма, а што програмирање?
2. Како се нарекуваат јазиците во кои се програмира?
3. Кои се системски програми, а кои се апликативни програми?
4. Како се групираат системските програми?
5. Која е главната поделба на софтверот?
6. Дали Microsoft Windows е апликативна програма?
7. Во која група апликативни програми би ја ставиле програмата Microsoft Excel?
8. Наведете три апликативни програми кои работат на Вашиот компјутер. Обидете се да ги ставите во соодветната категорија од второто ниво на ***слика 1.1.1.***
9. Обидете се да дадете пример за програма за

```
читај a, b, c
ако a > b
        тогаш
                p ← a
        инаку
                p ← b
ако p > c
        тогаш
                n ← p
        инаку
                n ← c
печати n
```

Слика 1. .6

секоја од наведените групи софтвер од третото ниво на ***слика 1.1.1***.

10. Што е алгоритам? Наведете една дефиниција.
11. Што е алгоритамски чекор?
12. Каков може да биде алгоритамот според детализираноста?
13. Како може да се претстави еден алгоритам?
14. Со какви дијаграми графички се претставуваат алгоритмите?
15. Наведете ги графичките симболи (блокови) за стандардно графичко претставување на алгоритмите.
16. Што е изворна програма?
17. Дали може компјутерот директно да изврши изворна програма?
18. Што е структурирано програмирање?
19. Кои техники се користат при структурираното програмирање?
20. Како графички се претставуваат структурираните алгоритми?
21. Што претставува програмски модул?
22. Зошто е корисна модуларната техника за програмирање?
23. Објаснете ја (или побарајте да Ви ја објаснат) техниката за програмирање одгоре надолу.

1.4 Програмирање

За да може некој да напише програма за решавање на одредена задача со помош на компјутер, потребно е да знае да програмира.

Програмирањето *е процес на пишува е програма.*
***Луѓето што програмираат се нарекуваат* програмери.**

Фази на програмирање

За да се реши некоја задача со помош на компјутер, не е доволно само да знаеме да програмираме и да напишеме програма. Програмирањето е само еден дел од тој процес, кој се состои од неколку фази:

- Поставување на задачата.
- Дефинирање алгоритам (постапка) за решавање на задачата.
- Пишување програма.
- Тестирање на програмата.

I фаза: Поставување на задачата

При поставување на задачата, треба точно да се дефинираат и да се прецизираат условите под кои таа ќе се решава. Затоа, прво треба задачата правилно да се разбере. Тоа се прави со анализа на нејзината природа, а ако е потребно, и со навлегување во стручната област на која ѝ припаѓа.

За да го дообјасниме погоре кажаното, ќе поставиме неколку задачи и ќе ги анализираме.

1. Да се најде збирот на првите 10 природни броја.
2. Да се реши систем од две линеарни равенки со две непознати.
3. Да се направи список на артиклите во една продавница по количина и по вредност и да се пресмета вкупната вредност на стоката.
4. Од 16 дрвца од кибрит двајца играчи А и Б земаат наизменично по 1, 2, 3 или 4 дрвца. Победува оној кој последен ќе земе дрвце. Дали може еден од нив постојано да победува?

За првата задача, секој ќе рече дека не е тешка – кога знам да собирам, знам и да напишам програма.

За втората задача, треба да се знаат некоја метода за решавање на систем од две линеарни равенки со две непознати, како методата на замена или методата на спротивни коефициенти.

За третата задача, треба да се знае:

- Името на секој артикл.
- Количината на секој артикл.
- Единечната цена по килограм, по литар или по пакување.

За да ја решиме четвртата задача, нема да ставаме дрвца во компјутерот, туку од бројот 16 наизменично ќе одземаме по онолку дрвца колку што ќе каже секој играч – 1, 2, 3 или 4. Суштината на задачата е да се смисли начин по колку дрвца да се земаат, знаејќи колку зел противникот и уште колку останале, за се доземат последните и да се победи.

II фаза: Дефинирање алгоритам за решавање на задачата

По извршената анализа на задачата, потребно е да се дефинира (најде/смисли) или да се избере (ако знаеме повеќе) алгоритам за нејзино решавање. Најчесто, постапката произлегува од извршената анализа на задачата и од консултираната стручна литература од областа во која припаѓа задачата. Притоа, треба да се води сметка постапката да биде применлива за извршување на компјутер. Во неа јасно треба да се наведат сите активности (операции) што треба да ги изврши компјутерот за да даде точни резултати. Секоја операција треба да биде еднозначно дефинирана, а и редоследот на операциите треба да биде точно зададен. Се разбира, целата постапка треба да биде конечна, односно да завршува по конечен број на операции, т. е. по конечно време на извршување. Вака дефинираната постапка за решавање на некоја задача, како што беше и претходно споменато, се нарекува алгоритам.

Да напоменеме дека за решавање на иста задача може да се напишат повеќе различни алгоритми.

III фаза: Пишување програма

Пишувањето програма претставува запишување (изразување, кодирање) на алгоритамот со елементите на некој програмски јазик. (За програмските јазици ќе зборуваме подетално во следниот поднаслов).

IV фаза: Тестирање на програмата

Во четвртата фаза се врши тестирање на програмата за да се провери дали дава точни резултати. Тестирањето се врши најчесто со вредности за кои ги знаеме решенијата или чии решенија можеме (на некој начин) да ги провериме.

Програмски јазици

Луѓето меѓусебно се разбираат и комуницираат со помош на јазикот. Така, без разлика дали е претставен со знаци, со симболи или со говор, јазикот претставува основно средство за комуникација.

Јазиците може да бидат:

- **Природни.**
- **Вештачки.**

За комуникација меѓу себе, луѓето ги користат природните јазици, како на пример, македонскиот, англискиот, рускиот итн. За комуникација помеѓу луѓето и машините (или помеѓу две машини), се користат вештачките јазици.

Посебен вид вештачки јазици се програмските јазици, кои се развиени за комуникација меѓу човекот и компјутерот. Со програмските јазици се изразува постапката (алгоритамот) за решавање на некоја задача запишана во текстуална форма наречена **програма**. Програмите се пишуваат со наредби кои компјутерот ги „разбира“ и ги извршува оние дејства кои се „искажани“ со нив.

Од појавата на компјутерите до денес се развиени голем број[2] програмски јазици. Тие се делат на три групи:

- **Машински јазик.**
- **Симболички јазици.**
- **Виши програмски јазици.**

Машински јазик

Машинскиот јазик (англ. machine language) е јазикот на компјутерот. Тој е единствениот јазик што секој компјутер го разбира и на кој компјутерите непосредно работат. Се состои од одреден број **машински инструкции** (англ. machine

[2] Се смета дека досега се развиени над 3 000 програмски јазици.

instructions), кои се изразуваат само со нули и единици, т. е. само со бинарни цифри[3], ***слика 1.4.1.***

Меѓутоа, записот со бинарни цифри е многу голем и непрегледен. Поради тоа, машинските програми (на хартија) најчесто се запишуваат во *хексадецималниот броен систем*[4], ***слика 1.4.1***.

Машински инструкции

Бинарен запис	Хексадецимален запис
10111000110101100001011O	B8D616
1000111011011000	8ED8
10100000000000000000000	A00000
00000010000001100000000100000000	02060100
10111011000100001000000O	BB1080
1000100000000111	8807
1011010001001100	B44C
1100110100100001	CD21

Слика 1.4.1

Симболички јазици

Симболичкиот јазик (англ. assembly language) е на повисоко ниво од машинскиот. Кај првите компјутери, програмите се пишувале само на машински јазик. Со текот на времето, за полесно и за побрзо претставување на програмата за поставената задача, програмерите почнале да се служат со кратки едноставни зборови, наречени **симболи** или **мнемоници**[5] за означување на машинските инструкции. Така се изградиле симболичките јазици. Името го добиле токму поради тоа што со нив машинските инструкции и податоци се изразуваат на симболички начин.

На пример, машинската програма од ***слика 1.4.1*** има симболички запис како на ***слика 1.4.2***[6].

[3] Бинарни цифри се 0 и 1. Тие се цифри на т.н. *бинарен броен систем*. Нам ни е познат *декадниот броен систем* во кој има 10 цифри – 0, 1, 2, 3, 4, 5, 6, 7, 8 и 9.

[4] *Хексадецималниот броен систем* има 16 цифри, и тоа: 0, 1, 2, 3, 4, 5, 6, 7, 8, 9, A, B, C, D, E и F.

[5] Мнемоника е техника на паметење.

[6] На почетокот на секоја наредба на ***слика 1.4.2*** е прикажана адресата на која се наоѓа таа во меморијата.

Симболичките јазици имаат поголем степен на разбирливост од машинскиот јазик. На пример, од ***слика 1.4.2*** се гледа дека MOV е кратенка за поместување (англ. move), ADD значи додавање, т. е. собирање (англ. add) итн.

Но симболичките јазици се неразбирливи за компјутерот, што значи дека програмата напишана на симболички јазик не може непосредно да се извршува во компјутерот. Затоа, се изработени посебни системски програми наречени **преведувачи** (англ. compilers), кои програмата од симболички јазик ја преведуваат во програма на машински јазик.

```
16D7 : 0000    MOV    AX, 16D6
16D7 : 0003    MOV    DS, AX
16D7 : 0005    MOV    AL, [0000]
16D7 : 0008    ADD    AL, [0001]
16D7 : 000C    MOV    BX, 8010
16D7 : 000F    MOV    [BX], AL
16D7 : 0011    MOV    AH, 4C
16D7 : 0013    INT    21
```

Слика 1.4.2

Симболичките јазици се **машински ориентирани јазици** бидејќи секој компјутер или семејство компјутери има свој сопствен симболички јазик, што зависи од процесорот на компјутерот.

Симболичките јазици најчесто се познати под името **асемблерски јазици** (англ. assembly languages), а нивните преведувачи се нарекуваат **асемблери** (англ. assemblers).

Виши програмски јазици

Вишите програмски јазици (англ. high-level languages), се јазиците на кои се мисли кога зборуваме за програмирање. Тие се развиле почнувајќи од 50-тите години на XX век кога компјутерите со својата универзалност почнале да навлегуваат во многу области на човековото работење. Кај производувачите и корисниците на компјутери се наметнало прашањето: Како компјутерите да се доближат до човекот за да може да ги користат и луѓе кои не ја познаваат техниката на машинското и симболичкото програмирање?

Основната идеја во решавањето на овој проблем била програмите да се пишуваат на јазик сличен со човечкиот природен јазик. За кратко време, вишите програмски јазици биле општоприфатени и користени од голем број корисници.

Брзото прифаќање и масовното користење на овие јазици се должи на следниве факти:

- *Програмирањето значително е олеснето поради блискоста на овие јазици со природниот писмен начин на изразување на човекот и со терминологијата од областа во која припаѓа задачата.*
- *Овозможуваат брзо и ефикасно запишување на алгоритмите.*
- *Не бараат познавање на техничките својства на компјутерот и не се зависни од видот (типот) на компјутерот на кој се извршува програмата, односно иста програма може да се извршува на различни видови (типови) компјутери.*
- *Брзо и лесно се учат.*
- *Овозможуваат размена на програми и на искуства меѓу корисниците.*

Сите наведени факти се базираат на структурата на вишите програмски јазици, која видно се разликува од структурата на машинскиот, односно на симболичките јазици.

Иако вишите програмски јазици се вештачки јазици, тие се изградени на сличен начин како и природните. Имено, секој од тие јазици си има своја **азбука** составена од букви, цифри и специјални знаци: интерпункциски, знаци за аритметички операции, за споредување и друго[7].

Со комбинирање на знаците од абецедата, се формираат елементарните конструкции на јазикот – **зборовите**, кои го сочинуваат **речникот на јазикот**. На пример: read, do, while, new, OPEN, MULTIPLY, STOP, if итн.

Зборовите имаат одредено значење, но во програмите не може да фигурираат како самостојни елементи кои директно ќе влијаат врз работата на компјутерот, туку се комбинираат во јазични конструкции наречени **реченици** или **наредби** (англ. statements). Наредбите имаат точно дефинирано значење и може во програмата да постојат како самостојни целини, т. е. да предизвикаат одредена акција во компјутерот.

Еве неколку примери на наредби:

```
if (ti > jas) System.out.println("Cestitam. Ti pobedi. ");
throw new UserDefinedException("Kreiran e UserDefinedException! ");
MULTIPLY BROJ1 BY BROJ2 GIVING PROIZVOD.
```

Важно е да потенцираме (како и кај природните јазици) дека секоја низа (комбинација) знаци од абецедата не претставува збор од јазикот, односно секоја конструкција од зборови не претставува наредба. Според ова, и вишите програмски јазици, како и природните, си имаат своја **граматика** која ги содржи правилата за градење зборови и наредби.

Секој јазик содржи одредено множество **резервирани зборови** (англ. reserved words). Некои од нив имаат посебно значење за програмскиот јазик и тие се нарекуваат **клучни зборови** (англ. key words). Програмерите можат да користат и други зборови, различни од резервираните, кои се формирани по одредени правила. Тие се нарекуваат **идентификатори** (англ. identificators).

Постојат строги правила по кои се конструираат наредбите, кои се наречени **синтаксички правила** или само **синтакса** (англ. syntax) на јазикот. Со тие правила се проверува исправноста на секоја наредба во програмата.

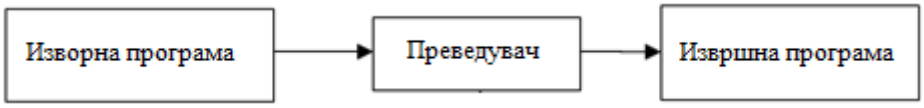
Секоја наредба во програмата мора да има точно дефинирано значење и да предизвика точно дефинирани дејства. Значењето (смислата) на наредбите е наречено **семаника** (англ. semantics) на јазикот.

Програмите напишани на виши програмски јазици не може директно да се извршуваат на компјутерот. За таа цел, се потребни посебни системски програми што ќе извршат нивно преведување на машински јазик, наречени **преведувачи** (англ. compilers). Секој виш програмски јазик има свој преведувач.

[7] Кај програмските јазици најчесто се користи англиската азбука.

Програмата напишана на виш програмски јазик се нарекува **изворна програма** (англ. source program), а преведената машинска програма се нарекува **извршна програма** (англ. executive program).

Преведувањето шематски е претставено на следнава слика.



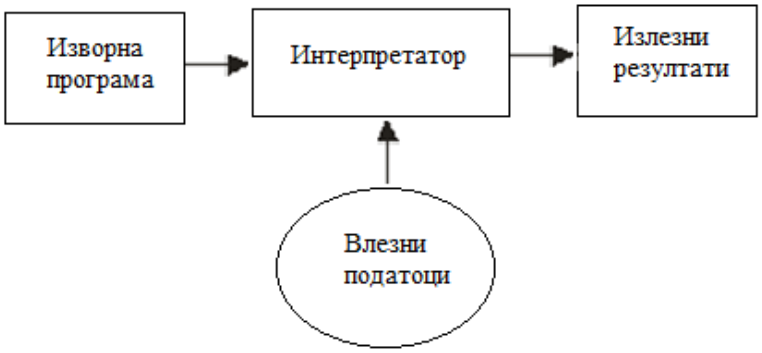
Слика 1.4. а

Извршната програма обично се запамтува како датотека на некоја надворешна меморија за оттаму да се повикува и да се извршува по потреба. При повикувањето, таа се вчитува во работната меморија на компјутерот и се извршува.

Честопати се користи и директно извршување на изворните програми, без да се преведуваат во извршни. Тоа се прави со други системски програми наречени **интерпретатори** (англ. interpreters).

Интерпретаторите не прават датотека во машинска форма на изворната програма, туку ги интерпретираат и ги извршуваат нејзините наредби кога таа се наоѓа во работната меморија.

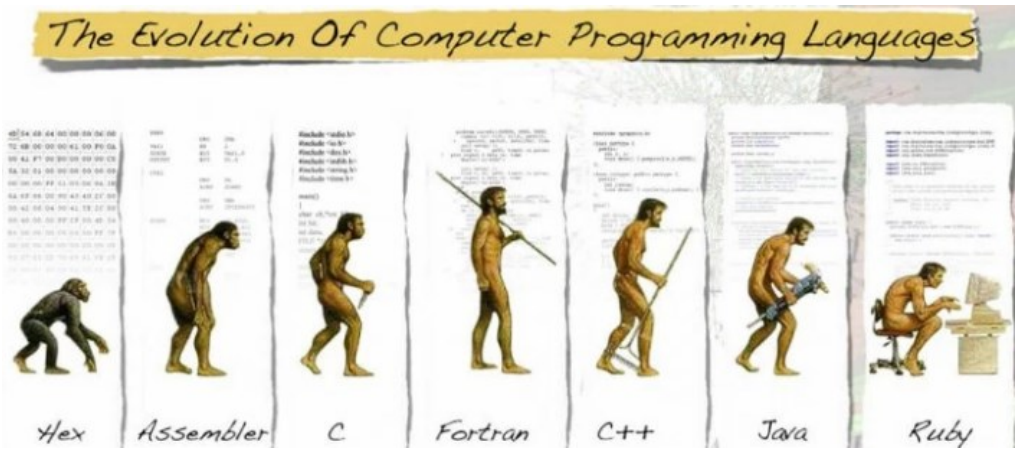
Шемата на интерпретирање на изворна програма е дадена на ***слика 1.4. б***.



Слика 1.4. б

Вишите програмски јазици не зависат од машината на која ќе се извршува некоја програма напишана со нив и затоа се нарекуваат **машински независни јазици**. Затоа, при градбата на вишите програмски јазици не се води сметка за видот (типот) на компјутерот на кој ќе се извршуваат програмите напишани со нив, туку се води сметка за видот на проблемот што се решава. Велиме дека тие се ориентирани кон решавањето на одреден вид проблеми (економски, технички, статистички итн.) и се нарекуваат **проблемски ориентирани јазици**.

Развојот на вишите програмски јазици започнал во 50-тите години од минатиот век. Нивниот развој фигуративно е претставен со следнава слика, преземена од литературата.



Во ***табела 1.4.1*** е дадено хронолошки појавувањето на неколку поважни програмски јазици.

Година	Јазик		
1945	Машински јазик		
1950	Симболички јазик		
1955	Fortran		
1960	Algol	Cobol	Lisp
1965	Basic	Simula	PL/I
1970	Pascal	Prolog	Logo
1975	Cobol	Scheme	ML
1980	C	Ada	Eiffel
1985	Objective-C	SQL	Perl
1990	C++	Visual Basic	Python
1995	Delphi	HTML	Ruby
	Java	JavaScript	PHP
2000	C#	Scala	R
2007	Go	Clojure	
2010	Swift	Google Dart	Julia
	Hack	Rust	Pixie
2012	TypeScript		
2016	Kotlin		

Табела 1.4.1

Денес, модерните програмски јазици развиени по 2010 година се наменети за програмирање на веб-апликации кои работат на интернет, а кои ги интегрираат добрите карактеристики на неколку јазици. Тие се **скриптни јазици** (англ. scripting languages), со кои се изработуваат посебни програми т.н. **скрипти** (англ. scripts) кои, пак, се извршуваат на веб-страниците, но при извршување на други програми. Скриптите не се преведуваат, туку се интерпретираат. Скриптните јазици се дизајнирани за интеграција и комуникација со други програмски јазици, како: HTML, Java, C++ итн.

Најпознати скриптни јазици се: JavaScript, PHP, Python, VBScript итн.

Најкористените програмски јазици за програмирање на веб се дадени на ***слика 1.4.4***.

Слика 1.4.4

На ***слика 1.4.5 а, б, в, г, д*** и ***ѓ*** се претставени едноставни програми за решавање на задачата за наоѓање на збирот на првите 10 природни броеви, напишана во C++, Java, Pascal, FORTRAN, Rubby и Python.

```
#include <iostream>
using namespace std;

void main()
{
        int n,suma;
        suma = 0;
        n = 1;
        while(n <= 10) {
                suma = suma + n;
                n = n + 1;
        }
        cout << suma << endl;
}
```

а

Програма во C++

```
public class JavaPrimer {
   public static void main(String[] args){
        int n,suma;
        suma = 0;
        n = 1;
        while (n <= 10) {
            suma = suma + n;
            n = n + 1;
        }
        System.out.println(suma);
   }
}
```

б

Програма во Java

```
PROGRAM Suma;
VAR  N,Suma:integer;
BEGIN
     Suma := 0;
     N := 1;
     WHILE N <= 10 DO
        BEGIN
                Suma := Suma + N;
                N := N + 1;
        END;
     WriteLn(Suma);
END.
```

в

Програма во PASCAL

```
      SUMA = 0
      N = 1
      WHILE N <= 10
             SUMA = SUMA + N
             N = N + 1
      ENDWHILE
      WRITE(*, 30)SUMA
 30   FORMAT(I10)
      STOP
```

г

Програма во FORTRAN

```
suma = 0
n = 1
while n <= 10
  suma = suma + n
  n = n + 1
end
puts suma
```

д

Програма во Ruby

```
suma = 0
n = 1
while n <= 10:
    suma = suma + n
    n = n + 1
print suma
```

ѓ

Програма во Python

Слика 1.4.5

Пишувањето на програмите се врши во некој уредувач на текст, а потоа тие се запишуваат (чуваат) на некоја надворешна меморија како датотека под некое име. Таквата програма не може да се изврши бидејќи е потребно претходно да се преведе.

Изворните програми најчесто имаат наставка по името на програмскиот јазик во кој е напишана, која е кратенка од програмскиот јазик одделена од името на програмата со точка: .pas, .bas, .for, .cpp, .java, итн. На пример: zbir.pas, prva.bas, mojaPrograma.cpp, podredi.java итн.

Извршните програми кои се добиваат по преведување на изворните програми имаат наставка: .exe, .com, .bat, .bin, .dll итн. На пр.: zbir.exe, prva.com итн.

Генерации програмски јазици

Според историскиот развој, програмските јазици се делат на пет генерации (англ. Generation Languages – GL), и тоа:

- Прва генерација (1 GL): машински јазик.
- Втора генерација (2 GL): симболички јазици.
- Трета генерација (3 GL): виши програмски јазици (C++, Java, Fortran и др.).
- Четврта генерација (4 GL): многу виши програмски јазици (англ. very high-level languages) (Perl, PHP, Python, SQL и др.).
- Петта генерација (5 GL): високи програмски јазици[8] (Mercury, Prolog, OPS5 и др.).

Поделба на програмските јазици

Поделба на програмските јазици може да се направи и според начинот на кој се обработуваат податоците:

- **Императивни**.
- **Декларативни**.

[8] Некои автори ги нарекуваат јазици за вештачка интелигенција (англ. artificial intelligence languages).

Императивниот начин е наредбен начин, при кој програмите се пишуваат со наредби за обработка на податоците.

Декларативниот начин е описен начин, кој опишува како да се дојде до резултатот.

Императивните јазици се делат на:

- **процедурални** (Pascal, C, Fortran, Basic, Ada, Modula и други),
- **објектно ориентирани** (C++, Java, C#, Delphi, Smalltalk и други).

Главна карактеристика на овие јазици е користењето на променливи, наредби и процедури[9]. Во процедуралните јазици, процедурите со кои се обработуваат податоците се одвоени, а во објектно ориентираните јазици процедурите (наречени функции или методи) се скриени во самите објекти.

Декларативните јазици се делат на:

- **функциски** (Lisp, Scheme, Miranda, Haskell, ML и други),
- **логички** (Prolog и други).

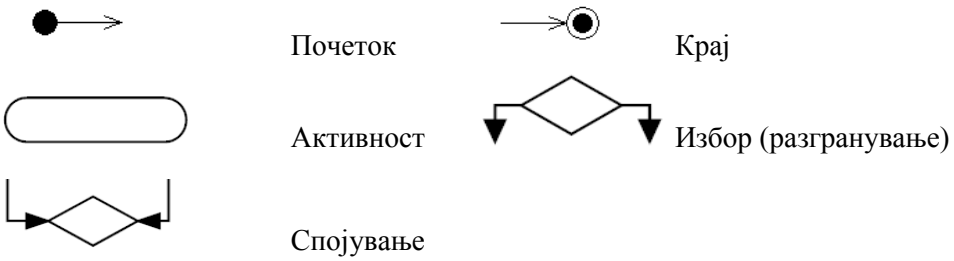
Главна карактеристика на функциските јазици е користење на изрази и функции. Програмата претставува функција (или група функции) составена од попрости функции.

Во логичките јазици, основна единица за пресметување е релацијата, што е поопшта од пресликувањето. Програмата се состои од: множество факти, множество дозволени заклучоци (правила за докажување), методи за заклучување и цел на доказот.

Дијаграм на активности

За графичко претставување на работата на програмите, се користат разни дијаграми. За унифицирано претставување на дијаграмите, кон крајот на 90-тите години од минатиот век е создаден графичкиот јазик наречен Unified Modeling Languga – UML.

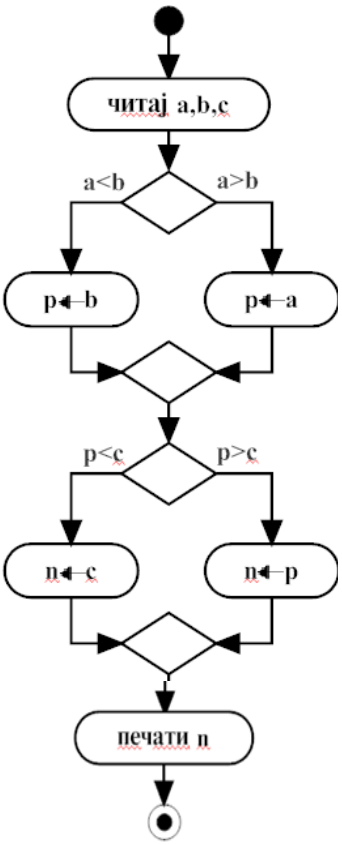
UML има голем број графички симболи. Ние ќе ги користиме следните:



Дијаграмот од ***слика 1. .*** , изразен со UML-јазик, ќе биде како на ***слика 1.4.6***.

[9] За променливи и процедури ќе зборуваме подоцна.

Прашања за проверка на знаењето



Слика 1.4.6

1. Што е програмирање?
2. Како се нарекуваат луѓето што програмираат?
3. Објаснете ги фазите за решавање на некоја задача со помош на компјутер.
4. Што треба да знаете за да можете да пишувате програми?
5. За кои програмски јазици сте слушнале?
6. Со кои цифри се запишани програмите во машинскиот јазик? Зошто не се програмира во машински јазик?
7. Во кој броен систем се запишуваат машинските програми на хартија?
8. Дали е поедноставен машинскиот јазик или симболичките јазици?
9. Како се нарекуваат системските програми што имаат задача да ги преведуваат програмите од симболички на машински јазик?
10. На кои факти се должи масовното користење на вишите програмски јазици?
11. Има ли разлика меѓу азбуката на природен и на програмски јазик?
12. Дали зборовите во вишите програмски јазици имаат значење?
13. Од што се состои речникот на еден виш програмски јазик?
14. Може ли една програмска наредба да се состои само од еден збор?
15. Како се нарекуваат зборовите што ги формираат програмерите во програмите?
16. Како се нарекуваат правилата по кои се конструираат наредбите во програмските јазици?
17. Што е семантика на јазикот?
18. Како се нарекува програмата напишана на виш програмски јазик, а како програмата преведена на машински јазик? Кои наставки ги имаат соодветните датотеки?
19. Која е разликата помеѓу преведувачите и интерпретаторите?
20. Зошто велиме дека вишите програмски јазици се машински независни јазици?
21. Наведете неколку виши програмски јазици.
22. Наведете ги генерациите на вишите програмски јазици.

23. Наведете една поделба на вишите програмските јазици.
24. Која е главната карактеристика на процедуралните јазици, а која на објектно ориентираните јазици?
25. Со кој графички јазик денес се претставуваат дијаграмите? Нацртајте ги неговите основни графички симболи.

1.5 Интегрирана развојна околина

Според кажаното во претходните потточки, за да можеме да почнеме со програмирање, мора да се запознаеме со некој виш програмски јазик. Потоа, ја пишуваме програмата во некој уредувач на текст и ја зачувуваме како текстуална датотека.

За да може да видиме дали програмата што сме ја напишале го прави она за што е напишана, треба да ја преведеме во машински јазик и да ја извршиме. Рековме дека при преведување на машински јазик, се користи некој преведувач и се добива извршна програма/датотека. Добиената извршна програма треба да ја извршиме и дури тогаш да видиме што прави нашата програма. Ако користиме интерпретатор, тогаш програмата директно се преведува и извршува, т. е. се интерпретира.

За да им се олесни работата на програмерите, сите програми кои ги извршуваат горенаведените функции се стават (интегрираат) во една околина, наречена ***интегрирана развојна околина*** ***ИРО*** (англ. Integrated Development Environment – IDE). За програмирање во C++, повеќе се користат интегрираните развојни околини, како: Microsoft Visual Studio, Code::Blocks, Netbeans, Eclipse, Visual Studio Code, CLion, CodeLite и други.

Секоја интегрирана развојна околина е составена од множество апликации кои се користат при програмирање. Тоа множество апликации зависи од програмскиот јазик, а задолжителни апликации се: **текстуален уредувач** (англ. text editor), **преведувач** (англ. compiler), **дибагер** (англ. debugger) и **поврзувач** (англ. linker).

Текстуален уредувач

Тоа е апликација која овозможува внесување текст од тастатурата директно во изворната програма, како и за извршување на основните операции за обработка на текстови. Таа дава можност за зачувување на програмите на диск и отворање на зачуваните програми за повторна обработка, т. е. за измена.

Уредувачот на текст ги има сите стандардни алатки за внесување и за обработка на текст. При уредувањето на текстот, можеме да ги користиме речиси истите наредби кои ги имаме во програмите за уредување на текст за канцелариско работење, како: Microsoft Office Word, OpenOffice Writer и др. Исто така, важат сите команди за движење на покажувачот од глувчето.

Командите за селектирање, копирање, пренесување и бришење текст се исти како во останатите уредувачи на текст. Притоа, се користат командите од менито за уредување, најчесто наречено Edit.

Во многу случаи, може да има потреба од барање на одреден текст низ програмата, а по наоѓањето, и од негово заменување со друг текст. Тоа се реализира со командите во менито, најчесто наречено Search.

Преведувач/Интерпретатор

Преведувачот е специјална апликација која ја преведува изворната програма (напишана во некој програмски јазик) во извршна програма на машински јазик за да може компјутерот да ја изврши.

При преведувањето, преведувачот ја преведува целата изворна програма во извршна програма, која може веднаш да се изврши (ако е во работната меморија) или да се зачува во датотека на некоја надворешна меморија и подоцна (или веднаш) да се изврши, без повторно да се преведува.

Интерпретаторот ја преведува и извршува програмата дел по дел (наредба по наредба), т. е. по неколку наредби одеднаш. При повторно извршување на програмата, интерпретаторот мора повторно да ја преведе и изврши секоја наредба (или група наредби).

И преведувачите и интерпретаторите имаат предности и недостатоци.

Денес, некои програмски јазици, како Java и Visual Basic, користат комбинација од преведување и интерпретирање. На пример, јава-преведувачот ја преведува изворната програма во т.н. меѓукод, познат под името **бајт-код** (англ. bytecode). Програма во бајт-код може да се изврши со интерпретатор на кој било компјутер.

Слично се прави и при преведување на програма во C++, само што преведувањето на изворната програма во меѓукод се третира како фаза на претпроцесирање. Потоа, добиената форма на програмата се преведува во машински јазик, ***слика 1.5.1***.

Преведувачите и интерпретаторите се дизајнирани да проверуваат грешки во програмата. Најчести се јазичните (синтаксичките) грешки кога наредбите се неправилно напишани. На пример, ако наместо close, сме напишале cloce. Се јавуваат и програмски (семантички) грешки при неправилно користење на наредбите на јазикот. На пример, ако наместо do {... }while(), сме напишале do {... }for()и сл.

Дибагер

За жал, преведувачите можат да откријат грешки во програмата поради неправилно користење на програмскиот јазик, било синтакички, било семантички. Друг тип на грешки кои често се случуваат, особено од програмерите почетници, се логичките грешки, т. е. програмата не го прави тоа за што е напишана. Тие грешки многу тешко се откриваат и затоа треба да се провери (тестира) начинот на кој се извршуваат програмите со претходно подготвени влезни податоци за кои резултатот е познат. Тие влезни и излезни податоци се нарекуваат ***тест-примери***.

Дибагер е апликација која помага при барање на логичките грешки. Таа овозможува да го следиме извршувањето на програмата на тест-примерите чекор по чекор и, на тој начин, да ги откриеме логичките грешки. За време на тој процес, дибагерот може да покажува и некои меѓурезултати кои помагаат да се определат местото и видот на грешката.

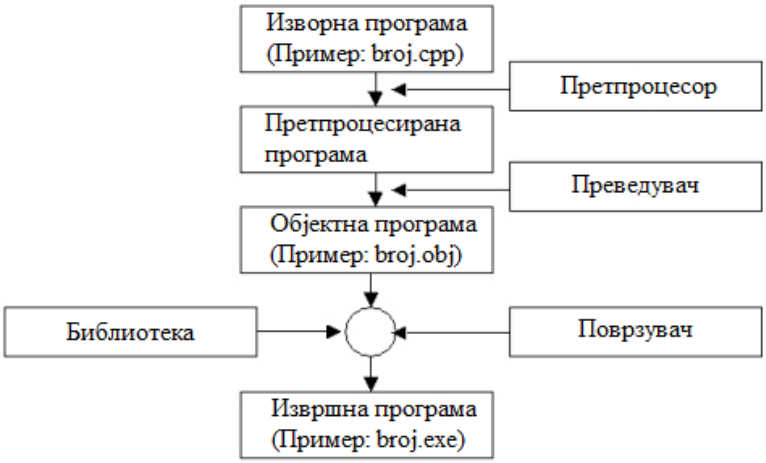
Поврзувач

Понекогаш програмата е премногу голема за да се напише како една датотека. Освен тоа, различните делови може да се пишуваат од различни програмери. Некои делови од една програма може да бидат искористени и во друга програма. Затоа, овие делови треба да бидат напишани и зачувани како самостојни датотеки. Ако некоја програма треба да биде составена од повеќе самостојни програми, тие треба да се обединат со поврзување во единствена датотека, т. е. во единствена извршна програма. Тоа се прави со посебна апликација наречена **поврзувач**.

Друга улога на поврзувачот е да ги „поврзе“ библиотечните програми со програмата што ја пишуваме. Секој програмски јазик има библиотеки (наречени и модули) со готови програми, кои може да се користат во која било програма што ја пишуваме. На пример, библиотечна програма за пресметување на квадратниот корен од реален број x, во многу програмски јазици е наречена sqrt(x). Ако во нашата програма има наредба y = sqrt(x), тогаш поврзувачот ја поврзува нашата програма со библиотеката во која се наоѓа програмата sqrt(x). (Името sqrt е кратенка од square root).

Пореална шема на преведување на изворна програма во извршна, од онаа дадена на ***слика 1.4. а*** и ***б***, е шемата од ***слика 1.5.1***.

Интегрираната развојна околина вообичаено изгледа како множество од повеќе апликации со кои секојдневно работиме. Таа има основен прозорец со *работна област*, лента за менијата и други вообичаени елементи: копчиња за затворање и промена на големината на прозорецот, копчиња за минимизирање итн. Во *работната област* се отвораат прозорците на одделните програми. Во лентата за менија се наоѓаат паѓачките менија (листи од наредби), со кои се пови-



Слика 1.5.1

куваат функциите на околината: мени за уредување, мени за преведување, за поврзување, за извршување на програмите итн. Во повеќето интегрирани околини за програмирање вообичаено има мени за помош кое дава можност за прегледи поврзани со правилата на програмскиот јазик и работа со самата околина за програмирње.

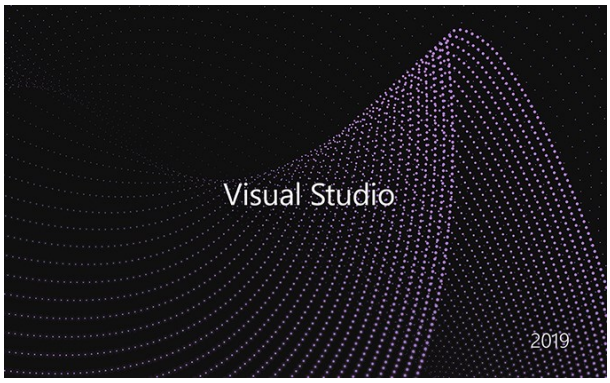
Во продолжение, ќе ги објасниме интегрираните развојни околини Microsoft Visual Studio 2019 и Code::Bloks.

Интегрираната развојна околина Microsoft Visual Studio 2019

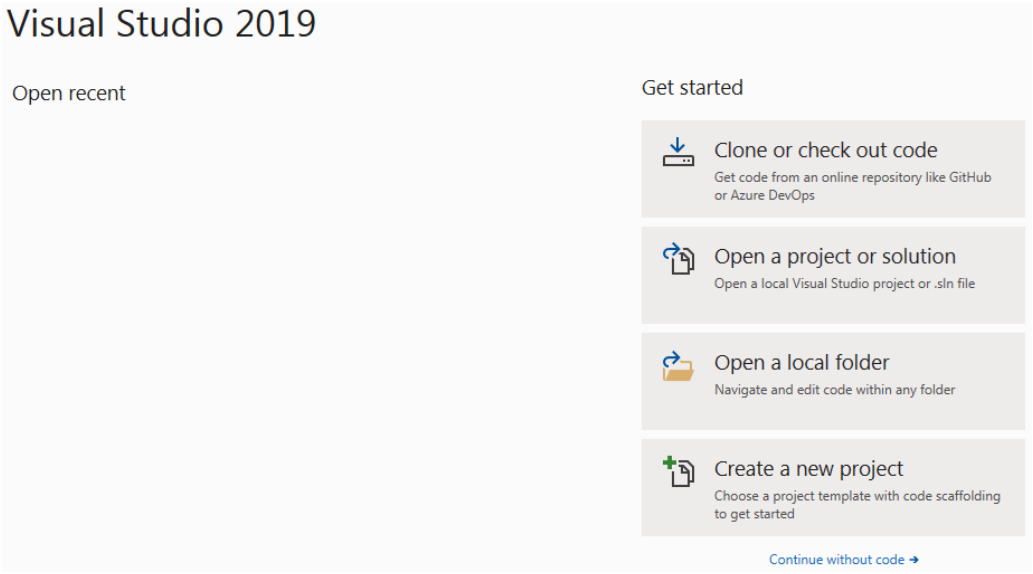
Microsoft Visual Studio 2019 (MVS 2019) може да се преземе од повеќе интернет-страници.

По инсталирањето и стартувањето, ќе се добие најавната слика на MVS 2019.

Потоа, се јавува мени за избор на начинот на стартување на MVS 2019, ***слика 1.5. .***



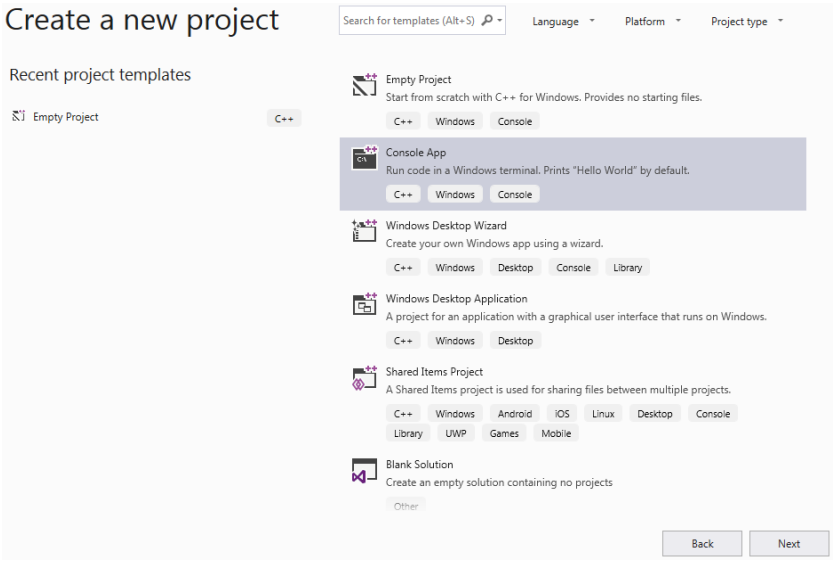
Слика 1.5.2



Слика 1.5.

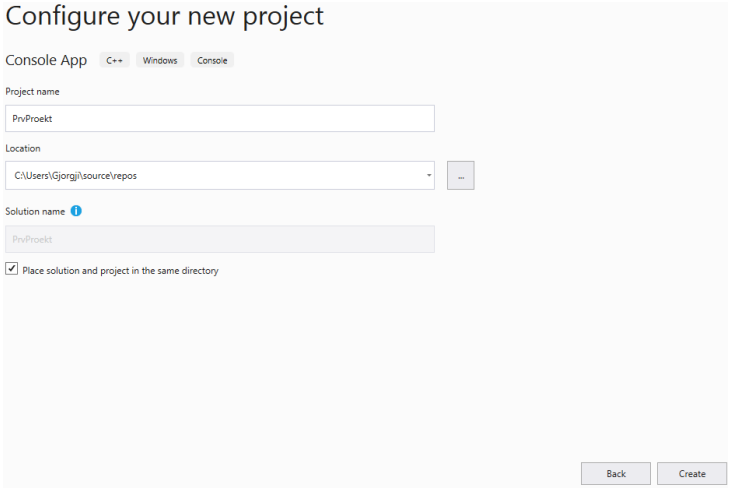
Почетниците најчесто ги избираат вториот и четвртиот начин на стартување.

Со избор на четвртиот начин, се отвора менито за креирање нов проект, ***слика 1.5.4***. За почетниците е најдобро да креираат празен проект (Empty Project), или конзолна апликација (Console App), при што се прикажува и пример на програма во C++. Потоа, се кликнува на копчето Next.



Слика 1.5.4

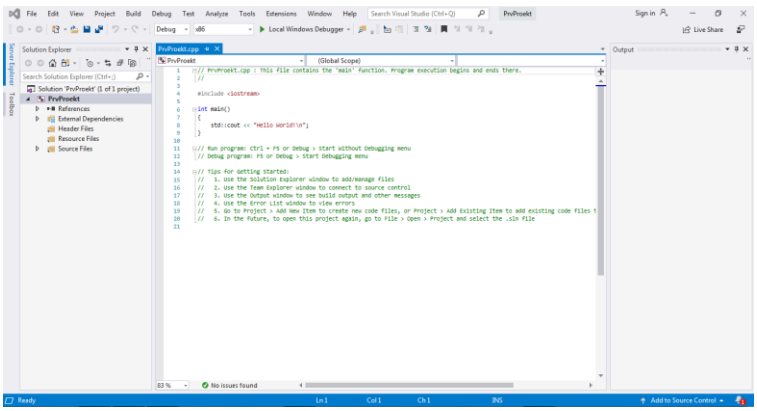
Се отвора прозорецот за конфигурирање на нов проект, ***слика 1.5.5***.



Слика 1.5.5

Во него ќе го внесеме името на новиот проект, на пример PrvProekt и ќе го означиме (со кликнување на него) квадратчето пред Place solution and project in the same directory.

По кликнување на копчето Create, започнува креирање на проектот и се отвора прозорецот од интегрираната развојна околина на MVS 2019, ***слика 1.5.6***. Во него се прикажува и програма во C++ под истото име како и проектот, PrvProekt.cpp. Над и под програмата има повеќе објаснувања, кои овде нема да ги разгледуваме.



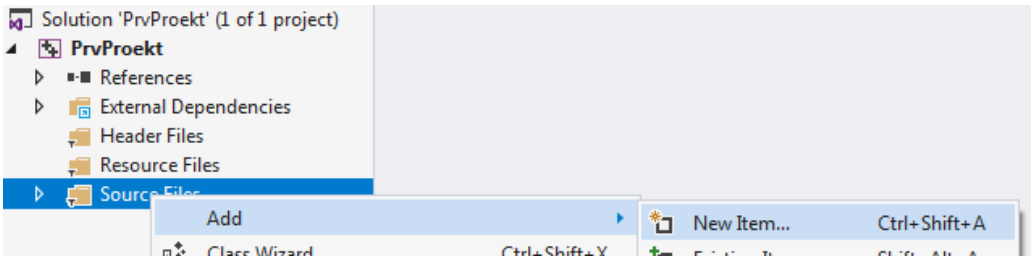
Слика 1.5.6

Ако не сакаме да се појавува програмата, тогаш од менито на ***слика 1.5.4*** треба да избереме Empty Project.

Програмата PrvProekt.cpp се затвора со кликнување врз знакот x по името.

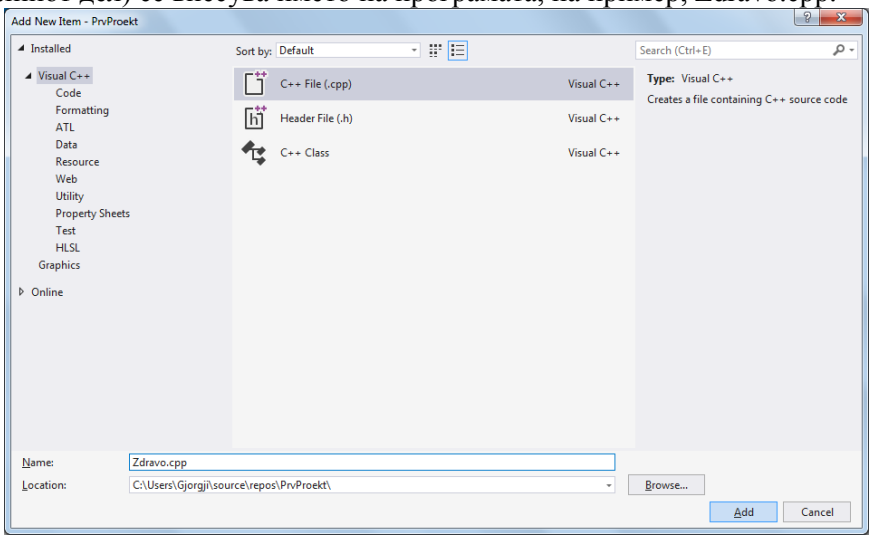
Во прозорецот Solution Explorer се прикажуваат проектите креирани во ИРО на MVS 2019. Така, на ***слика 1.5.6*** се појавува директориум на проектот PrvProekt со неговите поддиректориуми.

Со десен клик врз поддиректориумот Source Files од прозорецот Solution Explorer и избор на командата New Item… од подменито Add (***слика 1.5.***), се отвора прозорецот Add New Item – PrvProekt, ***слика 1.5.8***.



Слика 1.5.

Во десната рамка се кликнува на C++ File (.cpp), а потоа во полето Name (на долниот дел) се внесува името на програмата, на пример, Zdravo.cpp.



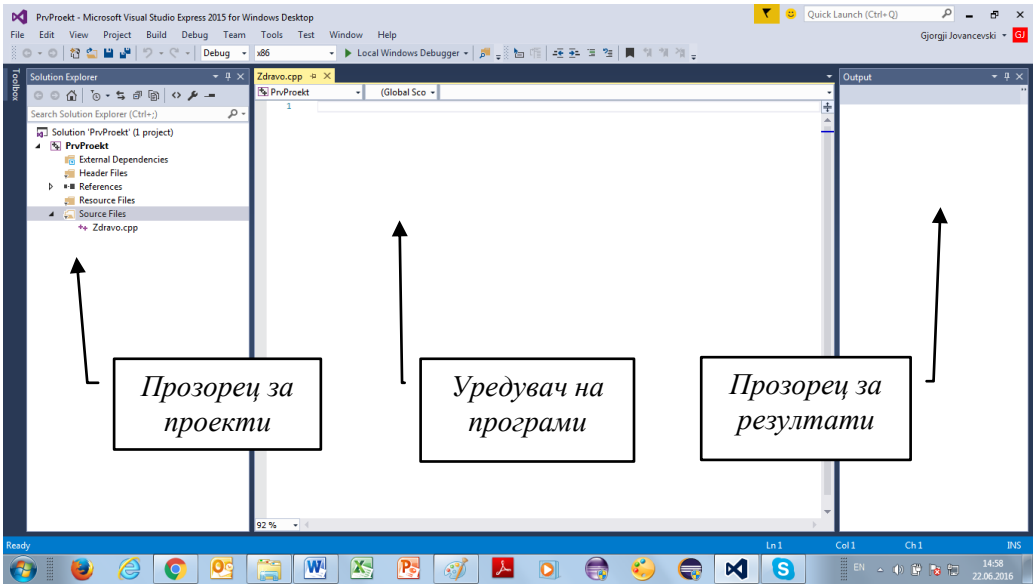
Слика 1.5.8

Со кликнување на копчето Add, се отвора прозорецот за пишување и уредување на програмата Zdravo.cpp, ***слика 1.5.9***.

Главен прозорец

На ***слика 1.5.9*** е претставен главниот прозорец, т. е. основната околина за програмирање во MVS 2019. Таа ги содржи следните прозорци:

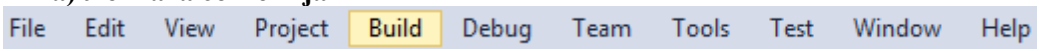
- Прозорецот со проекти (Solution Explorer).
- Прозорецот за пишување и уредување на програмите – уредувач.
- Прозорецот за прикажување на резултати – излезен прозорец (Output).



Слика 1.5.9

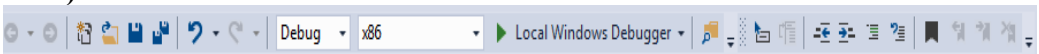
Основните елементи на главниот прозорец се:

а) Лентата со менија



Лентата со паѓачки менија се наоѓа во најгорниот дел на прозорецот, веднаш под неговиот наслов. Во неа се наоѓаат менијата: File, Edit, View, Project, Build, Debug, Tools и други. Секое мени содржи команди за околината и/или други менија.

б) Лентите со алатки



Лентите со алатки (копчиња за стартување на најчесто користените команди во околината) се наоѓаат непосредно под лентата со паѓачки менија. Алатките во околината за програмирање се активни или неактивни (бледи). Алатките преку кои се задаваат повеќе команди десно до нив имаат стрелка надолу и со кликну-

вање на неа, се отвора паѓачко мени. Ако се кликне на самата алатка, тогаш се отвора прозорец преку кој се задаваат истите команди.

Прва C++ програма

Во уредувачот од ***слика 1.5.9*** можеме да ја напишеме и уредиме програмата Zdravo.cpp, ***слика 1.5.10***.

Со командата Compile од подменито Build (или со Ctrl + F7), се преведува програмата и во излезниот прозорец (Output) добиваме порака (***слика 1.5.11***) дали програмата е успешно преведена.

Zdravo.cpp

PrvProekt

```
1   #include <iostream>
2   using namespace std;
3
4   int main() {
5
6         // Mojata prva programa Zdravo.cpp
7
8      cout << "Zdravo, kako ste?" << endl;
9
10     system("pause");
11     return 0;
12  }
```

Слка 1.5.10

Output

Show output from: Build

```
1>------ Build started: Project: PrvProekt, Configuration: Debug Win32 ------
1>Zdravo.cpp
========== Build: 1 succeeded, 0 failed, 0 up-to-date, 0 skipped ==========
```

Слка 1.5.11

Со командата Start Without Debugging од подменито Debug (или со Ctrl + F5), се извршува програмата и резултатот се прикажува во прозорецот од ***слика 1.5.12.***

C:\Users\Gjorgji\source\repos\PrvProekt\Debug\PrvProekt.exe

```
Zdravo, kako ste?
Press any key to continue . . .
```

Слка 1.5.12

За затворање на прозорецот со резултати, се притиска кое било копче од тастатурата.

За да ја зачуваме програмата на диск, ја задаваме командата Save од подменито File (или Ctrl + S). За да ја зачуваме програмата на диск под друго име, ја задаваме командата Save As од подменито File.

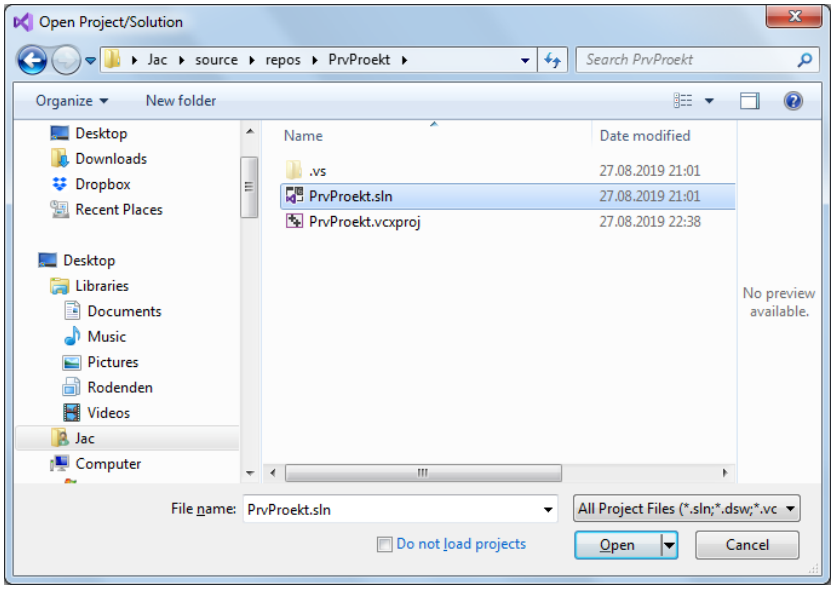
Во еден проект може да има повеќе програми запишани како датотеки. Знаеме дека секоја датотека има име и наставка. Изворните програми напишани со јазикот C++ имаат наставка .cpp. Значи, име на некоја програма во C++ може да биде: P1.cpp, prog.cpp, PrvaPrograma.cpp, mojaPrograma. cpp итн.

Вежби

Вежба 1.5.1

а) Програмата Zdravo.cpp дополнете ја да изгледа како на ***слика 1.5.15***.

При стартувањето на MVS 2019 по вторпат (и секој наредент пат), се појавува страницата на MVS 2019, ***слика 1.5.*** . Ако сакаме да отвориме некој постоен проект, треба да кликнеме на линкот Open a project or solution. Ќе се отвори прозорецот од ***слика 1.5.1*** за избор на проектот.

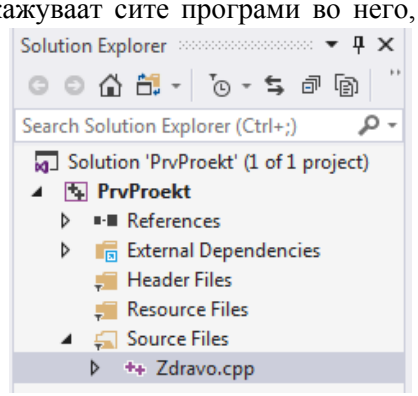


Слика 1.5.1

Со кликнување на некој од проектите, на пример PrvProekt.sln, тој се отвора во прозорецот Solution Explorer и се прикажуваат сите програми во него, ***слика 1.5.14***.

Со двојно кликнување врз името на некоја програма (на пример, на Zdravo.cpp), таа ќе се отвори во уредувачот. Во него ја дополнуваме програмата Zdravo.cpp, како што е прикажано на ***слика 1.5.15***.

б) Зачувајте ја програмата со командата Save (или Ctrl + S).

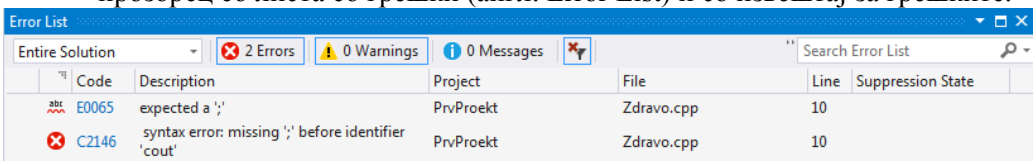


Слика 1.5.14

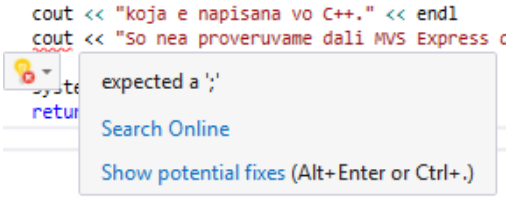
```
1   #include <iostream>
2   using namespace std;
3
4   int main() {
5
6       // Mojata prva programa Zdravo.cpp
7
8       cout << "Zdravo, kako ste?" << endl;
9       cout << "Ova e primer za programa" << endl;
10      cout << "koja e napisana vo C++." << endl;
11      cout << "So nea proveruvame dali MVS 2019 dobro raboti." << endl;
12
13      system( "pause" );
14      return 0;
15  }
```

Слика 1.5.15

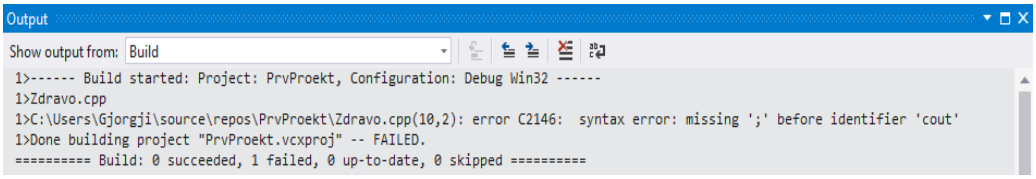
в) Ако при пишување на програмата сте погрешиле некоја наредба, при нејзиното преведување, ќе се појави порака за грешки во излезниот прозорец. На пример, ако сме заборавиле да ставиме точка и запирка на крајот од последната наредба, при преведување на програмата, ќе се отвори прозорец со листа со грешки (англ. Error List) и со извештај за грешките.



Забележувате дека наредбата по онаа во која се случила грешката е потцртана со брановидна линија. Ако го доведеме покажувачот врз линијата, ќе се прикаже рамка со порака за грешката.



Во излезниот прозорец (Output) се јавува порака за видот и местото на грешката. По ставање точка и запирка, програмата ќе работи коректно.



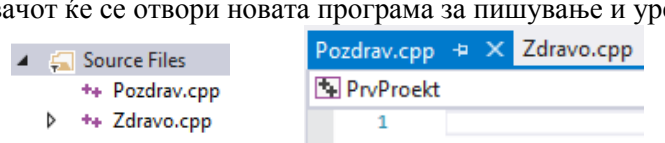
г) Линиите во програмата кои започнуваат со две коси црти //, служат за внесување коментари во програмата и тие не се извршуваат. Добро е се-

која програма на почетокот да има коментар за тоа што работи. Ова ќе го практикуваме и ние.

Вежба 1.5.2

а) Креирајте уште една програма во проектот PrvProekt, на пример, под име Pozdrav.cpp.

Програмата се креира со истата постапка како и програмата Zdravo.cpp, т. е. со десен клик врз Source Files и избор на Add ▶ New Item… Во прозорецот што ќе се отвори (Add New Item – PrvProekt) се кликнува на C++ File (.cpp) и во полето Name (на долниот дел од прозорецот) се внесува името Pozdrav.cpp. На крајот, се кликнува на копчето Add. Во прозорецот Solution Explorer, во поддиректориумот Source Files ќе се прикаже името на новата програма. Исто така, во уредувачот ќе се отвори новата програма за пишување и уредување.

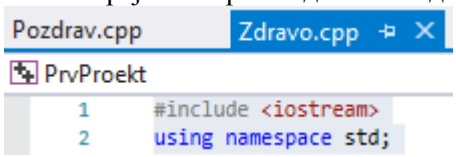


Гледаме дека и програмата Zdravo.cpp е отворена во уредувачот.

Преминување од една во друга програма во уредувачот се врши со кликнување врз името на програмата.

За да затвориме некоја програма отворена во уредувачот, се задава командата Close од подменито File или се кликнува врз знакот x десно од името на програмата.

б) Кликнете врз насловот од програмата Zdravo.cpp и таа ќе се отвори. Селектирајте ги првите две команди,



ископирајте ги со Ctrl + C, кликнете врз името на програмата Pozdrav.cpp и поставете ги со Ctrl + V.

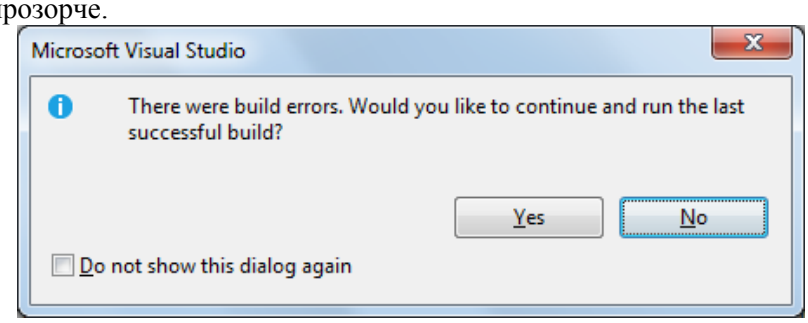
Оставете една празна линија со Enter и напишете ја главната функција со команди за коментар и за печатење на пораки, ***слика 1.5.16***.

```
1   #include <iostream>
2   using namespace std;
3
4   int main() {
5       // Mojata vtora programa vo C++
6
7       cout << "Moeto ime e ..." << endl;
8       cout << "Moeto prezime e ..." << endl;
9       cout << "Mojot rodenden e na ..." << endl;
10
11      system("pause");
12      return 0;
13  }
```

Слика 1.5.16

Зачувајте ја програмата со Ctrl + S, преведете ја со Ctrl + F7 и извршете ја со Ctrl + F5.

Ќе забележите дека при извршување, се јавува порака за грешка во следното прозорче.

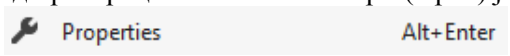


[10]

Пораката се однесува на тоа дека грешката се јавила при градењето на проектот. Во еден проект може да има повеќе програми кои се поврзуваат во една извршна програма (велиме дека „се гради проектот“), но мора да има само една програма со која започнува извршувањето на проектот. Тоа е програмата која ја содржи функцијата main() и се нарекува **апликација** (англ. application). Бидејќи во нашиот проект PrvProekt има две програми кои ја содржат функцијата main(), се јавува наведената грешка.

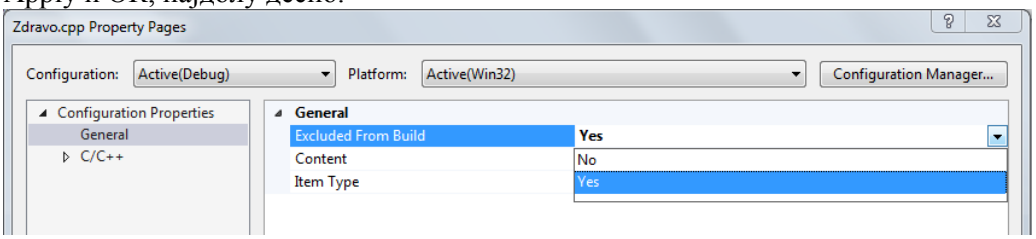
Ова се решава на различни начини, а еден најлесен е да се исклучат од проектот сите програми кои ја содржат функцијата main(), освен онаа програма која сакаме да биде апликација и од која ќе започне извршувањето на проектот.

За да ја исклучиме програмата Zdravo.cpp, ќе кликнеме врз неа со десен клик и од прозорецот што ќе се отвори (Open) ја задаваме командата:

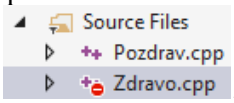


[10] Ако се кликне на Yes, ќе се изврши последната успешно извршена програма. Ако се кликне на No, ќе се изврши последната неуспешно извршена програма.

Ќе се отвори прозорецот Zdravo.cpp Property Pages и во менито Excluded From Build од рамката General се избира Yes. Потоа, се кликнува на копчињата Apply и OK, најдолу десно.



Во прозорецот Solution Explorer ќе забележете дека знаците ++ пред името Zdravo.cpp се сменети во +–.

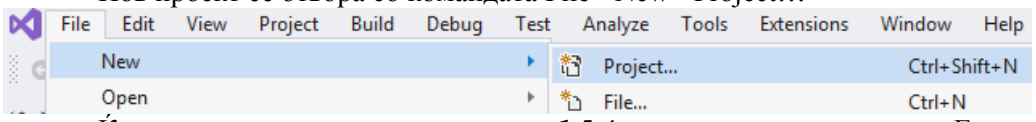


Тоа значи дека програмата Zdravo.cpp е исклучена од проектот и дека само програмата Pozdrav.cpp е апликација и може да се изврши.

Вежба 1.5.3

а) Креирајте нов проект (на пример, VtorProekt) и во него една програма.

Нов проект се отвора со командата File ▸ New ▸ Project…



Ќе се отвори прозорецот од ***слика 1.5.4*** и се кликнува на линкот Empty Project или Console App, како што е објаснето кај ***слика 1.5.4*** и ***слика 1.5.5***. Во полето Name од ***слика 1.5.5*** (на долната страна) се пишува името на проектот, на пример, VtorProekt. На крајот, се кликнува на копчето Create.

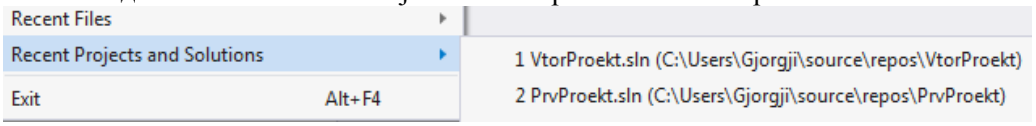
Во прозорецот Solution Explorer ќе се отвори вториот проект, а првиот ќе се затвори.

Програма во проектот се отвора, според постапката наведена во ***Вежба 1.5.2 а.***

Ако сакаме да преминиме во првиот проект (или во некој друг отворен проект), него го избираме од листата со креирани проекти со командите:

File ▶ Recent Projects and Solutions

Од листата што ќе се појави го избираме саканиот проект.



Интегрираната развојна околина Code::Blocks

Интегрираната развојна околина Code::Blocks може да се најде на интернет на адресата: http://www.codeblocks.org/downloads.

Во централниот дел на страницата има три линка: **Download the binary release, Download the source code** и **Retrieve source code from SVN**. За наједноставна инсталација, се препорачува да се избере првиот линк **Download the binary release**. По кликнувањето на овој линк, се отвора нова страница на која има понудено преземање на Code::Blocks за оперативниот систем под кој работи Вашиот компјутер. За почетниците се препорачува да ја симнат верзијата која вклучува MinGW setup. Во времето на пишување на оваа книга, тоа е линкот codeblocks-20.03mingw-setup.exe кој е наменет за корисниците на сите оперативни системи Windows. По кликнување на локацијата од каде што сакате да го преземете Code::Blocks (на приме **Sourceforge.net**), се отвора нова страница, која по истекот на 5 секунди сама нуди опција да се зачува датотеката на локација избрана од корисникот. По зачувувањето на датотеката, треба да се следат инструкциите за инсталирање.

Стартување

При инсталацијата на Code::Blocks во менито Programs од основното мени Start, се креира група на програми со име CodeBlocks. За да се стартува интегрираната околина за програмирање, треба да се избере CodeBlocks од оваа група.

Главен прозорец

Во главниот прозорец (***слика 1.5.1***) на CodeBlocks има алатки за зачувување, пребарување, преведување и извршување на програми, алатки за дибагирање итн. Основните елементи ќе ги објасниме во продолжение.

а) Лента со менија

Лентата со менија се наоѓа во најгорниот дел на прозорецот, веднаш под неговиот наслов. Во неа се наоѓаат менијата: File, Edit, View, Search, Project, Build, Debug, Fortran, wxSmith, Tools, Tools+, Plugins, DoxyBlocks, Settings и Help. Секое мени содржи наредби за околината и/или други менија. Значењето на повеќето од нив ќе биде разгледано понатаму.

б) Ленти со алатки

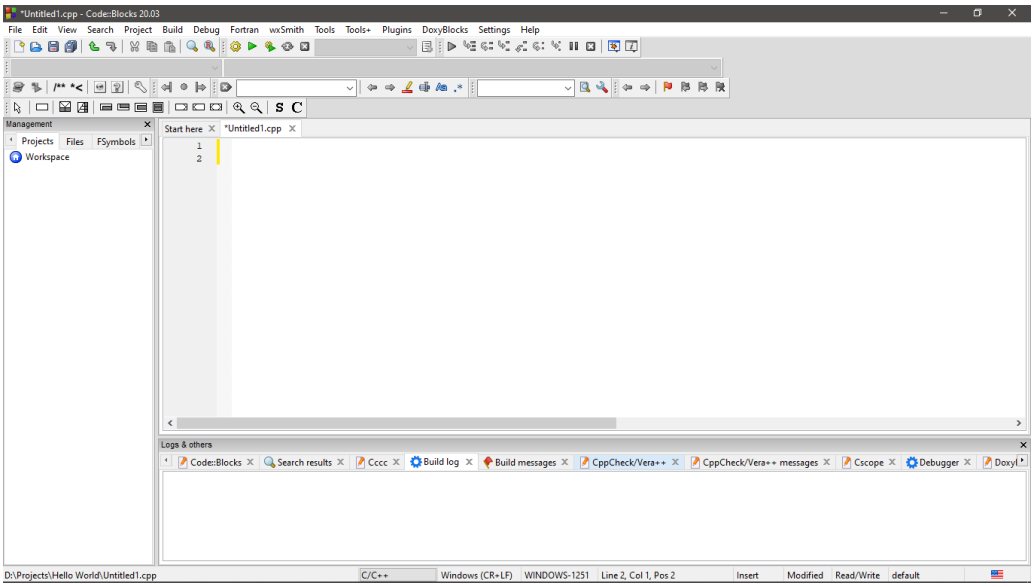
Лентите со алатки (копчиња за стартување на најчесто користените команди на околината) се наоѓаат непосредно под лентата со паѓачки менија. Алатките во околината за програмирање Code::Blocks се поделени во групи, во завис-

ност од нивното значење. Покажувањето и криењето на различните групи алатки, се реализира со избирање на опцијата View ▸ Toolbars. Секоја група има свое име и ако пред името има знак за штиклирање ✓, групата се гледа на екранот, а ако нема, таа не се гледа. Секоја алатка соодветствува на некои од командите на менито.

Во главниот прозорец се наоѓаат прозорците на апликациите на околината. Најважните од нив се:

- Прозорецот за уредување на текст – уредувач. Во него може да има неколку програми, во зависност од тоа во колку програми работи програмерот во даденият момент. Името на програмата е прикажано во горниот дел на прозорецот за уредување. Името на програмата која е активна во моментот е различно означено од имињата на другите програми.
- Прозорецот за резултати – излезен прозорец. Во него програмите даваат информација за својата работа при преведување, поврзување, стартување итн. Ако при преведувањето бидат откриени грешки, тогаш тие се покажуваат во рамката Build messages на овој прозорец.
- Прозорецот за организација на работата на програмата.

Прозорците може да се затвораат, да се зголемуваат или намалуваат итн.



Слика 1.5.1

Уредувач на текст

а) Внесување и обработка на текст

Уредувачот на текст ги содржи сите стандардни алатки за внесување и обработка на текст. При уредувањето на текстот, можеме да ги користиме речиси истите опции кои ги имаме во програмите за уредување на текст за канцелариско работење (како Microsoft Office Word, OpenOffice Writer и др.). Исто така, важат сите опции за движење на покажувачот на глувчето.

б) Селектирање, копирање и преместување на текст

Овие операции се задаваат преку командите на менито Edit.

в) Барање и заменување

Во многу случаи, може да има потреба од барање на одреден текст низ програмата, а по наоѓањето, и од негово заменување со друг текст. Тоа се реализира со командите во менито Search, како Find, Replace итн.

Преведување и извршување на програми

Менито Build содржи команди за преведување и извршување на програми. Програмата се преведува со командата Build ▸ Build (или Ctrl + F9). Како резултат во папката во која се наоѓа датотеката со изворната програма, се креира соодветна извршна програма. Извршната програма се стартува со командата Run (или F9).

При извршување на програмата, се отвора прозорец во кој се внесуваат влезните податоци и се прикажуваат резултатите.

Датотеки создадени од околината

- Датотеките со наставка .cpp ги содржат изворните програми и се добиваат по извршувањето на командата File ▸ Save или Ctrl + S.
- Датотеките со наставка .exe содржат извршна програма еквивалентна на изворната. Тие се креираат при преведување на изворната програма.

Следење на извршувањето на програмата

Следењето на извршувањето на програмата е многу важно за барање на логички грешки во програмата. Околината Code::Blocks дава можност за такво следење. За оваа цел, може да се одредат линии во текстот (наречени контролни точки), во кои сакаме да се стопира извршувањето на програмата и да се прикаже содржината на податоците кои сакаме да ги следиме. Кога започнуваме со извршување на програмата во режим на следење, таа работи нормално додека не

стигне до контролната точка, а потоа прекинува. Во такви моменти, програмерите можат да ја разгледаат содржината на избраните променливи за набљудување. Потоа, продолжува извршувањето на програмата до следната контролна точка. Сите команди поврзани со следењето на програмата се во менито Debug.

Контролна точка се поставува со позиционирање на покажувачот во избраната линија во програмата и задавање на командата Debug ▸ Toggle Breakpoint (или F5). Притоа, лево од линијата се поставува црвена точка. Со кликање со глувчето на точката, таа се отстранува. Следењето на програмата започнува со задавање на командата Debug ▸ Start. Програмата се извршува сè до достигнувањето на контролната точка кога прекинува и се појавува една жолта стрелка во линијата. Оттаму следењето на програмата продолжува преку останатите команди од менито:

Debug – Next Step (се извршува само една наредба, по што жолтата стрелка преминува во следната линија).

Continue (се извршува програмата до следната контролна точка или до крајот ако нема друга контролна точка).

Step into (се користи само за наредби во кои се повикува функција и потребно е следење на повиканата функција).

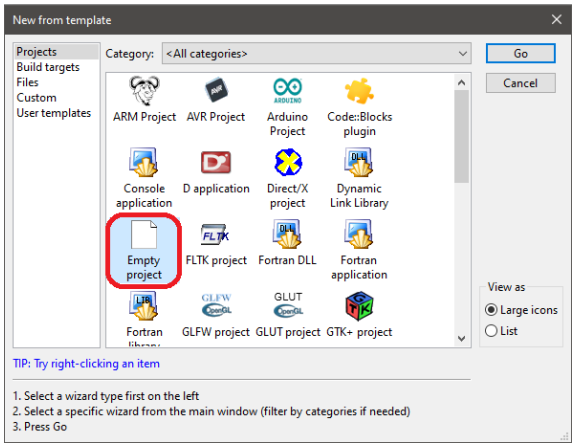
Следењето се прекинува со командата Debug ▸ Stop Debugger.

Во секој момент програмерите можат да користат специјален начин за задавање на (привремена) контролна точка со командата Debug ▸ Run to Cursor. Пред да се зададе командата, покажувачот треба да се постави на почетокот на линијата од која сакаме да продолжи следењето на извршувањето на програмата.

Вежби

Вежба 1.5.4

Стартувајте ја Code::Blocks. Одберете креирање на нов проект со наредбата File ▸ New ▸ Project ▸ Empty Project ▸ Go, ***слика 1.5.18***.

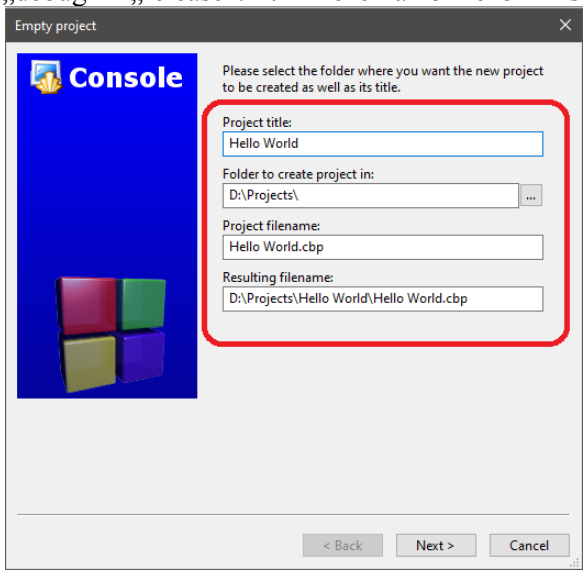


Слика 1.5.18

Во следниот прозорец (***слика 1.5.19***) внесете „Project Title“, „Folder“, „Project filename“ и „Resulting file Name“. На пример, за наслов на проектот внесете PrvProekt, а останатите полиња оставете ги како што се.

По кликнување на копчето Next, во прозорецот што ќе се отвори (***слика 1.5.20***) одберете GNU GCC Compiler и означете ги (со кликнува-

ње во квадратчињата (✓)) следните две опции за да креирате конфигурации „debug“ и „release“. Кликнете на копчето Finish.

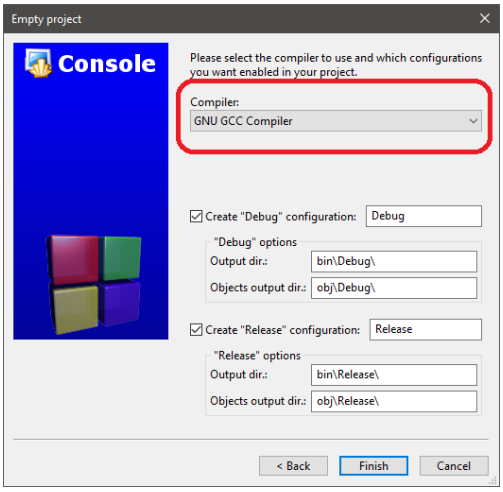


Слика 1.5.19

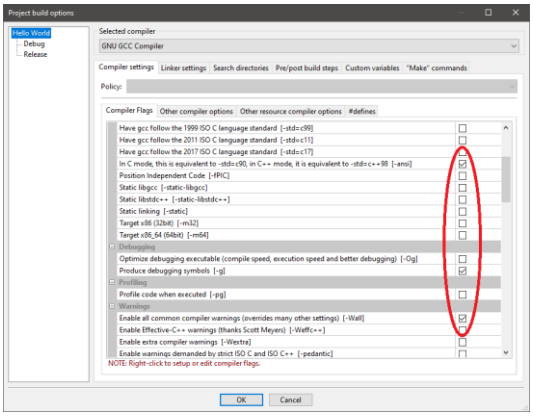
Додадете изворна датотека во проектот: File ▸ New ▸ File ▸ C/C++ Source. Потоа, внесете го името на датотеката со полната патека (на пример, Vezhba1) во соодвената папка каде што е проектот и не заборавајте да го вклучите „Add file to active project“.

За секој проект треба да се постават следните опции: „Project ▸ Build Options.. ▸ Compiler Flags“.

Со ова, креиравте еден проект кој во себе (засега) содржи една празна датотека, која е спремна во неа да ја напишете Вашата прва програма.



Слика 1.5.20



Слика 1.5.21

Вежба 1.5.5

а) Во отворената датотека од **Вежба 1.5.4** внесете ја следнава програма:

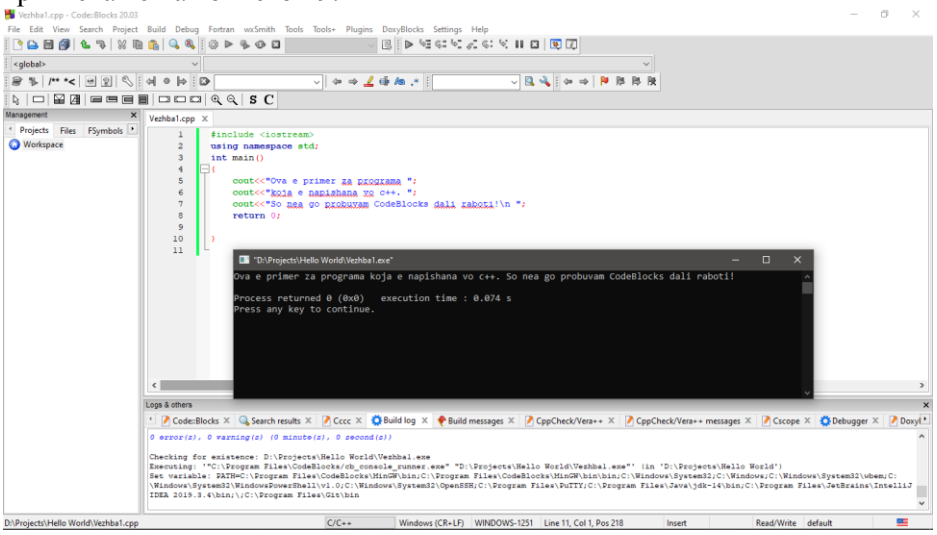
```
#include <iostream>
using namespace std;

int main()
{
	cout << "Ova e primer za programa";
	cout << "koja e napishana vo C++.";
	cout << "So nea go probuvam CodeBlocks dali raboti!\n";
	return 0;
}
```

Слика 1.5.22

б) Зачувајте ја програмата со командата File ▸ Save, на пример, под име Vezhba1.

в) Преведете ја програмата Vezhba1.cpp со командата Ctrl + F9.

г) Ако при внесувањето на програмата Vezhba1.cpp сте пропуштиле нешто во однос на дадениот текст под а), отстранете ги грешките и зачувајте ги промените со притискање на Ctrl + S. Потоа, извршете ја програмата со притискање на копчето F9.



Слика 1.5.2

Вежба 1.5.6

а) Креирајте нов проект и повторно поставете ја претходната програма. Извршете ја.
б) Намерно направете некоја грешка во програмата. На пример, наместо cout, напишете caut во една од наредбите за печатење. Обидете се сега да ја извршите програмата. Што се случува? Поправете ги грешките и повторно извршете ја програмата.
в) Ископирајте ја четвртата линија од програмата и поставете ја како петта линија. Повторно извршете ја програмата. Кој е резултатот од извршувањето?

Вежба 1.5.7

Креирајте нов проект, напишете ја следнава програма (***слика 1.5.24***) и извршете ја:

```
#include <iostream>
using namespace std;

int main()
{
	// Mojata vtora programa vo C++

	cout << "Moeto ime e... " << endl;
	cout << "Moeto prezime e... " << endl;
	cout << "Mojot rodenden e na... " << endl;

	return 0;
}
```

Слика 1.5.24

1.6 Вовед во C++

Кратка историја на C++

Во 1972 година во лабораториите „AT&T Bell Labs“ е дизајниран јазикот C од Dennis Ritchie (Денис Ричи). Во него се вградени неколку особини кои ги немале тогашните програмски јазици, и тоа:

- Дозволува пристап до ресурси на многу ниско ниво.
- Погоден е за системско програмирање[11].
- Може да се извршува на различни компјутерски платформи.
- Работи под различни оперативни системи.

Во 1980 година во C се додадени можности за работа со т.н. **класи** (англ. classes) и таа верзија на C е наречена „C со класи“.

Понатамошниот развој на оваа верзија, односно нејзиното надградување, во 1983 година довело до нов јазик кој е наречен C++ бидејќи претставува надградба на C. C++ претставува јазик за т.н. **објектно ориентирано програмирање** (англ. object-oriented programming).

C++ е дизајниран од Bjarne Stroustrup (Бјарне Строуструп), а целта е:

- да се направи јазик за општа намена подобар од C,
- да поддржува **апстрактни типови на податоци**[12] (англ. abstract data types),
- да поддржува објектно ориентирано програмирање.

Денес, C++ спаѓа меѓу најкористените јазици за изработка на програми од разни видови, кои може да варираат од кориснички апликации за најразлични намени, па сè до системски алатки.

Елементи на јазикот C++

Јазикот C++ има азбука, која е множеството на дозволени симболи, и тоа:

- Малите букви од абецедата: a – z.
- Големите букви од абецедата: A – Z.
- Цифрите: 0 – 9.
- Специјалните знаци: ~ ! @ # $ % ^ & * ( ) – + = { } [ ] : ; '...' "..." < > ? /.

[11] Програмирање на хардверско ниво.

[12] Тоа се структури со кои може да се пресликува реалниот свет во програма. На пример, во една програма може да се дефинира ученик, кола, правоаголник и сл. и со нив да се извршуваат соодветни операции, како: пресметување на успехот на ученик, плоштината на правоаголник и сл. Со апстрактни типови податоци се работи во објектно ориентираното програмирање.

Од азбуката на C++ се формираат зборови кои може да бидат:

- Клучни (анг. keywords) зборови.
- Бројни, симболички и текстуални константи.
- Имиња (идентификатори).
- Оператори.

Посебен вид симболи се одделувачите (сепараторите) со кои се одделуваат зборовите еден од друг. Тоа се:

- Празно место (бланко).
- Нов ред.
- Табулатор.

Како и сите виши програмски јазици, така и C++ има своја **граматика** која ги содржи правилата за градење зборови и наредби. C++ има одредено множество **резервирани зборови**. Некои од нив имаат посебно значење за C++ и тие се нарекуваат **клучни зборови**. Програмерите можат да формираат зборови по одредени правила наречени **идентификатори**. Резервираните зборови не може да бидат идентификатори.

Со точно дефинирани комбинации на зборови и симболи, се конструираат наредбите на јазикот. Притоа, се користат строги правила наречени **синтаксички правила** или само **синтакса** на јазикот. При преведување на програмата, со синтаксичките правила се проверува исправноста на секоја наредба.

Секоја наредба во програмата мора да има точно дефинирано значење и да предизвика точно дефинирани дејства. Значењето (смислата) на наредбите е наречено **семантика** на јазикот.

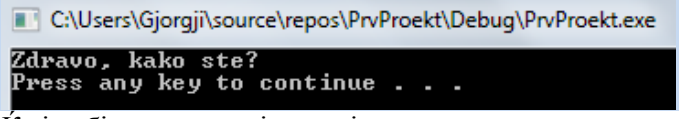
На пример, синтаксичкото правило во C++ за главната (main()) функција е:

```
int main() {
        naredba;
        ...
        naredba;
}
```

Програма во C++

На ***слика 1.6.1*** е дадена програмата Zdravo.cpp која ја креиравме во точката **1.5 Интегрирана развојна околина, *слика 1.5.10*.**

Програмата печати на екранот:



Ќе ја објасниме секоја линија од програмата.

Линија 1: Директива

Директива со која се вклучува библиотеката **iostream**, која содржи функции за влезни и за излезни операции во програмата.

Линија 2: Директива

Директива за формирање на именски простор (опсег) на имиња.

Линија , и 9: Празна линија

Zdravo.cpp

PrvProekt

```
1   #include <iostream>
2   using namespace std;
3
4   int main() {
5
6       // Mojata prva programa Zdravo.cpp
7
8       cout << "Zdravo, kako ste?" << endl;
9
10      system("pause");
11      return 0;
12  }
```

Слка 1.6.1

Празните линии служат за поголема прегледност на програмата, а преведувачот ги игнорира.

Линија 4: Главна функција

Програмите во C++ се состојат од **функции**[13]. Секоја програма во C++ мора да има само една главна функција наречена main() и мора да биде декларирана, како што е во примерот. Зборот int значи дека функцијата main() враќа резултат цел број. Подоцна ќе видиме дека може да се декларираат и функции кои не враќаат резултат.

Линија 4: Влезен дел на големите загради – почеток на функција

Влезниот дел на големите загради по функцијата main(), означува почеток на **телото на функцијата.** Телото на секоја функција завршува со излезниот дел од големите загради, во нашиот пример, тоа е излезниот дел од заградите во *Линија 12*. Линиите во телото на функцијата (*Линија 6* до *Линија 11*) се вовлекуваат надесно и тоа го нарекуваме **назабување** (англ. indentation).

Влезниот дел од големите загради со кој почнува функцијата може да се стави и во следната *Линија 5*.

Линија 6: Коментар

```
// Mojata prva programa Zdravo.cpp
```

Добра програмерска практика е програмите да се коментираат. Затоа, препорачливо е на почетокот на секоја програма да се става **коментар** за тоа што работи програмата. Коментарите се корисни и на почетокот на секоја функција.

Во дадената програма, *Линија 6* не му кажува ништо на преведувачот и тој не ја преведува. Затоа, таа линија се нарекува коментар. Коментар е текст кој е корисен за оној кој ја чита програмата и е наменет за поголема прегледност и

[13] **Функција** во програмските јазици е програма која по извршувањето дава резултати. На пример, функција за множење на два броја, за делење на два броја, за наоѓање квадратен корен од број итн.

разбирање. Исто така, коментарите се корисни и за други програмери кои треба да ја доправат, изменат или подобрат програмата. Често се случува програмерот да внесе коментар во програмата со цел да може полесно да ја разбере кога подоцна самиот тој ќе ја разгледува.

Забележувате дека секој коментар започнува со две коси црти //. Значи, сè што ќе се напише по // до крајот на линијата претставува коментар.

При преведување на програмата, преведувачот ги игнорира (не ги преведува) коментарите.

Линија 8: Наредба за печатење.

```
cout << "Zdravo, kako ste? " << endl;
```

cout	– наредба за печатење
<<	– оператор за печатење
"Zdravo, kako ste? "	– текст кој се печати
endl	– наредба за крај на тековната линија
;	– знак за крај на наредбата, односно разделувач на наредбите.

Гледаме дека тоа што сакаме да се отпечати го ставаме во наводници. Најчесто, во наводници се става некој текст, кој претставува **низа од знаци** и се нарекува **стринг**[14] (англ. string).

Линија 10: Наредба за пауза

Наредба за пауза пред да се затвори прозорецот со резултати. Оваа наредба не е задолжителна, но е корисна ако сакаме да ги видиме дотогашните резултати.

Линија 11: Наредба за враќање на вредност

Ако вредноста што се враќа е 0 (како во нашиот пример), значи дека програмата коректно се извршила. Ако се врати која било друга вредност (1, 2...), значи дека нешто било погрешно при извршување на програмата. Ние ќе ја користиме наредбата **return 0**.

Линија 12: Излезен дел од големите загради за крај на функцијата main().

Линиите 1 и 2 мора да се вклучат во секоја програма C++.

Функцијата main() има стандардна форма:

```
int main() {
	Naredba;
	...
	Naredba;
	return 0;
}
```

[14] Подоцна ќе објасниме што се стрингови и како се користат.

Програмите во C++ може да се пишуваат со кој било уредувач на текст. Потоа, се преведуваат и извршуваат, во зависност од интегрираната развојна околина.

Наредба за печатење

Рековме дека наредбата за печатење на стрингови има форма:

```
cout << "Zdravo, kako ste?" << endl;
```

Значи, ако сакаме да се напише уште нешто на екранот, едноставно под наредбата од *Линијата 8* во програмата од ***слика 1.6.1*** ќе додадеме уште наредби, како што е направено на ***слика 1.6.2***, иста со ***слика 1.5.15***.

```
1   #include <iostream>
2   using namespace std;
3
4   int main() {
5
6       // Mojata prva programa Zdravo.cpp
7
8       cout << "Zdravo, kako ste?" << endl;
9       cout << "Ova e primer za programa" << endl;
10      cout << "koja e napisana vo C++." << endl;
11      cout << "So nea proveruvame dali MVS 2019 dobro raboti." << endl;
12
13      system( "pause" );
14      return 0;
15  }
```

Слика 1.6.2

По извршувањето на програмата, ќе го добиеме следниот излез:

```
Zdravo, kako ste?
Ova e primer za programa
koja e napisana vo C++.
So nea proveruvame dali MVS 2019 dobro raboti.
Press any key to continue . . .
```

Гледаме дека секој стринг во наредбите се печати во нов ред. Тоа се постигнува на тој начин што на крајот на наредбата cout се става т.н. манипулатор endl[15].

Тоа значи дека следната наредба по наредбата за печатење која на крајот го има манипулаторот endl, ќе печати во нова линија (нов ред).

За да печатиме во продолжение во истата линија, претходната наредба не треба да има на крајот endl.

На пример, ако наредбата во Линијата 9 од ***слика 1.6.2*** на крајот нема endl, тогаш ќе се отпечати:

[15] Во C++ постојат повеќе манипулатори (помошни функции) за контрола на внесување и на печатење податоци.

```
Zdravo, kako ste?
Ova e primer za programakoja e napisana vo C++.
So nea proveruvame dali MVS 2019 dobro raboti.
Press any key to continue . . .
```

Како што забележуваме, последниот збор од втората наредба и првиот збор од третата наредба се споени. За да се разделат треба да се остави празно место на крајот на стрингот од втората наредба или на почетокот на стрингот од третата наредба.

Наредбите во Линиите 8, 9, 10 и 11 понекогаш се нарекуваат и **искази**. Можете да забележите дека секој исказ завршува со точка и запирка (;).

Гледаме дека наредбата за печатење започнува со зборот cout (се чита „си аут“), по кој следат операторот за печатење <<, наречен **оператор за вметнување** (англ. insertion operator) и изразот кој се печати. Во нашиот пример, изразот претставува низа од знаци (стринг), кој се печати на стандардниот излезен уред на компјутерот, обично на екранот.

Општата форма на наредбата за печатење е:

```
cout << izraz ;
```

Операторот << е бинарен оператор чиј лев операнд е зборот cout, а десен операнд е изразот кој треба да се отпечати.

Операторот << може да се користи повеќепати во иста наредба. На пример:

```
cout << "Zdravo, kako ste? " << "Ova e primer za programa. " << endl;
```

Притоа, двата стринга ќе се отпечатат во иста линија:

```
Zdravo, kako ste? Ova e primer za programa
```

Во една линија може да се напишат повеќе наредби за печатење, при што ќе бидат разделени со знакот точка и запирка. На пример, ако наредбите во Линиите 9 и 10 од програмата на ***слика 1.6.2*** ги напишеме во една линија:

```
cout << "Ova e primer za programa. "; cout << "koja e napisana vo C++."
<< endl;
```

излезот ќе биде сосема ист.

Со една наредба за печатење, можеме да отпечатиме повеќе стрингови, како што е покажано на ***слика 1.6.*** . Притоа, ако наредбата е долга, можеме да ја префрлиме во следниот ред со притискање на копчето Enter пред или по операторот <<.

Резултатот од извршување на програмата ќе биде:

```
Avtor na knigata Kales Angja e Stale Popov
ili, so 2 naredbi
Avtor na knigata Kales Angja e Stale Popov
ili, so povekje naredbi
Avtor na knigata Kales Angja e Stale Popov
Press any key to continue . . .
```

```
1     #include <iostream>
2     using namespace std;
3
4     int main() {    // Pecatenje stringovi
5
6         cout << "Avtor na knigata" << " " << "Kales Angja"
7             << " e " << "Stale Popov" << endl;
8         cout << "ili, so 2 naredbi" << endl;
9         cout << "Avtor na knigata" << " " << "Kales Angja";
10        cout << " e " << "Stale Popov" << endl;
11        cout << "ili, so povekje naredbi" << endl;
12        cout << "Avtor na knigata";
13        cout << " ";
14        cout << "Kales Angja";
15        cout << " e ";
16        cout << "Stale Popov" << endl;
17
18        system( "pause" );
19        return 0;
20    }
```

Слика 1.6.

Бидејќи насловите на книгите се пишуваат во наводници, за да се отпечатат наводниците „...“, пред нив се става спротивна коса црта, \ т. е. \"...". Стрингот "Kales Angja" треба да се напише "\"Kales Angja\"".

Исто така, за печатење на празна линија се задава командата:

```
cout << endl;
```

Со овие две измени во програмата, излезот ќе биде:

```
Avtor na knigata "Kales Angja" e Stale Popov
ili, so 2 naredbi
Avtor na knigata "Kales Angja" e Stale Popov
ili, so povekje naredbi
Avtor na knigata "Kales Angja" e Stale Popov

Press any key to continue . . .
```

Честопати, некој подолг стринг треба да го отпечатиме во повеќе линии. За таа цел, се користи специјалната секвенца \n, која овозможува по неа да се печати во следната линија. На пример, ако сакаме да отпечатиме:

```
Avtor na knigata
"Kales Angja"
e
Stale Popov
```

тоа може да се направи со една наредба:

```
cout << "Avtor na knigata \n\"Kales Angja\" \ne\nStale Popov" << endl;
```

Во наредбата се користат секвенците \" за печатење на наводници и \n за преминување и печатење во нова линија.

Во C++ постојат повеќе т.н. **ескејп секвенци** (англ. escape sequences).

Ќе наведеме неколку:

\n Преминување во нова линија.
\" Печатење наводници "...".
\' Печатење полунаводници '...'
\t Преминување на следната табулаторна позиција.
\\ Печатење спротивна коса црта \.

Наредбите за печатење во програмата од ***слика 1.6.*** се наредени една по друга. Тие точно така и ќе бидат извршени од компјутерот, прво првата, потоа втората, па третата итн., т. е. наредбите ќе бидат извршени по редоследот како што се наредени. Ваквата структура од последователни наредби се нарекува **редоследна структура** или **секвенца** (англ. sequence).

На пример, со следнава програма

```
#include <iostream>
using namespace std;

int main() {
    // Primer na redosledna struktura - sekvenca

    cout << "    *" << endl;
    cout << "   * *" << endl;
    cout << "  * * *" << endl;
    cout << " * * * *" << endl;
    cout << "* * * * *" << endl;
    cout << " * * * *" << endl;
    cout << "  * * *" << endl;
    cout << "   * *" << endl;
    cout << "    *" << endl;

    cout << endl;
    system("Color 17");
    system("pause");
    return 0;
}
```

Слика 1.6.4

со редоследна структура наредби, ќе се отпечати сликава:

Во програмата ја додадовме наредбата system(„Color 17“); со која се поставува сина боја на позадината (бројот 1) и бела боја на текстот (бројот 7) за попријатен изглед на приказот на резултатите. (Читателот може да експеримен-

тира со оваа наредба, при што првиот број е боја на позадината, а вториот број е боја на текстот. Броевите може да бидат која било комбинација на првите 16 хексадецималните броеви: 0, 1, 2, 3, 4, 5, 6, 7, 8, 9, A, B, C, D, E, F).

Во нашите примери постојано ќе ги користиме последните 4 наредби.

Задачи за вежбање

1. Во програмата на ***слика 1.6.4***, во наредбата:
   ```
   system("Color 17");
   ```
   сменете ги броевите на бојата на позадината (сега е 1) и броевите на бојата на текстот (сега е 7), кои може да бидат 0, 1, 2, 3, 4, 5, 6, 7, 8, 9, A, B, C, D, E, F. Изберете ја комбинацијата која најмногу Ви се допаѓа.
2. Напишете програма која на екранот ќе го даде следниов излез:
   ```
       *
     *   *
   *       *
 *           *
* * * * * *
   ```
3. Напишете програма со која на екранот ќе го напишете Вашето име со знакот * или со друг знак.
4. Да се напише програма која ќе ги отпечати Вашето име и презиме, а во вториот ред ќе го отпечати бројот на Вашите години.
5. Поправете ја претходната програма да печати колку месеци сте возрасен/на наместо колку години сте возрасен/на. На пример, ако сте роден/а во мај 2003 година, а сега сме октомври 2020 година, тогаш сте возрасен/на (17 години) x (12 месеци) + (10 (октомври) – 5 (мај)) месеци.
6. Да се напише програма која ќе го отпечати резултатот од множењето на 1 234 со 5 678. Проверете дали сте добиле точен резултат со помош на калкулатор.
7. Да се напише програма која ќе го отпечати резултатот од множењето на 123 456 со 7 891 011. Што добивте како резултат? Дали тоа е точниот резултат?

Величини и податоци

Рековме дека алгоритмите се постапки за обработка на податоци, односно со нив се обработуваат влезните податоци по строго пропишан редослед и се добиваат излезни резултати. Се наметнува прашањето какви податоци може да се обработуваат со алгоритмите. Затоа, да видиме што се податоци.

Величина е мерка за некоја карактеристика на даден предмет или појава. На пример, следните карактеристики имаат своја величина: должина, волумен, маса, боја, возраст, брзина и др. Величините имаат вредности кои се изразуваат во мерки или вредности од некое множество. На пример: cm е мерка за должина,

m^3 е мерка за волумен, kg е мерка за маса, godina е мерка за возраст, m/sek е мерка за брзина итн.

Вредностите со кои се изразуваат величините се нарекуваат **податоци**. Значи, податоците имаат вредност. На пример, должината на чекорот на човек е 70 см, волуменот на коцка со страна 2 см е 8 $см^3$, возраста на професорот по информатика е 47 години итн. Податоците се обработуваат со алгоритми, т. е. програми.

Константи и променливи

Во многу случаи податокот, т. е. вредноста на некоја величина, не се менува. На пример, вредноста на бројот π = 3.14 е фиксна и затоа велиме дека таа е **константа** (англ. constant). Фиксните вредности се нарекуваат **литерали** (англ. literals)[16]. Во секој програмски јазик има вградени литерали кои може да се користат како константи во програмите.

На пример, константата π во програмскиот јазик C++ може да се запише со PI и може да се користи во формули[17], како:

```
obikolka = 2 * r * PI;     plostina = r * r * PI;       // * e znak za mnozenje
```

Но програмерите во програмите можат да зададат (именуваат) свои константи. На пример, ако во програмата се користи името на нашата држава, тогаш тоа ќе се зададе како константа со име drzava и вредност `"Makedonija"`[18]. Или, може да се користи константата meseci, која може да прими вредност 12 или некоја друга вредност. Ваквите константи, кои ги дефинираат и именуваат програмерите и на кои може да им се доделува вредност, се нарекуваат **именувани константи** (англ. named constants).

За разлика од константите, вредноста на податоците за некои величини се менува. На пример, возраста на некој човек, времето во текот на денот, брзината на некој автомобил итн., може да добиваат различни вредности и затоа, велиме дека податоците на тие величини се променливи. Велиме дека годините на брат ми се 10, моите години се 17, годините на кучето се 5 итн. Значи, 10, 17 и 5 се вредности на податокот возраст. Значи, возраст е податок кој може да има променливи вредности. Ваквите податоци, кои може да имаат променливи вредности, се нарекуваат **променливи** (англ. variables).

При обработка на податоци со помош на компјутер, константите и променливите имаат свои имиња. Нивните вредности треба да се внесат и да се па-

[16] Во програмите може да користиме разни литерали, т. е. фиксни вредности на податоците. На пример: Земјиното забрзување, матичниот број на граѓанин, името на некој град итн.

[17] Константите во C++ се пишуваат со големи букви (сите букви се големи).

[18] Ако литералот е текст, тогаш се става во наводници.

метат додека се обработуваат. На пример: brojPi, бројE, vozrast, vreme, brzina, boja итн.

Ако податокот е константа, тогаш во меморискатa локација под зададеното име (на пример, brojPi) вредноста не се менува за време на извршување на програмата.

Ако, пак, податокот е променлива, тогаш вредноста на меморискатa локација во која е запишана променливата под некое име (на пример, vozrast) се менува. На пример, vozrast = 17, vozrast = 20 итн. Велиме дека вредноста на податокот vozrast е променлива и затоа, податокот vozrast се нарекува променлива.

Математичките формули најчесто вклучуваат во себе променливи. Велиме дека во функцијата y = f(x), x и y се променливи бидејќи за различна вредност на x се добива различна вредност на y. Или, во формулата за пресметување на плоштината на правоаголник со страни a и b (P = a * b), страните a и b се променливи. Понекогаш, во формулите се вклучени и константи. На пример, во формулата за плоштината на круг е S = r * r * PI. Во формулата може да се јави и број како константа. На пример, V = 4 * r * r * r * PI / 3.

Со програмските јазици може да се обработуваат различни податоци, чии вредности се изразени со:

- цели броеви (… –2, –1, 0, 1, 2, 3…),
- реални броеви (… –2.34… –1.89… 0… 0.573… 123.45…),
- знаци[19] ('?', '*', '@', '7', '+'…),
- текстови[20] (`"Macedonia"`, `"Denes e 21-vi vek"`, `"Informatika"`…),
- слики, звук итн.

Велиме дека тоа се податоци од различен **тип** (англ. type). Покрај типот, податоците имаат **име** и **вредност**. Затоа, велиме дека податоците ги имаат следниве карактеристики:

- Име.
- Тип.
- Вредност.

Според вредноста што им се доделува на променливите, велиме дека тие се од тој тип. На пример, променливата vozrast (ако се изразува во години) е од **целобројен тип** бидејќи може да добива вредности само цели броеви: vozrast = 18, vozrast = 10, vozrast = 123 итн. Слично, променливата plostina (плоштината на круг) е од **реален тип** бидејќи добива вредности на децимални броеви, plostina = (3^2 · PI = 9 · 3.14 = 28.2735). Променлива од **знаковен тип** (може да добие вредност кој било знак ставен во полунаводници) е `znak = '+'`, променлива од **текстуален тип** (може да добие вредност кој било текст ставен во наводници) е `drzava = "Makedonija e vecna"` итн.

[19] Знаците во програмските јазици се означуваат со полунаводници.
[20] Текстовите во програмските јазици се означуваат со наводници.

Имиња на податоци

Имињата (идентификаторите[21]) на податоците во C++ може да бидат:

- Клучни или резервирани зборови[22].
- Кориснички дефинирани имиња.

Резервираните зборови имаат специфично значење и примена. Такви се: const, double, float, int, struct, unsigned, break, continue, else, for и други.

Кориснички дефинираните имиња се формираат од корисниците (програмерите) со почитување на следниве правила:

- Името започнува со буква (A – Z, a – z) или подвлечена црта (_).
- Должината на името е неограничена.
- Малите и големите букви се разликуваат. На пример, jas и Jas се различни имиња.
- Името може да содржи и мали и големи букви од абецедата, цифри 0 – 9 и подвлечена црта.
- Името не смее да содржи празно место.
- Името не смее да содржи специјален знак, како: !, @, #, $, ^ итн.
- Името не смее да биде клучен или резервиран збор.

На пример, правилни имиња на податоци се:

```
a, A1, iTi, i2j3, _ime, radius, R, broj_120, _char, TiUci.
```

Неправилни имиња на податоци се:

```
5_ka        – Името не може да почне со цифра.
do$lar      – Името не може да содржи специјален знак.
i 2         – Не е дозволено празно место во името.
char        – Не може резервиран збор да биде идентификатор.
```

За имиња на податоци не се препорачуваат резервираните зборови или зборови слични на резервираните или зборови кои содржат резервирани зборови.

На пример:

```
CHAR        – Сличен на резервираниот збор char.
_float      – Содржи резервиран збор float.
__          – Две подвлечени црти.
```

[21] Идентификатор (англ. identifier) доаѓа од зборот идентификација, што значи утврдување на идентитетот. Имињата на податоците служат за утврдување кој е податокот и затоа, се нарекуваат идентификатори.

[22] Клучните зборови имаат специјално значење во јазикот. Резервираните зборови се резервирани за јазикот и не може да се користат како идентификатори.

Типови на податоци

Тип на податок претставува конечно множество податоци над кое е дефинирано конечно множество операции[23]. Вредностите на секој тип на податоци се запишуваат во толку мемориски локации колку што е одредено со јазикот за тој тип. Оттука, често се вели дека типот на податокот покажува како да се интерпретира содржината на една или повеќе последователни мемориски локации во кои е сместена вредноста на некој податок.

Во C++ постојат три вида типови на податоци:

- Прости (неструктурирани).
- Сложени (структурирани).
- Адресни.

Типови на податоци

Прости (неструктурирани) типови		
	Интегрални[24]	
	целоброен	short, int, long, long long
	знаковен	char
	логички	bool
	реален	float, double, long double
	наброив	enum
Сложени (структурирани) типови		
	низа	array
	структура	struct
	унија	union
	класа	class
Адресни		
	покажувач	pointer
	референца	reference

Простите или **неструктурираните типови на податоци** се оние кои не може да се делат на делови. На пример, 123, –456, 1234567890, 'W', 123.45 итн. се прости (неструктурирани, неделиви) податоци. Додека, "Ti" е структуриран податок од тип текстуална низа[25], кој се состои од два знаковни (прости, неструктурирани) податоци 'T' и 'i'. Затоа, велиме дека **сложените (структурираните) типови на податоци** се состојат од повеќе прости типови.

[23] Ќе објасниме во продолжение.
[24] Интегрални типови се оние кои претставуваат една целина, т. е. не се деливи.
[25] Ќе објасниме подоцна.

Адресните типови на податоци се оние чија вредност е адреса на некоја мемориска локација[26].

Целобројниот тип на податоци и знаковниот тип може да бидат и неозначени (англ. unsigned):

```
unsigned[27] short
unsigned int
unsigned long
unsigned long long
unsigned char
```

Зборовите short, int, long, char, bool, float, double, enum, signed и unsigned се таканаречени **спецификатори на типот** (англ. type specifiers).

Целобројниот, реалниот, знаковниот и логичкиот тип се нарекуваат и **примитивни типови на податоци** (англ. primitive data types) или основни типови на податоци (англ. fundamental data types). Тие се **вградени типови на податоци** (англ. built-in data types).

Во продолжение, накратко ќе ги објасниме целобројните, реалните, знаковните и текстуалните типови на податоци.

Целобројниот тип на податоци се вредности на позитивните и на негативните цели броеви {... –2, –1, 0, 1, 2... } од точно одредено подмножество за секој поттип (short, int, long, long long). (Види ***табела 1. .1, 1. .2, 1. .***).

Реалниот тип на податоци се вредности на позитивните и на негативните реални, т. е. децимални броеви {... –2.0..., –2.001..., –1.0..., 0.0..., 1.0..., 2.0... } од точно одредено подмножество за секој поттип (float, double, long double).

Знаковниот тип на податоци се вредности од множеството знаци, и тоа: букви, цифри и специјални знаци. При нивното користење како вредност, се ставаат во полунаводници за да се разликуваат од други исти знаци. На пример, целиот број 5 се разликува од знакот '5'. Карактеристичен знаковен податок е едно празно место ' '.

Множеството знаковни податоци е точно подредено во т.н. табела ASCII (American Standard Code for Information Interchange), во која секој знак има свој претходник (освен првиот) и свој следбеник (освен последниот). (Види **Додаток Б: Знаци ASCII**).

Со спојување на знаковни податоци, се формираат низи (секвенци) од знаци, кои се третираат како посебен тип на податоци наречени стрингови.

За да се разликуваат од константите и од променливите во програмата, стринговите се запишуваат во наводници.

[26] Ќе објасниме подоцна.

[27] Unsigned значи дека податоците се само позитивни броеви и бројот 0, т. е. не може да имаат негативни вредности.

На пример:

```
"C++ Osnovi na programiranje"
"Makedonija"
"2020"
"Ponedelnik, 21.06.2045 godina"
" "       // Ова е стринг со едно празно место – бленк (англ. blank) стринг.
""        // Ova e prazen стринг (англ. null string) и се разликува
          // од blank стрингот.
"Stringot mora da se napise vo edna linija i da zavrsi so navodnici, ne smee da
 premine vo nov red, zatoa ova ne e string"
```

Забележете дека последниот од примерите не е стринг. Ако стрингот мора да биде подолг од една линија, тој се дели на повеќе стрингови, кои лесно се спојуваат со знакот +.

На пример, со изразот:

```
"Makedonija" + " " + "2020"
```

ќе се добие стрингот "Makedonija 2021".

Оваа операција со стрингови се нарекува **спојување** (англ. concatenation).

Стринговите претставуваат сложени (структурирани) типови на податоци. Нивниот тип во C++ се означува со **string**.

Стринговите не се дефиниран тип на податоци во јазикот C++, но се вклучени во **стандардната библиотека на C++** (англ. C++ Standard library).

Декларирање на константни и на променливи податоци

За да може да се обработуваат податоци со некоја програма, тие треба да се наоѓаат во работната меморија на компјутерот. Програмата мора да знае на која адреса во меморијата се наоѓа секој податок и колку мемориски локации зафаќа. Рековме дека бројот на меморискиите локации што го зафаќа некој податок зависи од неговиот тип, (види ***табела 1. .1, 1. .2, 1. .***). Затоа, за да знае преведувачот кој податок од кој тип е, покрај податокот, мора да се запише и неговиот тип. Таа постапка се нарекува **декларација на податоци** (англ. data declaration). При декларација, за секој податок се задаваат неговото име и типот. Ако при декларацијата се зададе и почетната вредност на податокот, тоа се нарекува **декларација и иницијализација** (англ. declaration and initialization).

Пред да се користи некоја константа или променлива во програмата (во разни операции, како: пресметување, копирање, печатење итн.), таа мора да биде декларирана. При преведувањето, преведувачот резервира соодветна меморија за секоја декларирана константа или променлива (според типот) или резервира и поставува вредност во локацијата ако се врши декларација и иницијализација. Притоа, програмерите не знаат на која локација во меморијата се наоѓаат податоците, туку преведувачот ја поврзува локацијата со името на константата или на променливата (идентификаторот) на податокот.

Рековме дека податоците може да бидат константи и променливи.

Константи се оние идентификатори чија вредност (содржина на локацијата во меморијата) не се менува за време на извршување на програмата. (Ако се обидеме да ја промениме вредноста на некоја константа, ќе се појави грешка).

Константите се декларираат со зборот const по кој се наведуваат типот и името, а потоа мора да се додели вредност со операторот за доделување =:

const *тип име* = *вредност*; [28]

За да се разликуваат константите од променливите, се користи практиката тие да се пишуваат со големи букви, а зборовите да се разделуваат со долна црта.

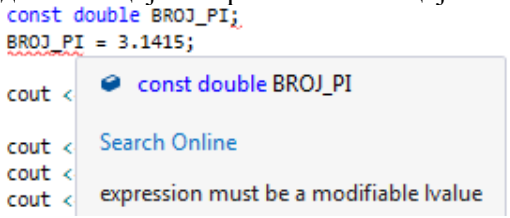
Примери:

```
const int M = 12;                          // Meseci vo godinata
const float ZEMJINO_ZABRZUVANJE = 9.81;
const double BROJ_PI = 3.1415;
const char DA = 'D';
const string DATUM = "Ponedelnik, 21.06.2036";
```

Константата M е од типот int и во нејзината локација може да се стави само целобројна вредност. Константата DA е од типот char, што значи дека вредноста што може да ѝ се додели мора да биде знак. Со други зборови, типот на константа одредува каква вредност може да добие константата.

Исто така, многу е корисно по константата да се напише коментар, ако е тоа потребно, за да се знае што е таа константа.

Да забележиме уште еднаш дека при декларација на константа, таа мора да се иницијализира. Ако се иницијализира по декларацијата, ќе се јави грешка.



Променливи се оние идентификатори чија вредност (содржина на локацијата во меморијата со која е поврзано нивното име), може да се менува.

Променливите се декларираат со:

тип име;

или:

тип име = *вредност*;

или:

тип име = *израз*;

[28] Во општите дефиниции ќе користиме коси букви. Изразите од еден збор ќе ги запишуваме со споени зборови, а изразите од повеќе зборови (како листи) ќе ги запишуваме со зборови споени со долна црта.

Последниов начин претставува истовремена декларација и иницијализација (доделување почетна вредност) на променлива. Вредноста на иницијализирана променлива може (подоцна) да се промени каде било во програмата, со наредба за доделување вредност на променливата.

За да може да се користи во програмата, променливата мора да е дефинирана. Таа се **дефинира** со доделување вредност при декларирање (со иницијализација) или со наредба за доделување.

Стандарден оператор за доделување вредност на променлива е = . Десната страна при доделување може да биде вредност или израз.

Наредбата за доделување вредност на променлива е:

име = *вредност*;

или:

име = *израз*;

Декларација, декларација и иницијализација и дефиниција на променливи може да се направат каде било во програмата.

Примери:

```
short i;                    // Deklaracija na celobrojnata promenliva i.
int m = 10, n = 17;         // Deklaracija i inicijalizacija na
                            // celobrojnite promenlivi m i n.
float tezina = 56.75f;      // Deklaracija i inicijalizacija na realnata
                            // promenliva tezina.
double zbir;                // Deklaracija na realnata promenliva zbir.
char znak = '+';            // Deklaracija i inicijalizacija na
                            // znakovnata promenliva znak.
string drzava;              // Deklaracija na stringot drzava.
i = 5;                      // Dodeluvanje vrednost na promenlivata i.
zbir = 123.45;              // Dodeluvanje vrednost na promenlivata zbir.
drzava = "MAKEDONIJA";      // Dodeluvanje vrednost na promenlivata
                            // drzava.
double prvBroj = (1 + 2 * 7) / (7 - 5);   // Deklaracija i inicijalizacija
```

Со првата наредба, преведувачот резервира меморија од 2 бајта за променливата i (бидејќи е од типот short), т. е. таа локација ја поврзува со името i. Ако се обидеме да ја користиме променливата i која е само декларирана, но не и иницијализирана, ќе се јави следнава грешка.

```
C4700    uninitialized local variable 'i' used
```

Со наредбата:

```
i = 5;
```

на променливата i ѝ се доделува вредност 5, т. е. оваа вредност се става на доделената локација за i во меморијата. По оваа наредба, променливата i може да се користи каде било во програмата.

Во втората линија од **Примери** се декларирани две променливи. Тоа е можно само ако променливите се од ист тип.

Во последниот пример, на десната страна од наредбата за доделување се наоѓа аритметички израз, кој прво се пресметува и потоа неговата вредност се доделува на променливата prvBroj.

C++ го дозволува и следниов начин на доделување вредности:

```
x = (y = 10) * (z = 5);      // Na x mu se dodeluva vrednost 50.
x = y = z = 20;              // na x, y i z im se dodeluva vrednost 20.
```

Нема правило за пишување имиња на променливите во C++, освен ограничувањата наведени во насловот **Имиња на податоци** од потточката **1.6 Вовед во C++**. Секој програмер си формира свој стил, но најчесто се користи стилот првата буква да е мала, а секој збор во името да започнува со голема буква. На пример: brojPI, edinecnaCena, imeIPrezime, plostinaNaKvadrat итн. Овој стил се користи и во програмскиот јазик C++, а ќе го користиме и ние.

Пример 1.6.1

Во примерот (***слика 1.6.5***) е декларирана константата BROJ_PI и ѝ е доделена вредноста 3.1415. Потоа, е декларирана променливата short edinecnaCena, на која ѝ е доделена вредност со наредба за доделување, а не при декларирањето.

```
1   #include <iostream>
2   using namespace std;
3
4   int main() {
5       // Deklaracija, inicijalizacija i pecatenje konstanti i promenlivi
6
7       const double BROJ_PI = 3.1415;  // Deklaracija i inicijalizacija na
8                                       //konstanta
9       short edinecnaCena;             // Deklaracija na celobrojna promenliva
10      int parcinjaSladoled = 100;     // Deklaracija i inicijalizacija
11      edinecnaCena = 50;              // Dodeluvanje vrednost na promenliva
12      cout << "Konstantata PI=" << BROJ_PI << endl;
13      cout << "Vrednosta na " << parcinjaSladoled << " sladoledi po "
14          << edinecnaCena;
15      cout << " denari iznesuva " << parcinjaSladoled * edinecnaCena
16          << " denari." << endl;
17
18      cout << endl;
19      system( "Color 17" );
20      system( "pause" );
21      return 0;
22  }
```

Слика 1.6.5

Резултатот од извршувањето на програмата е:

```
Konstantata PI=3.1415
Vrednosta na 100 sladoledi po 50 denari iznesuva 5000 denari.

Press any key to continue . . .
```

Задачи за вежбање

1. Напишете 3 програми во кои ќе ги вклучите наредбите за декларирање и иницијализација. Потоа, програмите дополнете ги со наредби за печатење на екранот со кои ќе ги отпечатите вредностите на сите декларирани променливи. Што се случува кога печатите вредност на неиницијализирана или недефинирана променлива?
2. Да се напише програма во која ќе декларирате и иницијализирате две променливи со наредбата int a = 12, b = 7;. Потоа, отпечатете ги нивниот збир, разлика и производ во три последователни линии на екранот.
3. Нека е декларирано/дефинирано:

```
int x;
float y;
char z;
x = 2; y = 5.8; z = 'P';
```

Да се напише програма со која ќе се отпечатат вредностите на променливите x, y и z во една линија, така што да бидат одвоени со по едно празно место, а потоа да се премине во нова линија.

4. Нека е декларирано/дефинирано:

```
int den = 21;
string mesec = "Juni";
int godina = 2020;
string denes = "Ponedelnik";
```

Да се напише програма во која ќе ги користите декларираните и иницијализирани променливи и ќе се отпечати:

```
Denes e Ponedelnik, 21 Juni 2021 godina.
```

Прашања за проверка на знаењето

1. Кој го дизајнирал јзикот C++?
2. Од што се состои множеството знаци во C++?
3. Набројте некои од специјалните знаци што се користат во C++.
4. Што се идентификатори?
5. Што е синтакса, а што семантика на јазик?
6. Како се делат идентификаторите?
7. Напишете ја формата на функцијата main().
8. Со кој знак се разделуваат наредбите во C++?
9. Зошто се врши назабување на програмата?
10. Како се означуваат коментарите во програма?
11. Која е наредбата за печатење во C++?
12. Кој е операторот за печатење?
13. Со која ескејп секвенца се преминува за печатење во нов ред?
14. Што е редоследна структура од наредби?
15. Што е величина, а што е податок?

16. Што е константа, а што променлива?
17. Кои вредности ги нарекуваме литерали?
18. Колку може да биде долго името на променлива во C++?
19. Кои се карактеристиките на податок?
20. Какви може да бидат имињата на податоците во C++?
21. Наведете ги правилата за дефинирање кориснички имиња (идентификатори).
22. Кои од следниве имиња на податоци се правилно напишани, а кои не се и зошто?

    а) _54abv б)jac_5T в) prinf г) aj De

    д) moeIme ѓ) for е) a.7_b9 ж) b9_c10
23. Објаснете што е тип на податок?
24. Кои типови на податоци ги знаете во C++?
25. Објаснете што се прости, а што сложени типови на податоци.
26. Каква може да биде содржината на адресните типови на податоци?
27. Кои спецификатори на типот ги знаете?
28. Што означува зборот unsigned ако се наведе пред типот на податок?
29. Кои вредности може да ги имаат податоците од целоброен тип, од реален тип, од знаковен тип и од текстуален тип. Со кои спецификатори се означуваат овие типови?
30. За кој идентификатор велиме дека е константа, а за кој дека е променлива?
31. Колку пати и на кои места во програмата може да се користи иста променлива?
32. Што ќе се случи ако во иста програма користите и константа и променлива со исто име?
33. Дали може името на програмата да го користите како променлива?
34. Што е декларација, а што дефиниција на променлива?
35. Што значи иницијализација на променлива?
36. Напишете ја општата форма на наредбата за декларација и иницијализација.
37. Наведете неколку примери за декларација и неколку за декларација и иницијализација на променливи.
38. Со кој оператор се доделува вредност на променлива или на константа?
39. Кога може да се користи некоја променлива која е само декларирана?
40. Дали следнава декларација е исправна: int m = 10, n = m;.

1.7 Типови на податоци

Целоброен тип на податоци int

Целоброен тип на податок може да има вредност на кој било цел број од множеството цели броеви Z = {... –2, –1, 0, 1, 2...}.

Рековме дека има четири типови на целобројни податоци: short, int, long и long long. Зависно од типот, големината на множеството вредности е различна.

Во ***табела 1. .1*** се дадени целобројни типови на податоци, опсегот на нивните вредности (англ. data range) и количината на меморијата[29] што ја зафаќаат.

Тип	Вредности	Меморија
short	–32768... 32767	16 бита
int	–2147483648... 2147483647	32 бита
long	–2147483648... 2147483647	32 бита
long long	–9223372036854775808... 9223372036854775807	64 бита
unsigned short	0... 65535	16 бита
unsigned int	0... 4294967295	32 бита
unsigned long	0... 4294967295	32 бита
unsigned long long	0... 18446744073709551615	64 бита

Табела 1. .1

Опсегот на вредностите на секој тип зависи од компјутерот на кој се извршува програмата. Минималните и максималните вредности се дефинирани како константи:

```
short                SHRT_MIN = -32768
                     SHRT_MAX = 32767
unsigned short       USHRT_MAX = 65535
int                  INT_MIN = -2147483648
                     INT_MAX = 2147483647
unsigned int         UINT_MAX = 4294967295
long                 LONG_MIN = -2147483648
                     LONG_MAX = 2147483647
unsigned long        ULONG_MAX = 4294967295
long long            _I64_MIN = -9223372036854775808
                     _I64_MAX = 9223372036854775807
unsigned long long   _UI64_MAX = 18446744073709551615
```

За операции со целобројни типови на податоци, се користат следниве аритметички оператори:

+	Собирање
–	Одземање и негација[30]

[29] Количината на меморијата што ја зафаќаат типовите податоци може да варира од една компјутерска платформа до друга.

*	Множење
/	Целобројно делење – цел дел од количникот на два цели броја
%	Делење по модул – остаток од делењето на два цели броја

Резултат од делењето на два цели броја со операторот / е целиот дел од делењето. На пример:

```
10 / 4 = 2,  -15 / 6 = -2,    20 / (-6) = -3,    (-14) / (-3) = 4
5 / 0  ќе јави грешка при преведување.
```

Резултат од примена на операторот % над два цели броја е остатокот од делењето. На пример[31]:

```
10 % 4 = 2,  -15 % 6 = -3,   20 % (-6) = 2,    (-14) % (-3) = -2
5 % 0  ќе јави грешка при преведување.
```

Приоритетот на операторите за цели броеви е:

1	*	/	%	ист приоритет
2	+	–		ист приоритет

При пресметување на аритметички изрази, операциите се извршуваат одлево надесно. Прво се извршуваат операциите со приоритет 1 (која било од трите операции), а потоа операциите со приоритет 2 (која било од двете операции).

На пример:

```
1 + 2 * 3 – 4 % 5 * 6 / 7 = 1 + 6 – 3 = 4
```

Со употреба на загради, се менува редоследот на извршување на операциите одреден со приоритетите на операторите.

На пример:

```
(1 + 2 * (3 – 4)) % (5 * 6 / 7) = (1 + 2 * (-1)) % 4 = (1 – 2) % 4 =
-1
```

Примери:

```
7 / 3 = 2
205 / 20 = 10
205 % 20 = 5
-25 % 3 = -1
7.0 / 4 = 1.7500
(1 + 2 * (3 + 4)) % 5 = 0
(1 + 2 * 3 + 4) % 5 = 1
1 + 2 * 3 + 4 % 5 = 7
-15 % (-4) = -3
15 % (-4) = 3
2147483647 + 1 = -2147483648      // Преполнување[32]
```

[30] Знакот минус (–) пред еден операнд се нарекува унарен оператор, т. е. унарен минус.

[31] Ако еден од операторите е негативен, знакот за остатокот зависи од преведувачот.

[32] За преполнување ќе објасниме подоцна.

Стандарден оператор за доделување вредност на променливи во наредбите за доделување е = . При доделување на неозначена константа, треба да се стави наставката (суфиксот) u или U. Во спротивно, ќе се третира како int.

И за следните типови на константи се става наставка:

тип	наставка
long	l или L
long long	ll или LL
unsigned long	ul или UL
unsigned long long	ull или ULL.

Примери:

```
int prvBroj, vtorBroj;
short cifra = 8;
long arapskiCifri = 1234567890;
prvBroj = 1 + 2 * 3 + 45 % 6;            (= 10)
prvBroj = prvBroj + cifra;               (= 18)
vtorBroj = prvBroj / cifra;              (= 2)
vtorBroj = vtorBroj % cifra;             (= 0)
const long OD_SONCE_DO_SATURN = 1424600000L;
long long astronomskaEdinica = 149597870700LL;
unsigned int siteCifri = 1234567890u;
unsigned long maxLong = 4294967295ul;
unsigned long long maxLL = 18446744073709551615ull;
x = (y = 10) * (z = 5);
x = y = z = 20;
```

Во следниот пример ќе претставиме целобројно делење, делење по модул, т. е. остаток од целобројно делење и приоритет на операциите за цели броеви.

Пример 1.7.1

```
1   #include <iostream>
2   using namespace std;
3
4   int main() {    // Operacii so celi broevi
5
6       // Celobrojno delenje - cel del od kolicnik na dva celi broja
7       cout << "10 / 4 = " << 10 / 4 << endl;
8       cout << "-15 / 6 = " << -15 / 6 << endl;
9       cout << "20 / (-6) = " << 20 / ( -6 ) << endl;
10      cout << "-14 / (-3) = " << ( -14 ) / ( -3 ) << endl;
11      // Delenje po modul - ostatok od delenje na dva celi broja
12      cout << "10 % 4 = " << 10 % 4 << endl;
13      cout << "-15 % 6 = " << -15 % 6 << endl;
14      cout << "20 % (-6) = " << 20 % ( -6 ) << endl;
```

Слика 1. .1

```
15      cout << "-14 % (-3) = " << ( -14 ) % ( -3 ) << endl;
16      // Prioritet na operatorite za celi broevi
17      cout << "1 + 2 * 3 - 4 * 5 % 6 / 7 = "
18          << 1 + 2 * 3 - 4 % 5 * 6 / 7 << endl;
19      cout << "(1 + 2 * (3 - 4)) * (5 % 6 / 7) = "
20          << ( 1 + 2 * ( 3 - 4 ) ) % ( 5 * 6 / 7 ) << endl;
21
22      cout << endl;
23      system( "Color 17" );
24      system( "pause" );
25      return 0;
26  }
```

Слика 1. .1 продол ение

Еден излез од извршување на програмата од ***слика 1. .1*** е:

```
10 / 4 = 2
-15 / 6 = -2
20 / (-6) = -3
-14 / (-3) = 4
10 % 4 = 2
-15 % 6 = -3
20 % (-6) = 2
-14 % (-3) = -2
1 + 2 * 3 - 4 * 5 % 6 / 7 = 4
(1 + 2 * (3 - 4)) * (5 % 6 / 7) = -1

Press any key to continue . . .
```

Скратена форма на операторите

Операторите во C++ може да се користат во т.н. **скратена форма** која се изразува со аритметички оператор и операторот за доделување =. Тие се наречени **сложени оператори за доделување** (англ. compound assignment operators).

Општата форма на операторите за доделување е:

аритметичкиоператор =

Скратени форми на операторите за доделување за целобројни податоци се:

+=	**-=**	***=**	**/=**	**%=**

Примери:

`prvBroj += cifra;`	е исто со	`prvbroj = prvBroj + cifra;`
`vtorBroj %= cifra;`	е исто со	`vtorbroj = vtorBroj % cifra;`

Во следниот пример е илустрирано користење на скратена форма на операторите.

Пример 1.7.2

На ***слика 1. .2*** е илустрирано користењето на скратената форма на операторите.

```cpp
#include <iostream>
using namespace std;

int main() {    // Skratena forma na operatori

    int cifra = 8;
    cout << "Cifra = " << cifra << endl;

    int prvBroj, vtorBroj, tretBroj;
    prvBroj = vtorBroj = 100;       // Ova e dozvoleno vo C++
    cout << "Prv broj = " << prvBroj << endl;
    cout << "Vtor broj = " << vtorBroj << endl;

    prvBroj += cifra;               // Skratena forma na operatorot +
    vtorBroj *= cifra;              // Skratena forma na operatorot *
    cout << "prvBroj += cofra =  " << prvBroj << endl;
    cout << "vtorBroj *= cifra =  " << vtorBroj << endl;

    // Dodeluvanje vrednosti na promenlivi vo izraz
    // I ova e dozvoleno vo C++
    tretBroj = ( prvBroj = 123 ) + ( vtorBroj = 321 );
    cout << "Tret broj = " << prvBroj << " + " << vtorBroj << " = "
        << tretBroj << endl;

    cout << endl;
    system( "Color 17" );
    system( "pause" );
    return 0;
}
```

Слика 1. .2

Еден излез од извршување на програмата е:

```
Cifra = 8
Prv broj = 100
Vtor broj = 100
prvBroj += cofra =  108
vtorBroj *= cifra =  800
Tret broj = 123 + 321 = 444

Press any key to continue . . .
```

Задачи за вежбање

1. Напишете програма во која за броевите x = 123 и y = 52 ќе ги пресметате и отпечатите: збирот, разликата, производот, целиот дел од количникот и остатокот од нивниот количник. Притоа, да се користи само една променлива z.
2. Да се напише програма со која ќе ги пресметате и отпечатите квадратот и кубот на целиот број celBrojX = 123, без да користите други променливи.

3. Да се напише програма со која ќе се отпечати средната цифра на трицифреniот број brojX = 274.
4. Возот од Битола за Скопје тргнува во 5 часот и 20 минути. Во Скопје пристигнува во 8 часот и 12 минути. Да се напише програма со која ќе се пресмета колку време возот патува од Битола до Скопје.
5. Да се напише програма со која ќе се пресмета збирот и разликата на аглите $\alpha = 100° 47' 38''$ и $\beta = 35° 53' 49''$.

Реален тип на податоци float, double, long double

Реален тип на податоци се оние кои добиваат вредности од множеството реални броеви. Под реален тип податоци најчесто се мисли на децималните броеви.

Во C++ се вградени три типа на реални податоци, и тоа: **float, double** и **long double**.

Податоците од реален тип може да бидат константни и променливи.

Иницијализацијата се врши како и кај целобројниот тип на податоци.

При декларирање на променливи од типот float, треба да се стави наставката (суфиксот) f или F. Во спротивно, ќе се третира како типот double.

Исто така, на типот long double се става наставка l или L.

Примери:

```
float x;
float vkupnaTezina;
double vkupnaSuma = 12345.67;
long double PI = 3.14159265358979323846;
const float ZEMJINO_ZABRZUVANJE = 9.81f;
float broj_e = 2.7182818284590452354F;
long double radiusNaElektron = 0.00000000000000282L;    //2.82x10^-15
```

Прецизноста на податоците од типот **float, double** и **long double** е различна, и тоа:

- Податоците **float** афаќаат 32 бита и имаат 7 значајни цифри.
- Податоците **double** зафаќаат 64 бита и имаат 15 значајни цифри.
- Податоците **long double** зафаќаат 80 бита и имаат 18 значајни цифри.

Опсегот на вредностите на реалниот тип на податоци е даден во ***табела 1. .2***. Опсегот е пресметан според прецизноста на претставување на броевите со 32, 64 и 80 бита. Постојат стандардни формати за претставување на целите и на децималните броеви[33].

[33] Постои стандард IEEE (Institute of Electrical and Electronics Engineers) со кој се опишани форматите за претставување на целите и децималните броеви во компјутерот, според прецизноста, т. е. според бројот на битови.

Тип	Вредности	Меморија
float	$\pm 3.4028234 \bullet 10^{-38} ... \pm 3.4028234 \bullet 10^{38}$	32 бита
double	$\pm 1.7976931348623157 \bullet 10^{-308} ...$ $\pm 1.7976931348623157 \bullet 10^{308}$	64 бита
long double[34]	$\pm 3.4 \bullet 10^{-4932} ... \pm 1.1 \bullet 10^{4932}$	80 бита

Табела 1. .2

Минималните и максималните вредности на децималните броеви во C++ зависат од преведувачот и се зададени како константи[35]. На пример, на мојот компјутер тие се:

```
float                FLT_MIN = 1.17549e-38
                     FLT_MAX = 3.40282e+38
double               DBL_MIN = 2.22507e-308
                     DBL_MAX = 1.79769e+308
long double          LDBL_MIN = 2.22507e-308
                     LDBL_MAX = 1.79769e+308
```

Аритметички оператори за податоците од реален тип се:

+	собирање
–	одземање или негација
*	множење
/	реално делење (ако барем еден од податоците е реален)

Ако при делењето со 0.0 се излезе надвор од опсегот на типот на броевите, не се јавува грешка, туку за резултат се добива inf или –inf, (бесконечно, англ. infinity).

Приоритетот на операторите за реалните броеви во аритметичките изрази се испитува одлево надесно, и тоа:

1	*	/	ист приоритет (мултипликативни оператори)
2	+	–	ист приоритет (адитивни оператори)

И овие оператори може да се користат во стандардна и во скратена форма:

```
+=      -=      *=      /=
```

[34] Овој тип се користел во поранешните верзии на C++. Во денешните стандарди за C++, овој тип е еквивалентен со double.

[35] Најмалите позитивни вредности за реалните типови податоци се дефинирани како константи. На мојот компјутер тие се:

```
float                FLT_EPSILON = 1.19209e-07
double               DBL_EPSILON = 2.22045e-16
long double          LDBL_EPSILON = 2.22045e-16
```

Пример 1.7.3

Со овој пример (***слика 1. .***) се покажува разликата во прецизноста на податоците од типот float (претставени со 32 бита) и од типот double (претставени со 64 бита).

```
1     #include <iostream>
2     using namespace std;
3
4     int main() {    // Operacii so realni broevi
5
6         // Za tip float
7         float prvBroj, vtorBroj, rezultat;
8         prvBroj = 1.000000000f;
9         vtorBroj = 0.999898989f;
10        rezultat = prvBroj - vtorBroj;
11        cout << "Razlika na decimalni broevi od tip float: " << endl;
12        cout << prvBroj << " - " << vtorBroj << " = " << rezultat << endl;
13
14        // Za tip double
15        double prviotBroj, vtoriotBroj, rezultatot;
16        prviotBroj = 1.000000000;
17        vtoriotBroj = 0.999898989;
18        rezultatot = prviotBroj - vtoriotBroj;
19        cout << "Razlika na decimalni broevi od tip double: " << endl;
20        cout << prviotBroj << " - " << vtoriotBroj << " = "
21            << rezultatot << endl;
22
23        cout << endl;
24        system( "Color 17" );
25        system( "pause" );
26        return 0;
27    }
```

Слика 1. .

Пример од извршување на програмата е:

```
Razlika na decimalni broevi od tip float:
1 - 0.999899 = 0.00010103
Razlika na decimalni broevi od tip double:
1 - 0.999899 = 0.000101011

Press any key to continue . . .
```

Од излезот се гледа дека иако вредноста на податоците е иста, резултатот е различен.

Формати за претставување на децималните податоци

Децималните податоци во програмските јазици се претставуваат на два начини:

- Во децимален формат (**f**-формат, **F**-формат), начин познат како **неподвижна точка** (англ. fixed point).
- Во експоненцијален формат (**e**-формат, **E**-формат), начин познат како **подвижна точка** (англ. floating point).

Буквата **f** доаѓа од зборот „float“, а буквата **e** доаѓа од зборот „exponent“.

Децималните броеви во f-формат (или F-формат) се запишуваат со децимална точка на фиксно место, која точно ги дели целиот и децималниот дел на бројот. Затоа, тој начин на претставување на децималните броеви се нарекува претставување со неподвижна точка**.** На пример: 1234.56, –0.25, 0.5432, –123.45 итн.

Во e-формат (или E-формат) точката е подвижна, при што се користи експонент. На пример, бројот 123456789.12 може да се запише со поместување на децималната точка каде било лево или десно:

$1234567891.2x10^{-1}$, $12345.678912x10^{4}$, $0.0012345678912x10^{11}$

Истите броеви во e- или E-формат се запишуваат на следниов начин:

1.234568e+08, 1.234568E+08.

Кога децималните броеви се печатат во e-формат (или E-формат), се запишува *само една цифра* лево од децималната точка, додека буквата e или E се пишува пред експонентот. Експонентот се пишува со знак – или +.

Печатените децимални броеви во формат f, F, e или E имаат прецизност од 6 точни цифри, а седмата се заокружува.

На пример, ако се дефинирани (декларирани и иницијализирани) следните променливи во f-формат и во e-формат:

```
float a = 1234.56f, a1 = -25000.f, a2 = 0.5432f;
double c = 76.543e3, c1 = -25.e-1, c2 = 2e3;
```

при нивно печатење во f-формат, се добива:

```
1234.560059      -25000.000000   0.543200
76543.000000     -2.500000       2000.000000
```

а при печатење во e-формат, се добива:

```
1.234560e+03     -2.500000e+04   5.432000e-01
7.654300e+04     -2.500000e+00   2.000000e+03
```

Слично, ако во програмата од ***слика 1. .*** малку се смени вредноста на променливата vtorBroj (место 0.999898989, се зададе 0.999998989), вредноста на променливата rezultat ќе се отпечати во **e**-формат:

```
Razlika na decimalni broevi od tip float:
1 - 0.999999 = 1.01328e-06
Razlika na decimalni broevi od tip double:
1 - 0.999999 = 1.011e-06

Press any key to continue . . .
```

Аритметички изрази и конверзија на типот на податоците

При пресметките, во програмите се јавуваат едноставни и сложени аритметичките изрази. Тие се изразуваат со константи, променливи и оператори.

Операторите може да бидат:

Унарни: + и –
Бинарни: + – * / и %

На пример:

–5 + 3, –y, a + b / c % d, 1.2 * x * x – 3.4 * x + 5.67

Ако во изразот има повеќе оператори, тие се извршуваат според приоритетот:

највисок: унарен + унарен –
среден: * / %
најнизок: + –

На пример, за a = 5, b = 4, c = 3, d = 2, изразот a + b / c % d ќе се пресмета:

```
a + b / c % d = 5 + 4 / 3 % 2 = 5 + 1 % 2 = 5 + 1 = 6
```

Приоритетот на операторите може да се смени со загради. На пример, ако претходниот израз го запишеме (a + b) / (c % d), ќе се пресмета:

```
(a + b) / (c % d) = (5 + 4) / (3 % 2) = 9 / 1 = 9
```

Унарните оператори во изразите се извршуваат оддесно налево. На пример, за a = 5 и b = –3:

```
a + ( –b ) = 5 + (– (–3 ) ) = 5 + ( +3 ) = 8
```
(Не може `5 (+ –) (–3) )`

Рековме дека типот на променливата одредува дека соодветната мемориска локација може да содржи само таков тип на податоци.

На пример, ако е декларирано:

```
int celBroj;
float realenBroj;
```

следните наредби за доделување се во ред:

```
celBroj = 5;
realenBroj = 6.7;
```

Но што се случува ако ги смениме вредностите, т. е.:

```
celBroj = 6.7;
realenBroj = 5;
```

Со првата наредба, бидејќи променливата celBroj може да добие вредност само цели броеви, се отсекува децималниот дел од 6.7 и на променливата celBroj ѝ се доделува вредност 6.

Со втората наредба, бидејќи променливата realenBroj може да добие вредност само реални броеви, се конвертира целиот број 5 во реален 5.0 и на променливата realenBroj ѝ се доделува вредност 5.0.

Ваквиот начин на доделување вредност на променлива од несоодветен тип се нарекува **имплицитна** (автоматска) **конверзија** или **типска принуда** (англ. type coercion).

Гледаме дека при доделување реална вредност на целобројна променлива се губат информации за податокот бидејќи се отсекува децималниот дел. За да се избегне губењето информации, се користи **експлицитна типска конверзија** (англ. type conversion), позната и како **кастирање на типот** (англ. type casting).

Кастирањето се врши со наведување на типот пред променливата или изразот кој треба да се конвертира. Така, претходните две наредби со кастирање ќе бидат:

```
celBroj = int(6.7);
realenBroj = float(5);
```

Бидејќи променливата celBroj ќе добие вредност 6 и во случај вредноста да е 6.99 (што е поблизу до 7), за да се изгубат помалку информации за податокот, добра програмерска практика е тој да се заокружи со додавање вредност 0.5.

Наредбата ќе биде:

```
celBroj = int(6.7 + 0.5);
```

Најчесто, аритметичките изрази содржат и целобројни и реални променливи, т. е. содржат мешани типови на податоци.

Во изразите со мешани типови на податоци, како:

```
realenBroj1 = 5 + 7.34;
realenBroj2 = 129.56 – 1.3e+2;
```

се врши имплицитна конверзија на типовите со помала прецизност во типовите со поголема прецизност.

Со следната група наредби:

```
int k = 5;
double p;
p = k + 7.34                (= 5.0 + 7.34 = 12.34)
```

прво се врши имплицитна (автоматска) конверзија на целобројната променлива k (= 5) во променлива од типот double (= 5.0), се собира со 7.34 и потоа збирот се доделува на променливата p (= 12.34).

Во изразот за пресметување на p, може да се користи и експлицитна конверзија:

```
p = float(k) + 7.34;
```

Со следнава наредба:

```
int m = p;                          (12 ← 12.34)
```

се врши имплицитна конверзија на 12.34 во 12, вредност што ѝ се доделува на целобројната променлива m.

Истото може да се направи и со експлицитна конверзија:

```
m = int(p);                         (= 12)
```

при што вредноста на променливата p експлицитно се претвора во типот int.

Следните примери исто така илустрираат конверзија:

```
r = double(3 * 5);          (= 15.0)
cout << int('a') << endl;   (Ќе се отпечати 97)
```

За конверзија на типот, може да се користи и форма која е типична за јазикот C (англ. C-like), т. е. наследена од него. Притоа, се користи операторот **( )**,

кој се нарекува оператор **cast**, наречен и **оператор за кастирање**. Тој се употребува така што прво се задава типот во кој се претвора вредноста на изразот, а по него во заградите следува изразот чиј тип се менува.

На пример:

```
m = (int)(5 + 7.34)          (= 12)
```

Претходните примери може да се запишат и на следниот начин:

```
k = (int)(5 + 7.34);         (= 12)
r = (double)(3 * 5);         (= 15.0)
cout << (int) 'a' << endl;   (Ќе се отпечати 97)[36]
```

Честа грешка се прави при делење на две целобројни променливи. На пример, ако збирот на 8 природни броеви е 100, средната вредност е 100 / 8 = 12.5:

```
        int zbir = 100;
        int broevi = 8;
        float srednaVrednost = zbir / broevi;//Неточно, целобројно делење)
треба   float srednaVrednost = float(zbir) / broevi;
или     float srednaVrednost = zbir / float(broevi);
или     float srednaVrednost = 1.0 * zbir / broevi;     (Без кастирање)
```

Во претходните примери користевме и ваква наредба:

```
prvBroj = prvBroj + cifra;
```

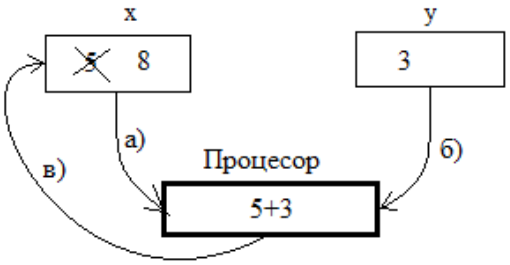
Ако ја запишеме оваа наредба во математичка форма, ќе добиеме невозможна равенка:

```
x = x + y
```

која е точна само ако y = 0.

Но овој израз во програмирањето се извршува коректно. Процесорот прво го пресметува изразот на десната страна, така што ја собира моменталната вредност на променливата x со вредноста на променливата y и добиениот збир го доделува како нова вредност на променливата x.

На пример, ако променливите имаат вредности x = 5 и y = 3, наредбата ќе се изврши според следнава шема.



а) Вредноста на x (= 5) се пренесува во процесорот.
б) На неа се додава вредноста на y (= 3).
в) Добиениот збир се става како нова вредност на x (= 8).

[36] Вредноста ASCII на знакот 'a' е 97.

Ако вредноста на променливата x се зголемува за 1 (велиме, се инкрементира), наредбата ќе биде:

```
x = x + 1;
```

За извршување на оваа наредба, воведен е посебен оператор, наречен **оператор за инкрементирање**[37] (англ. increment operator) ++. Претходната наредба напишана со овој оператор ќе биде:

```
++x;    или    x++;
```

Соодветно на операторот за инкрементирање, воведен е и **оператор за декрементирање** (англ. decrement operator) – –, со кој се намалува вредноста на променливата за 1. На пример, наредбите:

```
– –x;    или    x– –;
```

имаат ист ефект како и наредбата:

```
x = x - 1;
```

Пример 1.7.4

Со програмата од ***слика 1. .4*** е илустрирана имплицитна и експлицитна конверзија.

```
1   #include <iostream>
2   using namespace std;
3
4   int main() {    // Konverzija na tipot na podatoci
5
6       cout << "Implicitna konverzija (samo vo povisok tip)" << endl;
7       int k = 5;
8       double p;
9       p = k + 7.34;
10      cout << "5 + 7.34 = " << p << endl;
11      int m = p;
12      cout << "int m; \nm = 5 + 7.34 = " << m << endl;
13
14      cout << "\nEksplicitna (prinudna) konverzija" << endl;
15      cout << "m = int(5 + 7.34) = " << int( p ) << endl;
16      cout << "m = (int)(5 + 7.34) = " << ( int ) p << endl;
17      cout << "int('a') = " << int( 'a' ) << endl;
18      cout << "(int)'a' = " << ( int ) 'a' << endl;
19      int zbir = 100;
20      int broevi = 8;
21      cout << "100 / 8 = " << zbir / broevi << endl;
22      cout << "float(100) / 8 = " << float( zbir ) / broevi << endl;
23      cout << "100 / float(8) = " << zbir / float( broevi ) << endl;
24      cout << "1.0*100 / 8 = " << 1.0 * zbir / broevi << endl;
```

Слика 1. .4

[37] Овие оператори подетално ќе ги објасниме подоцна.

```
25
26      cout << endl;
27      system( "Color 17" );
28      system( "pause" );
29      return 0;
30  }
```

Слика 1. .4 продол ение

Излезот од извршување на апликацијата е:

```
Implicitna konverzija (samo vo povisok tip)
5 + 7.34 = 12.34
int m;
m = 5 + 7.34 = 12

Eksplicitna (prinudna) konverzija
m = int(5 + 7.34) = 12
m = (int)(5 + 7.34) = 12
int('a') = 97
(int)'a' = 97
100 / 8 = 12
float(100) / 8 = 12.5
100 / float(8) = 12.5
1.0*100 / 8 = 12.5
```

Задачи за вежбање

1. Да се напише програма за пресметување обиколката ($O = 2r\pi$) и плоштината ($P = r^2\pi$) на круг со радиус r = 18.3 см.
2. Да се напише програма за пресметување на плоштината и волуменот на паралелопипед со страни a = 12.5 см, b = 7.2 см и c = 15.8 см.
3. Да се напише програма за решавање на равенката: 5.27x – 23.15 = 0.
4. Да се напише програма за претворање на аголот $\alpha = 36.27°$ во радијани по формулата $\alpha^{rad} = \pi\alpha^{o} / 360$.
5. Да се напише програма за пресметување на брзината на автомобил кој за 5 часа поминува растојание од 465 километри.
6. Да се напише програма за печатење на 3-тата децимала од количникот на реалните броеви a и b, за a = 3 и b = 16.
7. Да се напише програма за пресметување на разликата меѓу вредноста на бројот $\pi = 3.1415$ и збирот $4\left(1-\frac{1}{3}+\frac{1}{5}-\frac{1}{7}+\frac{1}{9}-\frac{1}{11}+\frac{1}{13}-\frac{1}{15}\right)$.

Знаковен тип на податоци **char**

Податоците од **знаковен тип** може да имаат вредност кој било знак или која било целобројна вредност. Тие може да бидат променливи или константни податоци. Се декларираат со зборот **char**. Притоа, знакот се става во полунаводници, за да се разликува од ист знак во текст.

Примери:

```
const char taraba = '#';
char a;
char znak = '+';
char denar = 'd';
char ascii = 120;
```

Сите знаци кои се користат во програмскиот јазик C++, во компјутерот се репрезентираат (запишуваат во меморијата) со цели броеви – кодови, според табелата ASCII (American Standard Code for Information Interchange). (Види **Додаток Б: Знаци ASCII**). На пример, знакот Q има бинарна вредност (код) $01010001_2 = 81_{10}$, додека знакот '=' има код $00111101_2 = 99_{10}$. При печатење на овие кодови, се печатат соодветните знаци.

Податоците од знаковен тип може да се третираат како кој било целобројен тип, водејќи сметка за опсегот на вредностите. Со нив може да се извршуваат аритметички операции, тие може да се читаат, да се печатат и да се споредуваат.

На пример, ако се декларира променливата procent со:

```
char procent;
```

знакот % може да се додели со која било од следниве две наредби:

```
char procent = '%';
procent = 37;
```

Со следнава наредба:

```
procent++;
```

се инкрементира целобројната вредност на знакот % од 37 на 38, а со наредбата:

```
cout << procent << endl;
```

ќе се отпечати знакот &. Со други зборови, стандардните знаци ASCII се подредени и нивната декадна вредност е од 0 (000000) до 127 (1111111).

Проширената табела ASCII содржи 256 знаци со кодови од 0 до 255. Проширувањето е со знаците со кодови од 128 до 255.

Бидејќи знаците во C++ имаат целобројни вредности според табелата ASCII[38], тие може да се споредуваат. На пример, знакот '+' има ASCII-вредност 43_{16} или 67_{10}, а ASCII-вредноста на знакот ':' е 58_{16} или 88_{10}. Затоа, може да се напише '+' < ':' .

Вредностите и опсегот на податоците од знаковен тип се дадени во т***абела 1. .*** . Кодовите од –128 до –1 и од 128 до 255 се кодови на истите знаци.

Тип	Вредности	Меморија
char	–128... 127	8 бита
unsigned char	0... 255	8 бита

Табела 1. .

[38] Стандардната табела ASCII содржи 128 знаци, додека проширената табела ASCII содржи 256 знаци. Но за да се опфатат знаците од повеќе светски јазици, денес се користи т.н. уникод (англ. Unicode), кој претставува интернационален стандард за кодирање.

Минималната и максималната вредност на податоците од типто char се константите:

```
char                    CHAR_MIN = -128
                        CHAR_MAX = 127
unsigned char           UCHAR_MAX = 255
```

Пример 1.7.5

Да се пресмета каматата за вложен капитал од 100 000 денари ако каматната стапка е 7.5 %, по формулата kamata = $\frac{\text{kapital} \cdot \text{kamatnaStapka}}{100}$.

```
1   #include <iostream>
2   using namespace std;
3
4   int main() {    // Znakoven tip podatoci
5
6       char znakProcent = '%';
7       double kapital = 100000.0;
8       float kamatnaStapka = 7.5f;
9       double kamataIznos;
10      kamataIznos = kapital * kamatnaStapka / 100;
11      cout << "Za vlozen kapital od " << kapital << " denari" << endl;
12      cout << "so godisna kamatna stapka od " << kamatnaStapka
13          << znakProcent << endl;
14      cout << "se dobiva kamata od " << kamataIznos << " denari" << endl;
15
16      system( "Color 17" );
17      system( "pause" );
18      return 0;
19  }
```

Слика 1. .5

Еден излез по нејзиното извршување е:

```
Za vlozen kapital od 100000 denari
so godisna kamatna stapka od 7.5%
se dobiva kamata od 7500 denari

Press any key to continue . . .
```

Задачи за вежбање

1. Да се напише програма со која ќе декларирате толку знаковни променливи колку што има букви Вашето име и на секоја ќе ѝ доделите по една буква. Потоа, отпечатете ги така што на излезот ќе го добиете целото име.
2. Да се напише програма за наоѓање на разликата на редните броеви на буквите 'S' и 'D' во табелата ASCII.
3. Цената на некој производ е 123.45 денари, а во текот на една недела поскапел за 15 % и и потоа поевтинил за 8 %. Колку била цената на крајот од неделата? (За знакот % да се користи знаковна променлива).

4. Да се најде и отпечати знакот кој му претходи и знакот кој следи по знакот @ во табелата ASCII.
5. Да се одреди колку букви има меѓу две букви од абецедата.
6. Да се декларираат и иницијализираат две целобројни променливи a и b на вредности 12 345 и 678. Да се декларира и една знаковна променлива operator и да ѝ се доделува по еден од знаците: +, –, *, / и %. По секое доделување на знак, да се отпечатат a и b со знакот на операторот меѓу нив, како и резултатот од операцијата. За резултатот од операцијата, да се декларира соодветна променлива. На пример, за operator = '%' да се отпечати 12 345 % 678 = 141.

Логички тип на податоци bool

Податоците чија вредност може да биде или true или false се нарекуваат податоци од **логички тип** (англ. logical type). Резервираните зборови true и false се специјални константи во C++. При пресметување на логичките изрази, логичките константи true и false имаат вредност 1 и 0 и се печатат како 1 и 0.

Податоците од логички тип се декларираат со зборот **bool**.

Примери:

```
bool dane;
bool semafor = true;
```

Со податоци од логички тип може да се користат и **логичките оператори** (англ. logical operators)[39]:

&&	логичко ***И*** (AND)
||	логичко ***ИЛИ*** (OR)
!	логичко ***НЕ*** (NOT)

Овие оператори се нарекуваат и **условни оператори** (англ. conditional operators) бидејќи најчесто се користат при пресметување на вредноста (true или false) на некој посложен логички израз.

Логичките оператори се, всушност, функции, наречени **Булови функции** (англ. boolean functions)[40], по математичарот Џорџ Бул (George Boole) кој ги дефинирал.

Логичките оператори се дефинирани во следнава табела.

НЕ/NOT/!	***И***/AND/&&	***ИЛИ***/OR/||
!false = true	false && false = false	false || false = false
!true = false	false && true = false	false || true = true
	true && false = false	true || false = true
	true && true = true	true || true = true

Табела 1. .4

[39] Постојат и други логички оператори.
[40] Постојат и други Булови функции.

Примери:

Вредноста на следниве изрази за a = true и b = false ќе биде:

```
а)
(a || b) && (!a || b) = (true || false) && (false || false) = true && false = false

б)
(a && !b) && (!a || b) || (a || b) = (true && !false) && (!true || false) || (true || false) = (true && true) && (false || false) || (true || false) = true && false || true = false || true = true
```

Со сите претходно наведени типови на податоци може да се користат **релациските оператори** (англ. relational operators):

<	помало
<=	помало или еднакво
>	поголемо
>=	поголемо или еднакво
==	еднакво
!=	различно

Да напоменеме дека операторот == не е „две еднакво“, туку бинарен оператор за споредување на два операнди. На пример, ако z е логичка променлива, со наредбата:

```
z = x == y;
```

прво се споредуваат вредностите на x и y и ако се исти, на z ѝ се доделува вредност true, а ако не се исти, на z ѝ се доделува вредност false.

При користење на релациските оператори, треба да се внимава операндите да бидат од ист тип. Во спротивно, може да настане принудна конверзија на еден тип во друг, која може да доведе до погрешен резултат или до појава на порака за грешка.

На пример, во наредбата:

```
bool daNe = 1 == '1';
```

се врши имплицитна конверзија на знакот '1' во цел број и потоа се споредува со константата 1. Резултатот е false. Зошто?

Операндите на релациските оператори може да бидат и од друг прост тип. При користење во изразите, тие принудно се конвертираат во логички тип, и тоа: вредноста 0 во false, а сите ненулти вредности во true.

На пример, следниве наредби:

```
double radius = 1.23;
bool daNe = radius > 0;
```

се извршуваат коректно и логичката променлива daNe добива вредност true.

При споредување на реални броеви, треба да се внимава дека тие се пресметуваат приближно, а не точно.

На пример, со следните наредби:

```
float a = 1.0f / 5;
daNe = a == 0.2;
```

```
cout << "a = 1.0/5 = " << a << " dali e ednakvo 0.2? " << daNe<< endl;
```

резултатот ќе биде false, т. е. 0 за променливата daNe:

```
a = 1.0/5 = 0.2 dali e ednakvo 0.2? 0
```

Но може да се случи операндите и целиот израз да се коректно напишани (преведувачот да не јави грешка), а резултатот да биде погрешен. На пример, вредноста на изразот:

```
bool odlicenUspeh = ocenka == 4 || 5
```

е true, независно каква е вредноста на променливата ocenka бидејќи десниот операнд на операторот || секогаш има вредност true.

Всушност, исправниот израз треба да биде:

```
bool uspeh = ocenka == 4 || ocenka == 5;
```

кој добива вредност false ако ocenka е помала од 4.

Слично, наредбата:

```
bool podredeni = a < b < c;
```

е синтаксички исправна, но семантички не е, Така, за вредности a = 3 и c = 5, ќе даде погрешен резултат бидејќи прво се споредува 3 < b што дава резултат true или false, т. е. вредност 1 или 0. Потоа, се споредува 1 < 5 или 0 < 5 што секогаш дава резултат true. Но за b = 6 треба да се добие резултат false.

Исправната наредба е:

```
bool podredeni = a < b && b < c;
```

Логичките и релациските оператори се користат во **логички изрази** (англ. logical expressions), чија вредност може да биде true или false.

Примери:

```
int alfa = 60, x = 7, a = 3, b = 8, c = 2;
bool p = (alfa > 0) && (alfa < 90);
bool q = (x < -1) || (x > 1);
bool r = !(a < b + 5 * c) && (2 * c + 1 == 3 * b) || (c >= b);
cout << "p = " << p << "\nq = " << q << "\nr = " << r << endl;
```

Со следнава табела се претставени еквивалентни логички изрази наречени **Де Морганови правила** (англ. De Morgan's Laws).

Израз	Еквивалентен израз
!(a && b)	!a || !b
!(a || b)	!a && !b

Изразите во програмите може да бидат и доста сложени и во нив да се користат аритметички, логички и релациски оператори. При нивното пресметување, се почитува приоритетот на операторите, и тоа:

Највисок	негација (NOT) !	унарен +			унарен –
	аритметички	*	/	%	
		+	–		
	релациски	<	<=	>	>=
		==	!=		

	логички	(AND) &&						
		(OR) ||						
Најнизок	доделување	=	+=	–=	*=	/=	%=	

Пресметувањето на логичките изрази се врши одлево надесно, со почитување на приоритетот на операторите и заградите.

Во одредени случаи, C++ врши условна процена (англ. conditional evaluation, или short-circuit) на изразот.

На пример, при пресметувањена изразот:

```
(alfa > 0) && (alfa < 90)
```

за alfa = –30°, левиот подизраз (alfa > 0) добива вредност false и независно од вредноста на десниот подизраз `(alfa < 90)`, вредноста на целиот израз е `false`. Затоа, во случај на израз со операторот &&, десниот подизраз не се прес-метува.

Слично, при пресметување на изразот:

```
 (x < -1) || (x > 1)
```

за x = –2, вредноста на левиот подизраз е true и независно од вредноста на десниот подизраз, целиот израз добива вредност true. Затоа, десниот подизраз не се пресметува.

Пример 1.7.6

Со овој пример се печати табела на логичките функции AND, OR и NOT, пресметани во зависност од вредностите на две логички променливи a и b.

```
1   #include <iostream>
2   using namespace std;
3
4   int main() {    // Logicki tip podatoci
5
6       bool a, b;
7       cout << "-----------------------------------------" << endl;
8       cout << "a    b     a AND b  a OR b   NOT a   NOT b" << endl;
9       cout << "-----------------------------------------" << endl;
10      a = true;
11      b = true;
12      cout << a << "    " << b << "        " << ( a && b ) << "       "
13          << ( a || b ) << "        " << ( !a ) << "       " << ( !b ) << endl;
14      b = false;
15      cout << a << "    " << b << "        " << ( a && b ) << "       "
16          << ( a || b ) << "        " << ( !a ) << "       " << ( !b ) << endl;
17      a = false;
18      b = true;
19      cout << a << "    " << b << "        " << ( a && b ) << "       "
20          << ( a || b ) << "        " << ( !a ) << "       " << ( !b ) << endl;
21      b = false;
```

Слика 1. .6

```
22      cout << a << "     " << b << "         " << ( a && b ) << "         "
23          << ( a || b ) << "         " << ( !a ) << "         " << ( !b ) << endl;
24      cout << "-----------------------------------------" << endl;
25
26      cout << endl;
27      system( "Color 17" );
28      system( "pause" );
29      return 0;
30  }
```

Слика 1. .6 продол ение

По извршување на програмата, излезот е:

```
-----------------------------------------
a     b      a AND b   a OR b    NOT a    NOT b
-----------------------------------------
1     1          1         1          0        0
1     0          0         1          0        1
0     1          0         1          1        0
0     0          0         0          1        1
-----------------------------------------

Press any key to continue . . .
```

Пример 1.7.7

Со овој пример се илустрира користење на логички тип на податоци.

```
1   #include <iostream>
2   using namespace std;
3
4   int main() {    // Logicki tip podatoci
5
6       const double G = 9.81;  // Deklaracija i inicijalizacija na realna konstanta
7       int agolAlfa;           // Deklaracija na celobrojna promenliva
8       double zabrzuvanje;     // Deklaracija na realna promenliva
9       char znak = '*';        // Deklaracija i inicijalizacija na znakovna promenliva
10      bool daNe;              // Deklaracija na logicka promenliva
11      daNe = znak == '*';     // Ako vrednosta na promenlivata znak e *, togas
12                              // logickata promenliva daNe dobiva vrednost 1 (true),
13                              // vo sprotivno dobiva vrednost 0 (false).
14      cout << daNe << ": znakot e " << znak << endl;
15      zabrzuvanje = 9.9;
16      daNe = zabrzuvanje < G; // Ako vrednosta na promenlivata zabrzuvanje e pomala
17                              // od vrednosta na konstantata G, togas logickata
18                              // promenliva daNe dobiva vrednost 1 (true),
19                              // vo sprotivno dobiva vrednost 0 (false).
20      cout << daNe << ": " << zabrzuvanje << " < " << G << endl;
21      agolAlfa = 60;
22      daNe = ( agolAlfa > 0 ) && ( agolAlfa < 90 );
23      cout << daNe << ": " << agolAlfa << " e ostar agol." << endl;
```

Слика 1. .

```
24
25      cout << endl;
26      system( "Color 17" );
27      system( "pause" );
28      return 0;
29  }
```

Слика 1. . продол ение

По извршување на програмата, се добива следниот излез:

```
1: znakot e *
0: 9.9 < 9.81
1: 60 e ostar agol.

Press any key to continue . . .
```

Оператори со битови

C++ има уште една група логички оператори, наречена **оператори со битови** (англ. bitwise operators). Со овие оператори, логичките операции се извршуваат бит по бит.

&	логичко ***И*** за битови	~	комплемент
|	логичко ***ИЛИ*** за битови	<<	поместување во лево
^	исклучиво ***ИЛИ*** за битови	>>	поместување во десно

Табелата на логичките операции &, | и ^ над битовите p и q е:

p	q	p & q	p | q	p ^ q
1	1	1	1	0
1	0	0	1	1
0	1	0	1	1
0	0	0	0	0

Комплемент на бинарен број се добива со негација на битовите, 1 во 0 и 0 во 1. На пример, комплемент на бројот

$m = 000\ldots\ 00110101\ (= 53_{10})$

е бројот

$\bar{m} = 111\ldots\ 11001010\ (= -54_{10})$.

Операторите за поместување се користат за поместување (англ. shifting) на битовите налево или надесно. Затоа, се нарекуваат оператори за поместување или оператори **шифт**. На пример, со m << 1, се поместуваат битовите на бројот m за 1 место налево ($= 000\ldots\ 01101010 = 106_{10}$), а со m >> 1, битовите се поместуваат за 1 место надесно ($= 000\ldots\ 00011010 = 26_{10}$).

Пример 1.7.8

Нека се дадени бинарните броеви a = 00000101 (= 5_{10}), b = 00001110 (= 14_{10}). Да се отпечати вредноста на операциите:

a & b, a | b, a ^ b, ~a, a << 1, a >> 1 и –a >> 1.

```
#include <iostream>
using namespace std;

int main() {    // Operatori so bitovi
    int a = 5, b = 14;
    cout << "-------------" << endl;
    cout << " a=00000101" << endl;
    cout << " b=00001110" << endl;
    cout << "-------------" << endl;
    cout << "a & b   | " << ( a & b ) << endl;
    cout << "a | b   | " << ( a | b ) << endl;
    cout << "a ^ b   | " << ( a ^ b ) << endl;
    cout << "~a      | " << ( ~a ) << endl;
    cout << "a << 1  | " << ( a << 1 ) << endl;
    cout << "a >> 1  | " << ( a >> 1 ) << endl;
    cout << "-a >> 1 | " << ( -a >> 1 ) << endl;
    cout << "-------------" << endl;

    cout << endl;
    system( "Color 17" );
    system( "pause" );
    return 0;
}
```

Слика 1. .8

Со програмата се добива следнава табела:

```
-------------
 a=00000101
 b=00001110
-------------
a & b   | 4
a | b   | 15
a ^ b   | 11
~a      | -6
a << 1  | 10
a >> 1  | 2
-a >> 1 | -3
-------------
```

Кај операторите со битови, логичката операција се извршува над соодветните битови. Резултатот може да биде произволна целобројна вредност. Затоа,

5 & 14 = 00000101 & 00001110 = 00000100 = 4.

Треба да се воочи разликата помеѓу логичките оператори и операторите со битови. Логичките оператори се применуваат над еден или над два операнди, при што секоја вредност различна од нула се третира како логички точна. Притоа, резултатот на операцијата може да биде 0 или 1.

На пример, за вредностите во претходната табела: a && b = 1 и a || b = 1, бидејќи a и b имаат вредност различна од 0.

При користење на логичките оператори && и ||, треба да се внимава да не се заменат со операторите со битови & и |. На пример, за a = 5 и b = 14, изразите:

```
a | b = 15
a || b = 1
```

имаат различен резултат.

Скратена форма на операторите со битови е:

```
&=      |=      ^=      <<=      и      >>=.
```

Задачи за вежбање

1. Да се напише програма со која ќе се пресмета вредноста (true или false, т. е. 1 или 0) на логичкиот израз (a || !b) && !(a || b) за a = true и b = false.
2. Да се напише програма за наоѓање на вредноста на изразот:
   !(a – b < 5 * c) && (b * c % a + 1 == a * b) || (c < b) за a = 7, b = 3 и c = 4.
3. Да се напише програма за наоѓање на производот и количникот на целиот број 12 со 2^5. (*Упатство*: Множење на целиот број m со 2^n се врши со поместување на децималната точка за n места надесно, а делење со поместување на децималната точка за n места налево. Затоа, може да се користат операторите за поместување >> и <<).

Наброив тип на податоци

Податоците од наброив тип може да добиваат вредности од однапред зададено множество (листа) на константи, кои мора да бидат идентификатори, а не броеви. Исто така, знаеме дека вредностите на константите се пишуваат со големи букви.

Наброив тип на податоци се дефинира со зборот **enum** (англ. enumerate), а вредностите во листата се стават во големи загради. По излезниот дел од големите загради, не се става знакот точка и запирка.

Примери:

```
enum osnovnaBoja {CRVENA, ZELENA, SINA, ZOLTA, KAFENA}
enum godisnoVreme {PROLET, LETO, ESEN, ZIMA}
enum tocnoNetocno {TRUE, FALSE}
```

Секое име во листата има свој реден број, и тоа: 0, 1, 2, 3 итн. На пример, редниот број на CRVENA е 0, на ZELENA е 1, на SINA е 2.

Во горниот пример се дефинирани 3 наброиви типови, и тоа: osnovnaBoja, godisnoVreme и tocnoNetocno.

Следниве дефиниции на наброивите типови не се исправни бидејќи константите во листите не се идентификатори, т. е. не се формирани според правилата за идентификатори во C++:

```
enum brojniOcenki {1, 2, 3, 4, 5}
enum znaciOperatori {'+', '-', '*', '/', '%'}
```

Следниве две дефиниции на наброиви типови се исправни:

```
enum denovi {PON, VTO, SRE, CET, PET, SAB, NEDELA}
enum vikend {SABOTA, NEDELA}
```

Но не може да се користат во иста програма бидејќи константата NEDELA се редефинира со втората дефиниција.

```
denovi d = NEDELA;
vikend v = NEDELA
```
enum vikend::NEDELA = 1
Search Online
a value of type "vikend" cannot be used to initialize an entity of type "denovi"

Декларацијата на променливи од наброив тип се врши со наведување на типот, а по него се наведува листата на променливи.

Примери:

```
osnovnaBoja c, z, s;
godisnoVreme es, zi, le;
```

Овие променливи може да добиваат вредности само од оние константи што се наведени во листата при дефинирање на типот:

```
s = SINA;
zi = ZIMA;
```

Променливите од наброив тип не може да се печатат. При печатење на променливите s и zi, се печатат нивните редни броеви во листата со дефиниција:

```
s = 2
zi = 3
```

бидејќи редниот број на SINA е 2, а на ZIMA е 3.

Променливите од наброив тип не може да се инкрементираат имплицитно. На пример, следнава наредба:

```
s = s + 1;
```

не може ниту да се компајлира. Веднаш се јавува грешка.

```
s = s + 1;
```
cou
Error: a value of type "int" cannot be assigned to an entity of type "osnovnaBoja"

Грешката се јавува бидејќи при собирање со 1, променливата s принудно се конвертира во цел број 2 (редниот број во дефиницискатa листа) и се собира со 1. Резултатот е целобројна вредност 3 која не може да се додели на наброивата променлива s.

Ова може да се избегне со експлицитна конверзија

```
s = osnovnaBoja(s + 1);
```

по која променливата s ќе добие вредност ZOLTA.

На константите од наброив тип може да им се задаваат вредности (место редни броеви) и при дефиниција на типот.

Примери:

```
enum ocenka {ODLICEN = 5, MNDOBAR = 4, DOBAR = 3, DOVOLEN = 2};
enum meseci {JANUARI = 1, FEVRUARI, MART, APRIL, MAJ, JUNI, JULI,
             AVGUST, SEPTEMVRI, OKTOMVRI, NOEMVRI, DEKEMVRI}
```

Во вториот пример, редните броеви од JANUARI до DEKEMVRI ќе бидат од 1 до 12, а не од 0 до 11.

Податоците од наброив тип може да се споредуваат. Притоа, се споредуваат нивните редни броеви во дефиницискитe листи.

Може да се декларира и анонимен набројив тип (англ. anonymous enumeration type). На пример:

```
enum {jabolko, krusa, praska, sliva, cresa, visna, kajsija};
```

Дефиницијата на набројив тип се врши пред функцијата main(), ***слика 1. .9***.

Пример 1.7.9

Со овој пример се илустрираат дефиниција на набројив тип и декларација на променливи од набројив тип.

По извршување на програмата од ***слика 1. .9***, се добива следниов излез:

```
cBoja = 0, sBoja = 5
Dali se boite isti? 0
Vrednosta na zi = ZIMA e 3
```

```
1   #include <iostream>
2   using namespace std;
3
4   enum osnovnaBoja {
5       CRVENA, ZELENA, SINA
6   };
7   enum godisnoVreme {
8       PROLET, LETO, ESEN, ZIMA
9   };
10
11  int main() {    // Podatoci od nabroiv tip
12      osnovnaBoja cBoja, sBoja;
13      godisnoVreme zi = ZIMA;
14      bool tocnoNetocno;
15
16      cBoja = CRVENA;
17      sBoja = SINA;
18      tocnoNetocno = cBoja == sBoja;
19      cout << "cBoja = " << cBoja << ", sBoja = " << sBoja << endl;
20      cout << "Dali se boite isti? " << tocnoNetocno << endl;
21
22      cout << "Vrednosta na zi = ZIMA e " << zi << endl;
23
24      cout << endl;
25      system( "Color 17" );
26      system( "pause" );
27      return 0;
28  }
```

Слика 1. .9

Задачи за вежбање

1. Напишете програма во која ќе дефинирате наброив тип denoviVoNedelata, чија листа на константи ќе ги содржи имињата на деновите од PONEDELNIK до NEDELA. Потоа, декларирајте седум променливи од типот denoviVoNedelata и на секоја доделете ѝ по една константа од наброивиот тип denoviVoNedelata по случаен избор. Декларирајте уште една променлива од типот denoviVoNedelata, на пример, den и доделете ѝ вредност SREDA. Потоа, декларирајте логичка променлива tocnoNetocno, на која ќе ѝ доделувате вредност што се добива од споредбата на содржината на променливата den со секоја од седумте променливи. Отпечатете ги вредностите што ќе ги добива променливата tocnoNetocno. (Видете ја програмата од ***слика 1. .9***).
2. Да се напише програма во која ќе декларирате променлива од наброив тип casoviDnevno:
```
enum casoviDnevno {
   PON = 4, VTO = 6, SRE = 5, CET = 5, PET = 3
};
```
Да се пресмета вкупниот број на часови во неделата.
3. Да се напише програма во која ќе декларирате променлива од наброив тип ocenki. Дефиницијата на типот ocenki содржи имиња на 12 предмети со оценка за секој предмат. На пример, {MAK = 4, INF = 5...}. Да се пресмета просечниот успех на ученикот.

Стрингови

Во поднасловите **Наредба за печатење** и **Типови на податоци** од поточката **1.6 Вовед во C++**, се запознавме со поимот **стринг** (англ. string). Стринговите се податоци составени од знаковни податоци како секвенци од знаци. Таквите податоци се третираат како посебни структури составени од простиот тип char, а се наречени тип string.

Овие податоци се нарекуваат и **текстуални податоци** бидејќи нивната вредност е текст ставен во наводници, т. е. текстуална (стринг) константа. На пример, стринг-константи во C++ се:

```
"Zdravo, kako ste ? "
"ul. Goce Delcev broj 123."
"Macedonia bibliska zemja."
```

Стринг-константите во C++ може да содржат букви (големи и мали од абецедата), цифри и специјални знаци, како +, –, *, / и $.

Основните карактеристики на стринг-константите се:

- Се запишуваат во наводници.
- Секој знак има индекс, почнувајќи од најлевиот кој има индекс 0.
- Должината на текстуалната константа е еднаква на бројот на знаци, вклучувајќи ги и празните места.

- Максималната должина на една стринг-константа е 2 048 бајти.

Променливи од типот string може да се декларираат како и променливи од кој било прост тип и ним може да им се доделуваат стринг-константи.

Примери:

```
string moeIme;              // Deklaracija na promenlivata moeIme
string daNe;
string datumDenes = "01.01.2017"; //Deklaracija i inicijalizacija
string datumDenes("01.01.2017");  // ili (isto) deklaracija i
                                  // inicijalizacija
string dzvezdi(60, '*');    // Deklaracija i inicijalizacija
```

Доделување вредност на декларирана променлива од типот string се врши со знакот за доделување =.

Примери:

```
moeIme = "Gjorgji Jovancevski";
daNe = 'D';                         // Ova e dozvoleno
datumUtre = "21 juni 2053 godina";
```

Рековме дека секој знак во стрингот има свој индекс. На пример, moeIme[0] е 'G', moeIme[5] е 'j' , datumUtre[11] е '3' итн.

Да нагласиме дека податоците "A" и 'A' не се исти. Првиот податок е од типот string, додека вториот податок е од типот char.

Со стринговите може да се вршат разни операции, како: спојување (ја наведовме во поднасловот **Типови на податоци)**, споредување (англ. comparation) на два стринга, одредување должина (англ. length) на стринг и многу други.

Овие операции се извршуваат со функции од библиотеката <string> на *стандардната библиотека* на ***C*** . Затоа, за да ги користиме во програмите, а и за да можеме да декларираме променливи од типот string, мора на почетокот од програмата да се вклучи оваа библиотека, со директивата #include:

```
#include <iostream>
#include <string>
using namespace std;
```

Пример 1.7.10

Со програмата на ***слика 1. .10*** е илустрирано користењето на стринг променливи и стринг-константи.

Еден излез од извршување на програмата е:

```
Dva najgolemi makedonci bile:
Aleksandar Filip MAKEDONSKI  i Goce Nikola MAKEDONSKI
true: Aleksandar Filip MAKEDONSKI ziveel pred Goce Nikola MAKEDONSKI

Press any key to continue . . .
```

```cpp
#include <iostream>
#include <string>
using namespace std;

const string prezime = "MAKEDONSKI";
const string praznoMesto = " ";

int main() {    // String podatoci

    string imeMoe = "Aleksandar Filip";
    string imeTvoe = "Goce Nikola";

    string najgolemiMakedonci = imeMoe + praznoMesto + prezime;
    najgolemiMakedonci += " i ";
    najgolemiMakedonci += imeTvoe + praznoMesto + prezime;
    cout << "Dva najgolemi makedonci bile: " << endl;
    cout << najgolemiMakedonci << endl;

    bool daNe = imeMoe < imeTvoe;
    cout << boolalpha << daNe << ": " << imeMoe << praznoMesto
        << prezime << " ziveel pred " << imeTvoe << praznoMesto
        << prezime << endl;

    cout << endl;
    system( "Color 17" );
    system( "pause" );
    return 0;
}
```

Слика 1. .10

Задачи за вежбање

1. Да се напише програма во која ќе се декларираат две стринг променливи на кои ќе им се доделат вредности Вашето име и Вашето презиме. Да се декларира и знаковна променлива на која ќе ѝ се додели вредност првата буква од името на Ваш родител. Да се спојат трите променливи и да се отпечатат во форма како следниов пример:
   `Gjorgji J. Jovancevski`
2. Да се напише програма во која ќе се декларираат стринг променливи и ќе им се доделат следниве стринг-константи: "GRAF", "LOG", "BIO", "IJA" и "GEO". Потоа, со спојување, да се формираат и отпечатат зборовите: BIOGRAFIJA, BIOLOGIJA, GEOGRAFIJA и GEOLOGIJA.
3. Да се напише програма во која ќе се декларираат две стринг променливи и две знаковни променливи и ќе им се доделат константите: "KAL", "LAK", 'A' и 'B'. Потоа, со нивно спојување да се формира најдолг збор палиндром кој има

значење. (Палиндром е збор или реченица, кој се чита исто и однапред и одназад. На пример, палиндроми се: око, довод, елелесевеселеле итн.).

Преименување на типот на податоци

Честопати, заради читливоста на програмата и полесно снаоѓање во неа, за имињата на вградените типови на податоци пожелно е да се користат синоними (други имиња) кои ќе нè потсетуваат на нешто.

На пример, со следниве декларации:

```
int i, j, k;
double x, y, z;
char a, b;
bool dane, najdeno;
```

се декларираат променливи од типот int, double, char и bool.

Во C++ може да се воведе и друго име за некој постоен тип. (Притоа, не се креира нов тип). Опсегот на вредности и операциите со новото име на типот се идентични со постојниот тип.

За преименување на тип, се користи клучниот збор **typedef**.

На пример:

```
typedef int celobrojni;
typedef double realni;
typedef char znaci;
typedef bool logicki;
```

Потоа, може да се изврши декларирање со типот celobrojni рамноправно како со типот int, со типот realni рамноправно како со типот double итн.

На пример:

```
int broj = 5;
celobrojni broj1 = 8;
char znak = '$';
znaci znak1 = '@';
bool daNe = 1;
logicki daNe1 = true;
cout << broj << ' ' << broj1 << endl;
cout << znak << ' ' << znak1 << endl;
cout << daNe << ' ' << boolalpha << daNe1 << endl;
```

Излезот од горниот програмски сегмент е:

```
5 8
$ @
1 true

Press any key to continue . . .
```

Автоматско одредување на типот

Типот на променлива при нејзината декларација може да се одреди според типот на податокот со кој таа се иницијализира или ѝ се доделува вредност. Тоа се прави со клучниот збор **auto**.

На пример, ако декларацијата на променливите во програмата од ***слика 1. .10*** се зададе со зборот auto, програмата ќе се извршува коректно:

```
auto imeMoe = "Aleksandar Filip";
auto imeTvoe = "Goce Nikola";
auto najgolemiMakedonci = imeMoe + praznoMesto + prezime;
auto daNe = imeMoe < imeTvoe;
```

Клучниот збор auto најчесто се користи кога изразот за иницијализација на променливата е сложен, т. е. кога во него се користат разни типови на податоци и на функции и кога не може веднаш да се одреди типот на резултатот при пресметување на изразот. Преведувачот го одредува типот на резултатот при пресметување на изразот.

Прашања за проверка на знаењето

1. Кои типови на целобројни податоци (според опсегот на вредностите) ги знаете?
2. Какви вредности добиваат целобројните променливи декларирани како unsigned?
3. Што ќе се случи ако на променливата short ѝ доделите вредност 100 000?
4. Наведете ги операторите за операции со цели броеви.
5. Наведете неколку примери за декларација и декларација и иницијализација на променливи од целоброен тип.
6. Наведете ги аритметичките оператори за цели броеви во скратена форма.
7. Што значи операторот % = ?
8. За кои податоци велиме дека се од реален тип?
9. Кои типови на реални податоци ги има во C++?
10. Наведете неколку примери за декларација и декларација и иницијализација на променливи од реален тип.
11. Објаснете го претставувањето на децималните податоци со подвижна точка.
12. Објаснете ги форматите за печатење реален тип на податоци.
13. Претставете го бројот 34400.811 во е-формат.
14. Наведете неколку примери за декларација и декларација и иницијализација на променливи од реален тип.
15. Што е имплицитана конверзија на типот и како се нарекува поинаку?
16. Што е експлицитна конверзија на типот и како се нарекува поинаку?
17. Во кои типови може да се изврши имплицитна конверзија на типот int?

18. Како се врши експлицитна конверзија од повисок во понизок тип на податок?
19. Напишете наредба за конверзија на изразот 0.12 * 5.3 + 123 во типот int?
20. Објаснете ја наредбата: x = x = y;
21. Објаснете што значи инкрементирање, а што декрементирање.
22. Кој е оператор за инкрементирање, а кој за декрементирање?
23. Наведете примери за декларација на знаковни променливи.
24. Колкав мемориски простор зафаќа типот char?
25. Кои логички константи ги знаете? Зошто нема други?
26. Наведете ги логичките оператори.
27. Кои релациски операции ги знаете? Наведете ги соодветните оператори.
28. Кој ќе биде резултатот од изразот: !(a && b) || (!a || b), за a = true и b = false?
29. Дали се исправни следнава декларација и иницијализација:
    bool daNe = x > y < z;?
30. Наведете ги операторите со битови.
31. Најдете го резултатот од операциите:
    a & b, a | b и a ^ b, за a = 00010110 и b = 00010101.
32. Какви се вредностите на податоците од наброив тип?
33. Каде се става дефиницијата на наброив тип на податок?
34. Наведете пример за дефиниција на наброив тип и за декларација на променлива од тој тип.
35. Како се печатат податоци од наброив тип?
36. Дали текстуалниот тип на податоци спаѓа во прости типови?
37. Напишете пример за декларација на стринг-константа.
38. Напишете пример за декларација на стринг променлива.
39. Ако е дефинирано:
    typedef int celi;
    тогаш од кој прост тип ќе бидат променливите декларирани со
    celi a, b, c; ?
40. Кој збор се користи при иницијализација на променлива за да може преведувачот да го одреди нејзиниот тип?

1.8 Читање и печатење податоци

Читање и печатење бројни податоци

Читањето податоци во C++ е организирано како **поток** (англ. stream) од знаци. Стандардниот влезен и излезен поток се вршат со помош на функции (рутини[41], англ. routines), кои се наоѓаат во библиотеката <iostream>, која, пак, се вклучува во секоја програма со директивата:

```
#include <iostream>
```

Во библиотеката <iostream> се дефинирани операторите <<[42] и >>, наредбите cin, cout, endl и други, потребни за извршување на влезни и на излезни операции.

Исто така, во програмата мора да се вклучат и други библиотеки кои содржат функции за извршување одредени операции. На пример, ако во програмата користиме стрингови, треба да се вклучи библиотеката <string> со директивата #include:

```
#include <string>
```

Програмските линии кои почнуваат со директивата `#include` не ги преведува преведувачот, туку се обработуваат со посебна програма наречена **претпроцесор** (англ. preprocessor). Таа ги вклучува во програмата сите датотеки (англ. files) кои се наведени во директивите #include. Овие датотеки се стават во аглести загради, со кои се информира претпроцесорот нив да ги бара во т.н. директориум include од ***стандардната библиотека на C*** . Тие се нарекуваат **хедер-датотеки** (англ. header files).

Операторите за читање и печатење податоци се:

>>	влезен оператор,
<<	излезен оператор.

Операторот << значи „прати на“, а операторот >> значи „преземи од“.

Наредбите за влез и за излез се:

cin	Наредба за внесување вредности преку тастатурата (за стандарден влезен поток).
cout	Наредба за печатење вредности на екранот (за стандарден излезен поток).

При читање на вредности на податоците, се прескокнуваат празните места, табулаторите и знаците за крај (англ. enter). Читањето на вредностите започнува од првиот знак кој не е бланко.

На пример, ако се декларирани променливите i и x:

```
int i;
double x;
```

[41] Кратки програми.
[42] Овој оператор го користевме и досега за печатење.

читањето вредности за нив од тастатурата се врши со наредбата:

```
cin >> i >> x;
```

Ако на променливите celBroj и realenBroj им доделиме вредности:

```
celBroj = 123;
realenBroj = 456.78;
```

тогаш со наредбата:

```
cout << celBroj << realenBroj;
```

ќе се отпечати:

```
123456.78
```

За разделување на отпечатените вредности може да се користи едно празно место како знаковна константа. Така, со следнава наредба:

```
cout << celBroj << ' ' << realenBroj << endl;
```

ќе се отпечати:

```
123 456.78
```

endl (кој го користевме и претходно) е оператор за крај на линијата, т. е. служи за преминување во следната линија.

Ако треба да се остави повеќе од едно празно место, обично се користи стринг-константа.

На пример, со наредбата:

```
cout << celBroj << "    " << realenBroj << endl;
```

ќе се отпечати:

```
123    456.78
```

За печатење во нов ред, се користи знакот '\n', кој го објаснивме и користевме во досегашните примери.

На пример, со двете наредби:

```
cout << celBroj << '\n' << realenBroj << '\n';
cout << celBroj << endl << realenBroj << endl;
```

ќе се отпечати исто:

```
123
456.78
```

Неколку специфични карактери кои имаат специјално значење кај наредбата за излез мора да се отпечатат, со користење на излезни (ескејп) секвенци, наведени во поднасловот **Наредба за печатење** од потточката **1.6 Вовед во C++**.

Коментарите во C++ кои се во една линија се пишуваат со ставање на знакот // пред коментарот.

Коментар кој не е до крајот на линијата се става помеѓу ознаките /* и */.

На пример:

```
/*     So ovaa programa se izrabotuva najednostavna
       posetnica, so osnovnite podatoci:
       - ime na firmata
       - telefon
       - e-mail
*/
```

Претпроцесорската директива:

```
#include <iostream>
```

му кажува на претпроцесорот да ја вклучи датотеката <iostream>, која содржи дефиниции на функции за влезно-излезни операции.

Променливите во C++ може да се декларираат каде било во програмата, а може да се користат само по линијата на декларација, а не пред неа.

Наредбата:

```
cout << "Vnesi prv cel broj: ";
```

го користи стандардниот излезен поток cout и операторот << за да се отпечати наведениот текст на екранот од мониторот. Кога претходната наредба ќе се изврши, таа испраќа поток од знаци „Vnesi prv cel broj“ на стандардниот излез cout. Оваа наредба би можеле да ја прочитаме како: „на cout се испраќа текстот „Vnesi prv cel broj“ како поток од знаци“.

Следната наредба:

```
cin >> broj1;
```

го користи стандардниот влезен поток cin и операторот >> за преземање податоци кои се внесуваат преку тастатурата, во форма на низа од знаци. Оваа наредба може да се прочита како „cin го чита потокот од знаци кој се внесува преку тастатурата и му го доделува на broj1“. Во конкретниов случај, знаците кои се внесуваат се цифри на бројот, кои се конвертираат во цел број и тој му се доделува на целобројната променлива broj1.

Ќе наведеме неколку примери.

Примери

Пример 1.8.1

На ***слика 1.8.1*** е дадена едноставна програма во C++ за собирање на два броја, во која ќе илустрираме некои од особините разгледани во претходниот дел од текстот.

```
1   // Sobiranje na dva broja
2   #include <iostream>
3   using namespace std;
4
5   int main() {                // Pocetok na glavnata funkcija main().
6       int broj1;              // Deklaracija na celobrojnata promenliva broj1.
7       cout << "Vnesete prv cel broj: ";
8       cin >> broj1;           // Citanje vrednost na promenlivata broj1.
9       int broj2, zbir;        // Deklaracija na celobrojnite
10                              // promenlivi broj2 i zbir.
11      cout << "Vnesete vtor cel broj: ";
12      cin >> broj2;           // Citanje vrednost na promenlivata broj2.
13      zbir = broj1 + broj2;   /* Naredba za sobiranje na vrednostite na
14                              dve promenlivi */
15      cout << "Zbirot e " << zbir << endl;
```

Слика 1.8.1

```
16
17          cout << endl;
18          system("Color 17");
19          system("pause");
20          return 0;
21
22      } // kraj na glavnata funkcija main()
```

Слика 1.8.1 продол ение

Пример 1.8.2

Да се декларираат, внесат и отпечатат податоци од целоброен тип.

```
1   #include <iostream>
2   using namespace std;
3
4   int main() {
5       // Citanje i pecatenje celobrojni podatoci vneseni preku tastatura
6       unsigned short datumMesec;
7       unsigned short datumDen;
8       unsigned int datumGodina;
9       long long brojSo19cifri;
10      cout << "Vnesete den: ";    cin >> datumDen;
11      cout << "Vnesete mesec: ";  cin >> datumMesec;
12      cout << "Vnesete godina: "; cin >> datumGodina;
13      cout << endl;
14      cout << "Denes e " << datumDen << "." << datumMesec << "."
15          << datumGodina << endl;
16      cout << "\nVnesete 19 cifri: "; cin >> brojSo19cifri;
17      cout << "\nGo vnesovte brojot: " << brojSo19cifri << endl;
18
19      cout << endl;
20      system( "Color 17" );
21      system( "pause" );
22      return 0;
23  }
```

Слика 1.8.2

Излезот од програмата на ***слика 1.8.2*** е:

```
Vnesete den: 01
Vnesete mesec: 01
Vnesete godina: 2017

Denes e 1.1.2017

Vnesete 19 cifri: 1234567890123456789

Go vnesovte brojot: 1234567890123456789

Press any key to continue . . .
```

Пример 1.8.3

Да се декларираат, внесат и отпечатат податоци од реален тип.

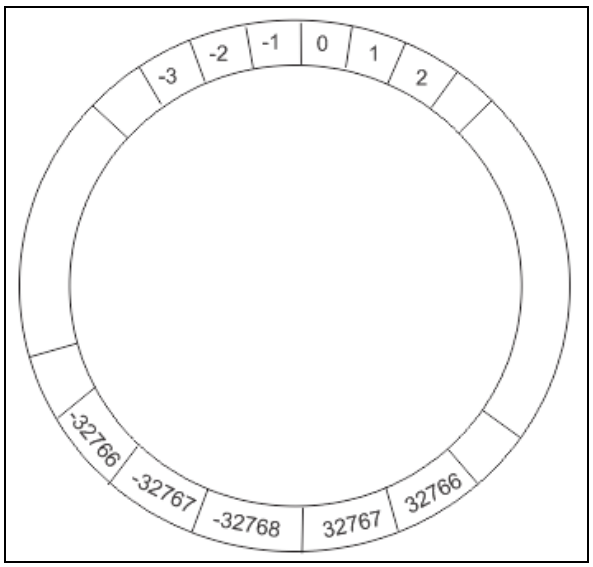
Во следната програма (***слика 1.8.***) се читаат два цели броја и се печати нивната аритметичка средина.

```
#include <iostream>
using namespace std;

int main() {
    // Citanje i pecatenje realni podatoci vneseni preku tastatura
    float brojPrv;
    double brojVtor, brojSredina;
    cout << "Vnesete prv decimalen broj: ";     cin >> brojPrv;
    cout << "Vnesete vtor decimalen broj: ";    cin >> brojVtor;
    brojSredina = ( brojPrv + brojVtor ) / 2;
    cout << "\nSredna vrednost od " << brojPrv << " i " << brojVtor
        << " e " << brojSredina << endl;

    cout << endl;
    system( "Color 17" );
    system( "pause" );
    return 0;
}
```

Слика 1.8.

Пример 1.8.4

На ***слика 1.8.4*** е даден пример за пречекорување на опсегот на броеви, зададен според типот на променливите.



Задачата илустрира некоректно користење на променливата vkupno бидејќи добива вредност која е поголема од опсегот на вредности, според дефиницијата на типот. Променливата vkupno е од типот short, чија максимална вредност е 32 767, а во пресметката добива вредност 60.000, која преминува во негативниот опсег на целите броеви од типот short (–32768 до –1).

Ова ќе биде појасно ако опсегот на целобројниот тип на податоци го претставиме затворено (кружно). Од него се гледа дека бројот 32 768 (кој е над-вор од опсегот) ќе се претстави со

бројот –32 768, а бројот 32 769, ќе се прет-стави со бројот –32 767. Затоа, за вредност на променливата vkupno ќе се добие:

vkupno = –32 768 + (60 000 – 32 767) – 1 = –5 536

Оваа појава на излегување на вредноста на променливата надвор од опсегот на типот од кој е декларирана променливата се нарекува **преполнување** (англ. overflow).

```
1   #include <iostream>
2   using namespace std;
3
4   int main() {
5       short edinecnaCena, vkupno; /* Deklaracija na promenlivi */
6       short parcinja = 1000;      /* Deklaracija so inicijalizacija*/
7       edinecnaCena = 60;          /* Dodeluvanje */
8       vkupno = edinecnaCena * parcinja;
9       cout << "Vkupnata cena za " << parcinja;
10      cout << " parcinja po " << edinecnaCena << " denari e "
11          << vkupno << " denari." << endl;
12
13      cout << endl;
14      system( "Color 17" );
15      system( "pause" );
16      return 0;
17  }
```

Слика 1.8.4

```
Vkupnata cena za 1000 parcinja po 60 denari e -5536 denari.

Press any key to continue . . .
```

Читање и печатење знаковни и стринг-податоци

Во претходните разгледувања рековме дека стринговите се низи од знаци, со кои може да се извршуваат операции, како: спојување, споредување и други.

Спојувањето се изведува со знакот +, а операндите може да бидат:

- Знаковни или стринг-литерали:

```
' ', '@', "programiranje vo"
```

- Именувани знаковни или стринг-константи декларирани со зборот const:

```
const char prazno = ' ';
const string plusplus = "++";
```

- Декларирани знаковни или стринг променливи:

```
char CBukva = 'C';
string naslov = "Osnovi na";
```

Со наредбата:

```
naslov = naslov + " " + "programiranje vo" + ' ' + "jazikot"
       + prazno + CBukva + plusplus;
```

променливата naslov ќе добие вредност:

```
"Osnovi na programiranje vo jazikot C++."
```

При спојување на стринг-константи, барем еден од операндите на знакот + мора да биде стринг променлива или именувана константа. Во спротивно, ќе се појави грешка.

```
string ab = "aaa" + "bbbb";
```

expression must have integral or unscoped enum type

Правилно е:

```
string a = "aaa";
string ab = a + "bbbb";
//      ili
string ab = string("aaa") + "bbbb";
```

Да напоменеме дека бројни податоци (од типот int, float, double) не може да се спојуваат со стрингови.

Пример 1.8.5

На ***слика 1.8.5*** се илустрирани читање и печатење знаковни и стринг променливи.

```
1   #include <iostream>
2   #include <string>
3   using namespace std;
4
5   int main() {
6       char ednaBukva;
7       string moeIme, moePrezime, moeImeIPrezime;
8       cout << "Vnesete edna bukva: "; cin >> ednaBukva;
9       cout << "Vnesete ime: ";    cin >> moeIme;
10      cout << "Vnesete prezime: ";    cin >> moePrezime;
11      cout << "Dali bukvata sto ja vnesovte: " << ednaBukva << endl;
12      cout << "se sodrzi vo imeto : " << moeIme << endl;
13      cout << "ili vo prezimeto: " << moePrezime << endl;
14      cout << endl;
15      cout << "Vnesete go imeto i prezimeto: ";
16      char znakZaKraj;
17      cin.get( znakZaKraj );
18      getline( cin, moeImeIPrezime );
19      cout << "Dali bukvata sto ja vnesovte: " << ednaBukva << endl;
20      cout << "se sodrzi vo imeto ili prezimeto: "
21          << moeImeIPrezime << endl;
22
23      cout << endl;
24      system( "Color 17" );
25      system( "pause" );
26      return 0;
27  }
```

Слика 1.8.5

Излезот од програмата е:

```
Vnesete edna bukva: J
Vnesete ime: Gjorgji
Vnesete prezime: Jovancevski
Dali bukvata sto ja vnesovte: J
se sodrzi vo imeto : Gjorgji
ili vo prezimeto: Jovancevski

Vnesete go imeto i prezimeto: Gjorgji Jovancevski
Dali bukvata sto ja vnesovte: J
se sodrzi vo imeto ili prezimeto: Gjorgji

Press any key to continue . . .
```

Забелешка: Иако при внесување на податок за стринг променливата moeImeIprezime се внесува стрингот „Gjorgji Jovancevski“, при печатење на истата променлива, се печати само „Gjorgji“. Ова ќе го објасниме во продолжение.

Форматирано печатење

При печатење на податоци, тие се разделуваат со едно или повеќе празни места, со користење на излезната (ескејп) секвенца '\t' за печатење на следната табулаторна позиција или со излезната секвенца '\n' за печатење во нов ред. Исто така, за печатење во нов ред се користи и манипулаторот endl.

За форматирање на податоците при печатење, се користат и други манипулатори. Некои од нив се дефинирани во библиотеката <iostream>, а некои се дефинирани во библиотеката <iomanip>, која (како и <iostream>) треба да се вклучи на почетокот од програмата:

```
#include <iostream>
#include <iomanip>.
```

Во ***табела 1.8.1*** се дадени неколку најчесто користени манипулатори чие дејство е во наредбата за печатење cout.

Манипулатор	Дејство	Опис
endl	моментално	Нова линија (исто како '\n ').
setw(n)	на следниот податок	Поставување минимална ширина на поле за печатење на следниот податокот во наредбата за излез. (Ако е бројот со повеќе цифри, се печати во повеќе колони). Се подразбира (по дифолт) (англ. by default) дека податокот е десно наслонет.
width(n)	на следниот податок	Исто со setw(n).
left	на следниот податок	Лево наслонување во полето поставено со setw(n).

right	на следниот податок	Десно наслонување (по дифолт) во полето поставено со setw(n).
dec	на следниот податок	Бројот се печати во децималниот формат (основа 10).
oct	на следниот податок	Бројот се печати во октални от формат (основа 8).
hex	на следниот податок	Бројот се печати во хексадецималниот формат (основа 16).
setprecision(n)	сите следни податоци	Поставување број на цифри кои ќе се печатат по децималната точка.
fixed	сите следни податоци	Печатење на децималните броеви во f-формат. Дифолт прецизноста е 6 цифри.
scientific	сите следни податоци	Печатење на децималните броеви во e-формат.
uppercase	сите следни податоци	За печатење на децималните броеви во e-формат, но со големо E. На пример, 1.234E+05. Ефектот се поништува со манипулаторот nouppercase.
boolalpha	сите следни податоци	Се печатат константите true или false.
noboolalpha		Се печатат константите 1 или 0.

Табела 1.8.1

Специфичности при читање податоци

Рековме дека излезот и влезот на податоци во програмите се врши преку потоци од знаци. Притоа, на почетокот од програмата треба да се вклучи библиотеката **<iostream>**, која содржи дефиниција на два типа на променливи: **ostream** и **istream**.

Типот ostream е поврзан со **излезниот поток** (англ. output stream), додека типот istream е поврзан со **влезниот поток** (англ. input stream) на податоци.

Во библиотеката <iostream> се декларирани две променливи од овие типови:

```
istream cin;
ostream cout;
```

Променливата cout заедно со **операторот за вметнување** (инсертирање) (англ. insertion operator) <<, е поврзана со стандардниот излезен уред на компјутерот, најчесто мониторот. Притоа, со една наредба за печатење, вредностите на променливите и константите кои треба да се отпечатат се вметнуваат во излезен поток.

На пример, со следниов програмски сегмент:

```
int m;
double b;
m = 12;
```

```
b = 34.5;
cout << m << b << endl;
```

во излезниот поток се вметнуваат вредностите на променливите m и b и на мониторот ќе се отпечати:

```
1234.5
Press any key to continue . . .
```

Слично, променливата cin заедно со **операторот за издвојување** (екстракција) (анг. extraction operator) >>, е поврзана со стандардниот влезен уред на компјутерот, најчесто тастатурата. Притоа, од влезниот поток се издвојуваат податоци кои се доделуваат на соодветни променливи во програмата.

На пример, ако во претходниот програмски сегмент додадеме наредба за читање на m и b:

```
cin >> m >> b;
cout << m << b;
```

и преку тастатурата внесеме влезен поток:

```
6 7.8
```

со притискање на копчето Enter, на променливите m и b ќе им се доделат вредностите 6 и 7.8. Излезниот поток ќе биде 67.8:

```
1234.5
6 7.8
67.8
Press any key to continue . . .
```

Напомена Честопати, се збунуваме на која страна треба да биде операторот, << или >>. Треба да се запамти дека операторот треба да се насочи кон *излез* << (за печатење) или од *влез* >> (за читање).

Забележувате дека во влезниот поток податоците 6 и 7.8 се разделени со празно место, што е правилно. Ако немаше празно место, тогаш на m ќе ѝ се доделеше вредност 67, а на b вредност 0.8.

Истиот влезен поток:

```
6 7.8
```

правилно ќе се прочита и со две наредби:

```
cin >> m;
cin >> b;
```

Со притискање на копчето Enter на крајот од влезниот поток, се генерира знак за крај на тековната линија и се преминува на нова линија. Затоа, тој се нарекува **знак за нова линија** (англ. newline character). При читање на влезниот поток, се чита додека не се најде знакот за нова линија.

Слично, излезниот поток се печати додека не се дојде до знакот за крај, кој (видовме во досегашните примери) се поставува со манипулаторот endl или со ескејп секвенцата '\n'. Кога ќе се дојде до него, покажувачот на екранот преминува во нова линија.

За да се прочита и празно место како знак, се користи посебна наредба:

```
cin.get(znakovnaPromenliva);
```

Оваа наредба е поврзана со типот istream, во кој рековме дека е декларирана променливата cin. Бидејќи cin е специјална променлива, со неа се поврзани повеќе функции, кои се дефинирани во библиотеката <iostream>. Една од тие функции е и функцијата get(), која се поврзува со променливата cin со т.н. **оператор точка** (англ. dot operator). Затоа, наредбата за читање се состои од променливата cin, точка и функцијата get().

На пример, за да се прочита влезниот поток од претходниот пример:

```
6 7.8
```

и празното место меѓу нив, треба да го напишеме следниот програмски сегмент:

```
cin >> m;
cin.get(z);
cin >> b;
cout << m << ' ' << z << ' ' << b << endl;
```

Притоа, променливата z ќе добие вредност празно место и излезот ќе биде:

```
6 7.8
6   7.8

Press any key to continue . . .
```

Поголем проблем со празни места се јавува при читање стрингови кои содржат и празни места. Така, во примерот од ***слика 1.8.5,*** во последниот дел:

```
cout << "Vnesete go imeto i prezimeto: ";cin >> moeImeIPrezime;
cout << "Dali bukvata sto ja vnesovte: " << ednaBukva << endl;
cout << "se sodrzi vo imeto ili prezimeto: " << moeImeIPrezime << endl;
```

се внесува цело име и презиме, а се печати само името:

```
Vnesete go imeto i prezimeto: Gjorgji Jovancevski
Dali bukvata sto ja vnesovte: J
se sodrzi vo imeto ili prezimeto: Gjorgji

Press any key to continue . . .
```

Ова се случува бидејќи со операторот >> се чита до првото празно место и, иако е внесено името и презимето, се чита само до празното место меѓу името и презимето. Затоа, променливата moeImeIPrezime добива вредност „Gjorgji“, која и се печати.

За правилно читање на стрингови, се користи функцијата getline(), со која се чита цела влезна линија до знакот за нова линија (ставен со Enter). Притоа, маркерот за читање се поставува на почетокот на следната линија за читање податоци.

Синтаксата на наредбата е:

```
getline(cin, stringPromenliva);
```

На пример, во програмата од ***слика 1.8.5***, наредбата:

```
cin >> moeImeIPrezime;
```

ќе ја замениме со наредбите:

```
char znakZaKraj;
cin.get(znakZaKraj);        // se cita znakot '\n'
getline(cin, moeImeIPrezime);
```

и излезот ќе биде исправен:

```
Vnesete edna bukva: J
Vnesete ime: Gjorgji
Vnesete prezime: Jovancevski
Dali bukvata sto ja vnesovte: J
se sodrzi vo imeto : Gjorgji
ili vo prezimeto: Jovancevski

Vnesete go imeto i prezimeto: Gjorgji Jovancevski
Dali bukvata sto ja vnesovte: J
se sodrzi vo imeto ili prezimeto: Gjorgji Jovancevski

Press any key to continue . . .
```

Да појасниме. Со наредбата:

```
getline(cin, moeImeIPrezime);
```

се чита внесениот стринг и ѝ се доделува на стринг променливата moeImeIPrezime. Меѓутоа, во претходната наредба за читање се користи операторот >>, со кој се чита до знакот за крај, но не и знакот за крај. Затоа, со наредбата:

```
cin.get(znakZaKraj);
```

се чита знакот за крај и маркерот за читање се позиционира на почетокот на новата линија за читање.

Напомена Ако за читање податоци во целата програма се користи функцијата getline(), тогаш не е потребно посебно читање на знакот за крај.

Пример 1.8.6

Со овој пример се илустрира користењето на функциите get() и getline(). Програмата пресметува просечна оценка на два ученика.

```
1   #include <iostream>
2   #include <string>
3   #include <iomanip>
4   using namespace std;
5
6   int main() {    // Ilustracija na funkciite get() i getline()
7
8       char enter;
9       string imeIPrezime_1;
10      int o1_1, o2_1, o3_1, o4_1, o5_1, o6_1, o7_1, o8_1;
11      cout << "Vnesete me i prezime na prviot ucenik: ";
12      getline( cin, imeIPrezime_1 );
13      cout << "Vnesete 8 oceni razdeleni so prazno mesto. " << endl;
14      cout << "Oceni: ";
15      cin >> o1_1 >> o2_1 >> o3_1 >> o4_1 >> o5_1 >> o6_1 >> o7_1 >> o8_1;
16      double prosek_1 = float( o1_1 + o2_1 + o3_1 + o4_1 + o5_1 + o6_1 +
17          o7_1 + o8_1 ) / 8;
18      cin.get( enter );
19
20      string imeIPrezime_2;
21      int o1_2, o2_2, o3_2, o4_2, o5_2, o6_2, o7_2, o8_2;
```

Слика 1.8.6

```
22      cout << "Vnesete me i prezime na vtoriot ucenik: ";
23      getline( cin, imeIPrezime_2 );
24      cout << "Vnesete 8 oceni razdeleni so prazno mesto. " << endl;
25      cout << "Oceni: ";
26      cin >> o1_2 >> o2_2 >> o3_2 >> o4_2 >> o5_2 >> o6_2 >> o7_2 >> o8_2;
27      double prosek_2 = float( o1_2 + o2_2 + o3_2 + o4_2 + o5_2 + o6_2 +
28          o7_2 + o8_2 ) / 8;
29
30      system( "cls" );
31      cout << left << setw( 20 ) << "  Ime i prezime" << right << setw( 25 )
32          << "  M  M  I  A  F  H  B  F" << "\tProsek" << endl;
33      cout << setw( 45 ) << right << "  a  a  n  n  i  e  i  i" << endl;
34      cout << setw( 45 ) << right << "  k  t  f  g  z  m  o  s" << endl;
35      cout << "-----------------------------------------------" << endl;
36      cout << left << setw( 21 ) << imeIPrezime_1 << "  " << o1_1 << "  "
37          << o2_1 << "  " << o3_1 << "  " << o4_1 << "  " << o5_1 << "  "
38          << o6_1 << "  " << o7_1
39          << "  " << o8_1 << "\t" << setprecision( 2 ) << prosek_1 << endl;
40      cout << left << setw( 21 ) << imeIPrezime_2 << "  " << o1_2 << "  " << o2_2
41          << "  " << o3_2 << "  " << o4_2 << "  " << o5_2 << "  " << o6_2 << "  "
42          << o7_2 << "  " << o8_2 << "\t" << setprecision( 2 ) << prosek_2 << endl;
43
44      cout << endl;
45      system( "Color 17" );
46      system( "pause" );
47      return 0;
48
49  }
```

Слика 1.8.6 продол ение

Излезот од програмата е:

```
  Ime i prezime          M  M  I  A  F  H  B  F   Prosek
                         a  a  n  n  i  e  i  i
                         k  t  f  g  z  m  o  s
-----------------------------------------------------------
Goce Delcev              5  5  4  5  4  4  5  5    4.6
Jane Sandanski           5  4  5  5  4  4  5  4    4.5

Press any key to continue . . .
```

Задачи за вежбање

1. Да се напише програма за печатење на следнава табела:

```
        Naslov                Reziser              Zanr                Godina
-------------------------------------------------------------------------------

Bal-kan-kan                 Darko Mitrevski      drama                 2005
Pred dozdot                 Milco Mancevski      drama                 1994
Najdolgiot pat              Branko Gapo          istoriski             1976
Crno seme                   Kiril Cenevski       drama                 1971
Volcja nokj                 Franc Stiglic        voen                  1955
```

Да се декларираат 5 стринг променливи за наслови на филмовите, 5 за режисерите и 3 за жанрот, како и 5 целобројни променливи за годините. Податоците да се внесат преку тастатурата во форма:

```
Film 1:
 Naslov: Pred dozdot
 Reziser: Milco Mancevski
 Zanr: drama
 Godina: 1994
```

Прашања за проверка на знаењето

1. Како е организирано читањето и печатењето податоци во C++?
2. Која библиотека е задолжително да се вклучи во секоја програма за да може да се читаат и да се печатат податоци?
3. Кои се операторите за читање и за печатење податоци?
4. Кои се наредбите за влез (внесување податоци преку тастатурата) и за излез (печатење податоци на екранот)?
5. Што е преполнување (англ. overflow)?
6. Која библиотека од C++ ги содржи функциите за форматирано печатење?
7. Со кој манипулатор се поставува ширината на полето во кое се печати некој податок?
8. Со кој манипулатор се поставува наслонувањето на податокот (лево или десно) во полето за печатење?
9. Со кој манупилатор се поставува прецизноста на податокот (бројот на децимални места) кој ќе се печати?
10. Со кој манипулатор се печатат логичките константи true и false?
11. Кој е операторот за вметнување (инсертирање) при печатење, а кој е операторот за издвојување (екстракција) при читање?
12. Напишете ја наредбата за читање еден знак.
13. Напишете ја наредбата за читање цела линија.
14. Со која наредба се чита само еден збор, а со која цела реченица?

Термини

- Сите програми кои може да ги извршува компјутерот, со едно име се наречени **софтвер.**
- Софтверот се дели на **системски програми** и **апликативни програми**.
- **Системски програми** се оние кои управуваат со работата на компјутерот.
- **Апликативни програми** или **кориснички програми** се оние кои извршуваат одредени работи за корисниците.
- **Програмски јазик** е вештачки јазик кој служи за комуникација меѓу човекот и компјутерот или за комуникација помеѓу два компјутери.
- **Алгоритам** е постапка од конечен број строго дефинирани активности и точно зададен редослед на нивното извршување.
- **Алгоритам** претставува постапка која се состои од конечно множество точно дефинирани активности, применети врз влезните податоци по строго пропишан редослед, со кои се доаѓа до излезни резултати.
- **Алгоритамски чекор** е едно дејство од алгоритамот.
- **Општиот** алгоритам се состои од поопшти чекори.
- **Деталниот** алгоритам се состои од подетални чекори кои може да се запишат со една наредба.
- **Псевдојазик** е јазик за текстуално запишување на алгоритмите.
- Графичкото претставување на алгоритмите се врши со т.н. **блок-дијаграми**.
- **Кодирање** е „изразување“, т. е. запишување на алгоритмите со наредбите од некој програмски јазик.
- **Изворна програма** претставува кодиран алгоритам со наредби од некој програмски јазик.
- **Извршна програма** е програма добиена од изворната програма со преведување, која може процесорот да ја изврши.
- **Преведувач** е системска програма која ја преведува изворната програма во извршна програма.
- **Интерпретатор** е системска програма која ја интерпретира и извршува изворната програма.
- **Алгоритамски контролни структури** се специјални структури со кои се контролира извршувањето на алгоритмите.
- **Структурирано програмирање** е програмирање во кое се користат техниките: програмирање одгоре надолу и модуларно програмирање.
- Со техниката **програмирање одгоре надолу** се разделува (расчленува) задачата на помали и поедноставни задачи т.н. **подзадачи.**
- Техниката **модуларно програмирање** се состои во програмирање на независни целини од алгоритамот како посебни модули, кои потоа се спојуваат секвенцијално во една целина.

- **Машински јазик** е јазик во кој машинските инструкции и податоците се изразуваат со нули и единици.
- **Симболички јазици** се јазици во кои машинските инструкции и податоците се изразуваат со симболи (мнемоници).
- **Асемблери** се преведувачи кои програмата од симболички јазик ја преведуваат во програма на машински јазик.
- **Виши програмски јазици** се јазиците блиски на природните јазици, во кои денес (најчесто) се програмира.
- Вишите програмски јазици се **машински независни јазици** бидејќи не зависат од машината на која ќе се извршуваат програмите (со претходно преведување).
- Вишите програмски јазици се **проблемски ориентирани јазици** бидејќи се наменети за проблеми од посебни области (економија, техника, вештачка интелигенција итн.).
- **Граматика** на програмски јазик е множество правила за градење зборови и наредби.
- **Синтакса** е множество правила за градење на наредби во програмските јазици.
- **Семантика** е значењето на некоја наредба, т. е. дејствата што таа ги предизвикува при извршувањето.
- **Резервирани зборови** се посебни зборови резервирани само за програмскиот јазик.
- Некои од резервираните зборови се **клучни зборови** бидејќи имаат посебно значење за програмскиот јазик.
- **Идентификатори** се зборови кои ги формираат програмерите по одредени правила и мора да се различни од резервираните зборови во програмскиот јазик.
- **Преведувач (компајлер)** е специјална програма за преведување на изворната програма во машинска програма.
- **Интерпретатори** се специјални програми кои директно ја извршуваат (интерпретираат) изворната програма.
- **Скриптни јазици** се програмски јазици за интеграција и комуникација со други програмски јазици.
- **Императивни програмски јазици** се оние во кои за обработка на податоците се користат наредби.
- **Декларативни програмски јзаици** се оние во кои се опишува како да се дојде до резултатот.
- **Процедурални програмски јазици** се императивни јазици во кои се користат **процедури** (функции) (група од наредби изразени како една целина).
- **Објектно ориентирани јазици** се императивни јазици во кои податоците се изразени како атрибути на објекти, а однесувањето на објектите се изразува со функции.

- **Функциски јазици** се декларативни јазици во кои се користат изрази и функции.
- **Логички јазици** се декларативни јазици во кои при пресметувањата, се користат релации, факти, заклучоци и методи за заклучување.
- **Unified Modeling Languge – UML** е посебен графички јазик за изразување на разни процеси (тек на алгоритам, преминување од една во друга состојба и сл.).
- **Интегрирана развојна околина** е околина со апликации кои се користат при програмирање.
- **Microsoft Visual Studio** е интегрирана развојна околина за пишување, уредување и извршување (тестирање) програми.
- **Code::Blocks** е интегрирана развојна околина за пишување, уредување и извршување (тестирање) програми.
- **Текстуален уредувач – едитор** е апликација за пишување и уредување програми.
- **Дибагер** е апликација за барање на логички грешки во програмата.
- **Поврзувач** е апликација за поврзување на повеќе датотеки (програми) во единствена датотека.
- **Апликација** во C++ е програма која ја содржи главната функција main().
- **<iostream>** е библиотека која содржи функции за влезни и за излезни операции во програмите.
- **namespace** е директива за формирање на именски простор (опсег) во меморијата за имиња на променливи, функции и други идентификатори.
- **Функција** е програма која по извршувањето дава резултати.
- **Назабување** претставува вовлекување на наредбите (најчесто за една табулаторна позиција) заради поголема прегледност на програмите.
- **Главна функција** е онаа со која започнува извршувањето на секоја апликација.
- **Коментар** е кој било текст ставен по две коси црти во линијата или помеѓу знаците /* на почетокот и */ на крајот.
- **cout** е наредба за печатење.
- **cin** е наредба за читање.
- **<<** е оператор за печатење.
- **>>** е оператор за читање.
- **endl** е наредба за преминување во нова линија.
- **;** е знак за крај на наредбата, т. е. разделувач на наредбите.
- **return 0;** е наредба за коректно завршување на апликацијата.
- **Редоследна структура** или **секвенца** е структура во која наредбите се извршуваат во редослед како што се наредени.

- **Излезни (ескејп) секвенци** се специјални секвенци изразени со комбинација на спротивна коса црта и некоја буква или знак. На пример, \n е ескејп секвенца за преминување во нов ред.
- **Стринг** е низа од знаци ставени во наводници, кои се третираат како посебен тип на податок.
- **Исказ** (најчесто) претставува математички израз кој има вредност. Често, наредбите во програмските јазици се нарекуваат искази.
- **Величина** е мерка за некоја карактеристика на даден објект или појава.
- **Податоци** се вредности со кои се изразуваат величините.
- **Литерали** се фиксни вредности кои не се менуваат.
- **Именувани константи** или само **константи** се идентификатори (имиња на величини) на кои може само еднаш да им се зададе вредност и таа не се менува при извршување на програмата.
- **Променливи** се идентификатори (имиња на величини) на кои може да им се задаваат различни вредности во програмата.
- **Тип** на податок претставува конечно множество податоци над кое е дефинирано конечно множество операции.
- **Прости** или **неструктурирани типови на податоци** се оние кои не може да се делат на делови.
- **Сложените** или **структурираните типови на податоци** се состојат од повеќе прости типови.
- **Адресните типови на податоци** се оние чија вредност е адреса на некоја мемориска локација.
- **Прости типови на податоци** или **вградени типови на податоци** наречени и **основни типови на податоци** се целобројниот, реалниот, знаковниот и логичкиот тип.
- **Спецификатори на типот** се имиња на типовите.
- Податоци **unsigned** се оние кои може да имаат само ненегативни вредности.
- **Стринг** е низа од знаци ставени во наводници.
- **Спојување** е операција за поврзување на два стринга еден по друг.
- **Декларација на променлива** е задавање на типот и на името на променливата.
- **Декларација и иницијализација на променлива** е задавање на типот и на името и доделување почетна вредност на променливата.
- Д**ефинирање променлива** е доделување вредност при декларација и иницијализација или доделување вредност со наредба за доделување на веќе декларирана променлива.
- **Целоброен тип на податок** е податок со вредност од множеството цели броеви Z = {... –2, –1, 0, 1, 2... }, а зависно од подмножеството, типовите на целобројните податоци се декларираат со зборовите: **short**, **int**, **long, long long, unsigned short, unsigned int, unsigned long** и **unsigned long long**.

- **Скратена форма на оператори** (или **сложени оператори**) се изразува со аритметичкиот оператор и операторот за доделување =. На пример, +=, –=, *= итн.
- **Реален тип на податок** е оној податок кој добива вредности од множеството реални броеви (се мисли на децималните броеви), а се декларира со зборовите: **float, double** и **long double**.
- Претставување децимален број со **неподвижна точка** е кога децималниот број се запишува со децимална точка на фиксно место, која точно ги дели целиот и децималниот дел на бројот.
- Претставување децимален број со **подвижна точка** е кога децималниот број се запишува со децимална точка на произволно место, при што се користи експонент за поместените места.
- **Имплицитна** (автоматска) **конверзија** или **типска принуда** е доделување вредност на променлива од несоодветен тип.
- **Експлицитна типска конверзија** или **кастирање на типот** е наведување на типот во кој сакаме да се конвертира променливата или изразот.
- **Оператор за инкрементирање** е ++.
- **Оператор за декрементирање** е – –.
- **Знаковен тип на податок** може да има вредност кој било знак или која било целобројна вредност и се декларира со зборот **char**.
- **Логички тип** на **податок** може да има вредност или true или false, а се декларира со зборот **bool**.
- **Логичките оператори** или **условни оператори** се: &&, || и !.
- **Булови функции** се: NOT, AND и OR.
- **Логички израз** е израз чија вредност може да биде true или false.
- **Релациски оператори** се: <, <=, >, >=, ==, !=.
- **Оператори со битови** се: &, |, ^, ~, << и >>.
- **Наброив тип** на **податок** може да има вредност од однапред дефинирано множество на константи (литерални вредности), кои мора да бидат идентификатори, а не броеви. Се декларира со зборот **enum**.
- **Текстуален податок** е низа од знаци ставени во наводници.
- **Преименување** на типот на податок претставува задавање на друго име (синоним) за типот.
- **Автоматско одредување на типот** се врши со зборот auto, при што типот на променливата го одредува преведувачот, според типот на изразот кој се доделува на променливата.
- **Хедер-датотеки** се датотеки од ***стандардната библиотека на C*** кои се вклучуваат со директивата #include и се стават во аглести загради <>.
- **Преполнување** настанува кога вредноста на променливата е надвор од опсегот на типот од кој е декларирана.

- **Операторот за вметнување** (инсертирање) << е поврзан со стандардниот излезен уред на компјутерот, најчесто мониторот.
- **Операторот за издвојување** (екстракција) >> е поврзан со стандардниот влезен уред на компјутерот, најчесто тастатурата.
- З**нак за нова линија** е знакот кој се генерира со притискање на копчето Enter.
- **Излезен поток** на податоци претставува низа од знаци од податоците што се печатат.
- **Влезен поток** на податоци претставува низа од знаци од податоците што се читаат.
- **<string>** е библиотека која содржи функции за работа со стрингови.
- **<iomanip>** е библиотека која содржи манипулатори за форматирано печатење.
- **<ostream>** е библиотека која содржи наредби за излезен поток.
- **<istream>** е библиотека која содржи наредби за влезен поток

Резиме

- Сите програми кои може да ги извршува компјутерот, со едно име се наречени софтвер.
- Работата на компјутерот е контролирана со т.н. системски програми.
- За да се изврши каква било програма со помош на компјутер, мора да постои апликатива програма која компјутерот може да ја изврши.
- За да може да се изведуваат какви било активности, треба да се познава точниот број на (елементарните) активности што треба да се извршат и нивниот редослед на извршување. Листата на елементарни активности (чекори) запишана според редоследот на извршување претставува алгоритам.
- При извршување на алгоритам, се внесуваат влезни податоци, а се добиваат излезни резултати. Понекогаш, излезните резултати не се добиваат директно, туку преку меѓурезултати.
- Зависно од алгоритамските чекори, алгоритамот може да биде општ или детален.
- Алгоритмите може да се запишат текстуално со псевдојазик, графички со блок-дијаграм или со наредбите на некој програмски јазик. Графичкото претставување може да биде со стандардни или со структурирани блок-дијаграми.
- Процесот на запишување на алгоритамот со наредби од некој програмски јазик се нарекува кодирање, а добиентиот текст се нарекува изворна програма.
- Изворна програма напишана во некој програмски јазик може да се преведе со преведувач во извршна програма, која процесорот може да ја изврши или истовремено да се преведува (интерпретира) и извршува со интерпретатор.
- За полесна контрола на текот на извршување на алгоритмите, тие се запишуваат со алгоритамски контролни структури. Според нив, таквото програмирање го добило името структурирано програмирање.
- Структурирано програмирање е начин на програмирање во кој се користат методите: програмирање одгоре надолу (расчленување на задачата на поедноставни задачи) и модуларно програмирање (изработка на посебни програми за независни целини).
- Решавањето на задача со помош на компјутер се состои од неколку фази: поставување на задачата, дефинирање алгоритам, пишување програма (програмирање) и тестирање на програмата.
- При поставување на задачата, треба точно да се дефинираат и да се прецизираат условите под кои таа ќе се решава.
- При дефинирање на алгоритам, претходно треба да се разбере задачата, да се изврши анализа на влезните податоци што што треба да се обработат, дали се потребни формули и/или методи и какви резултати треба да се добијат.

Потоа, се пишува алгоритам кој може да се испрограмира и програмата да се изврши на компјутер.

- По изготвување на програмата, таа треба да се тестира дали работи исправно (дали се добиваат точни резултати) за различни влезни податоци, за кои се познати резултатите.
- Програмските јазици се вештачки јазици кои се делат на: виши програмски јазици (во кои се пишуваат програмите), симболички јазици (во кои може, но не мора да се преведе програмата) и машински јазик (во кој се преведува програмата за да може компјутерот да ја изврши).
- Во машинскиот јазик, и инструкциите и податоците се запишуваат со битови, т. е. со цифри 0 и 1.
- Во симболичките јазици, и за инструкциите и за податоците се користат симболи. Симболичките јазици се машински ориентирани бидејќи секое семејство процесори имаат сопствен симболички јазик.
- Секој виш програмски јазик има: граматика, речник од резервирани (и клучни) зборови, правила за формирање на идентификатори (константи и променливи), синтаксички правила (синтакса) за конструкција на наредбите и семантички правила за проверка на значењето на наредбите.
- При пишување програма (програмирање), програмерите можат да формираат зборови (идентификатори) со посебни правила, со кои ќе ги именуваат податоците што се обработуваат. Идентификаторите не смеат да бидат резервирани зборови.
- Ако изворната програма се преведува со преведувач (компајлер), тогаш се прави извршна датотека на програмата во машинска форма. Ако изворната програма се интерпретира со интерпретатор, тогаш не се прави извршна датотека, туку интерпретаторот ги интерпретира и извршува наредбите.
- Скриптните јазици служат за комуникација со други програмски јазици. За нив не е потребен преведувач, туку интерпретатор.
- Изворните програми имаат наставка по името која е кратенка од јазикот во кој се напишани. На пример: .cpp, .java, .pas, .for итн. Извршните програми имаат наставка .exe, .com, .bat, .bin, .dll и др.
- Има повеќе генерации програмски јазици.
- Според начинот на обработка на податоците, програмските јазици се делат на: императивни (процедурални и објектно ориентирани) и декларативни (функциски и логички).
- Во објектно ориентираното програмирање се користат објекти кои имаат свои атрибути (податоци) и однесување или состојба (изразени со функции).
- За графичко претставување на алгоритмите и процесите во некој алгоритам, се користи јазикот Unified Modeling Languge – UML.
- Денес постојат разни интегрирани развојни околини за програмирање во C++, а најпознати се: Microsoft Visual Studio, Code::Blocks, NetBeans и други.

- Секоја интегрирана развојна околина има: уредувач на текст, преведувач, поврзувач и дибагер.
- Јазикот C++ е јазик за објектно ориентирано програмирање, дизајниран од Bjarne Stroustrup (Бјарне Строуструп) во 1983 година, како надградба на јазикот „C со класи“ од 1980 година, кој, пак, е надградба на јазикот C од 1972 година дизајниран од Dennis Ritchie (Денис Ричи).
- Програма (апликација) напишана во C++ мора да има една главна функција (main()) од која започнува извршувањето и други функции кои се повикуваат од главната функција.
- C++ е меѓу првите објектно ориентирани програмски јазици.
- За извршување на влезно-излезните операции, на почетокот од програмата мора да се вклучи библиотеката <iostream>.
- За поголема прегледност на програмата, се стават празни линии и коментари, кои не се преведуваат, а се врши и назабување на програмата.
- Секоја наредба во C++ завршува со точка и запирка (;).
- Програмите во C++ може да се пишуваат со кој било уредувач на текст.
- Програмата во C++ се состои од функции. Главната функција се нарекува main().
- Телото на секоја функција се става во големи загради.
- Секоја апликација мора да има само една главна функција, со која започнува извршувањето на апликацијата.
- Наредба за печатење во C++ е cout, која се користи со операторот за печатење <<.
- Наредба за читање во C++ е cin, која се користи со операторот за читање >>.
- Главната функција во C++ враќа резултат 0 ако е коректно извршена.
- Следната наредба по наредбата за печатење која на крајот го има манипулаторот endl печати во нова линија (нов ред).
- За печатење на специјални знаци, како “...“, '... ', \ итн. се користат излезни (ескејп) секвенци.
- Јазикот C++ има граматика, синтакса, семантика, резервирани зборови, идентификатори (зборови формирани од програмерите) и други елементи на еден програмски јазик.
- При преведување на програмата, со синтаксичките правила се проверува исправното запишување на секоја наредба.
- Секоја наредба во програмата мора да има точно дефинирано значење и да предизвика точно дефинирани дејства. Тоа се проверува со семантичките правила на јазикот.
- Со податоци се изразуваат вредности на величините, а величините се мерки за карактеристики на објекти или на појави. Податоците може да бидат константи (имаат една непроменлива вредност) или променливи (вредноста може да се менува).

- При декларирање на константи во C++, пред нивниот тип се става зборот const.
- Секој податок има: име, тип и вредност.
- Имињата (идентификаторите) на податоците во C++ може да бидат: резервирани зборови (во кои спаѓаат и клучните зборови) или кориснички дефинирани имиња (идентификатори), кои се формираат по одредени правила. Резервираните зборови не може да се користат како идентификатори.
- Типовите на податоци во C++ може да бидат прости (неструктурирани), сложени (структурирани – кои се состојат од прости типови) и адресни (чија вредност е адреса на мемориска локација).
- Прости типови се: интегрален (целобројен, знаковен и логички), реален и набројив.
- Сложени типови се: низа, структура, унија и класа.
- Константите и променливите се декларираат со наведување на типот и името. Ако во декларацијата се зададе вредност на константата или на променливата, тогаш се вршат декларација и иницијализација.
- За да се разликуваат константите од променливите, се практикува тие да се пишуваат со сите големи букви, а зборовите во името на константите да се разделуваат со долна црта.
- Константите се декларираат со зборот const по кој се наведуваат типот и името, а потоа, мора да се додели вредност на константата со операторот за доделување =.
- Пред да се користи во програмата, променливата мора да е дефинирана со задавање вредност, било со иницијализација, било со наредба за доделување.
- Кога на некоја променлива ѝ се доделува вредност поголема или помала од опсегот на нејзиниот тип, доаѓа до преполнување или потполнување и се јавува грешка.
- Целобројниот тип и знаковниот тип може да бидат и неозначени.
- Спецификатори на типот на податок се:
  - за целобројен тип: short, int, long и long long.
  - за знаковен тип: char.
  - за набројив тип: enum.
  - за структура: struct.
  - за класа: class.
  - за референца: reference.
  - за реален тип: float, double и long double.
  - за логички тип: bool.
  - за низа: array.
  - за унија: union.
  - за покажувач: pointer.
- Целобројниот тип на податоци се вредности од множеството цели броеви {... –2, –1, 0, 1, 2...}.
- Скратени форми на аритметичките оператори: +, –, * и / за целобројни и за реални податоци се: +=, –=, *=, /=. Операторот %= е скратена форма на

операторот % за целоброен тип на податоци. Овие оператори се нарекуваат сложени оператори.

- Реалниот тип на податоци се вредности од множеството реални (децимални) броеви {... –2.0..., –2.001..., –1.0..., 0.0..., 1.0..., 2.0...}.
- При декларирање на променливи од типот float, треба да се стави наставката (суфиксот) f или F. Во спротивно, тие ќе се третираат како променливи од типот double.
- Децималните податоци во програмските јазици се претставуваат во децимален формат (f-формат, F-формат) (начин познат како неподвижна точка) и во експоненцијален формат (e-формат, E-формат) (начин познат како подвижна точка).
- Буквата f доаѓа од зборот „float“, а буквата е доаѓа од зборот „exponent“.
- Децималните броеви во f-формат (или F-формат) се запишуваат со децимална точка на фиксно место.
- Кога децималните броеви се печатат во e-формат (или E-формат), се запишува *само една цифра* лево од децималната точка, а буквата е или E се пишува пред експонентот.
- Доделување вредност на променлива од несоодветен тип се врши со конверзија, а може да биде имплицитно (автоматски) или експлицитно (со кастирање). Во вториот случај, се наведува типот во кој треба да се конвертира вредност од несоодветен тип, а во првиот случај, не се наведува.
- За зголемување/намалување вредност на целобројна или на знаковна променлива за 1, се користи оператор за инкрементирање (++) или оператор за декрементирање (– –).
- Податоците од знаковен тип може да имаат вредност кој било знак или која било целобројна вредност. Тие се стават во полунаводници.
- Логичкиот тип на податоци може да имаат само вредност на специјалните константи true или false.
- При пресметување на логичките изрази, логичките константи true и false имаат вредност 1 и 0.
- За операции со логички тип на податоци, се користат логичките (условни) оператори: &&, || и !. Тие се оператори за Буловите функции AND (***И***), OR (***ИЛИ***) и NOT (***НЕ***).
- Релациски оператори се: <, <=, >, >=, == (еднакво) и != (различно).
- Оператори со битови се: & (логичко *И*), | (логичко *ИЛИ*), ^ (исклучиво *ИЛИ*), ~ (комплемент), << (поместување налево за 1 место) и >> (поместување надесно за 1 место).
- Наброив тип на податок се декларира со зборот enum, а константите се стават во листа во големи загради и се пишуваат со големи букви. По излезниот дел од големите загради, не се става знакот точка и запирка (;).
- Наброив тип на податоци се дефинира пред функцијата main();

- Стрингови или текстуални податоци се податоци составени од низа знаци ставени во наводници.
- За работа со стрингови, се користат функции од библиотеката <string>, која мора да се вклучи во програмата со директивата #include.
- Честопати, заради читливост на програмата и полесно снаоѓање во неа, за имињата на вградените типови на податоци пожелно е да се користат синоними (други имиња) кои ќе нè потсетуваат на нешто, со клучниот збор typedef.
- Кога сакаме типот на променливата да се одреди според типот на изразот кој ќе се доделува, таа се декларира со зборот auto.
- Читањето податоци во C++ е организирано како влезен поток од знаци, додека печатењето е организирано како излезен поток од знаци.
- Наредбата за читање од стандарден влез е cin, а наредбата за печатење на стандарден излез е cout. cin и cout се променливи од библиотеката <iostream>. cin е од типот istream кој е поврзан со влезниот поток на податоци, а cout е од типот ostream кој е поврзан со излезниот поток на податоци.
- Програмските линии кои почнуваат со директивата `#include` преведувачот не ги преведува, туку тие се обработуваат со посебна програма наречена претпроцесор**.**
- Датотеките кои се вклучени во програмата со директивата #include се стават во аглести загради, со кои се информира претпроцесорот нив да ги бара во т.н. директориум include од *стандардната библиотека на C++*.
- За форматирање на податоците, при печатење се користат манипулатори. Некои од нив се дефинирани во библиотеката <iostream>, а некои се дефинирани во библиотеката <iomanip>, која (како и <iostream>) треба да се вклучи на почетокот од програмата.
- Со притискање на копчето Enter, се генерира знак за нова линија и се преминува во нова линија. При читање на влезни податоци, се чита додека не се најде знакот за нова линија.
- *Стандардната библиотека на C++* содржи повеќе датотеки, наречени хедер-датотеки кои содржат групи од слични функции. Најчесто користени хедер-датотеки се: <iostream>, <iomanip>, <cmath>, <string>, <cctype> и други.

МОДУЛ

КОНТРОЛА НА ТЕК И ФУНКЦИИ ВО C++

2

Во оваа глава ќе се запознаете со следното:

- Алгоритамски контролни структури.
- Користење на редоследната алгоритамска контролна структура.
- Алгоритамска контролна структура за избор од две можности **ако–тогаш–инаку** и контролна наредба за избор од две можности if-else.
- Вгнездување на контролната наредба за избор од две можности if-else.
- Алгоритамска контролна структура за избор од повеќе можности **случај** и контролна наредба за избор од повеќе можности switch.
- Користење на условниот оператор ?:.
- Алгоритамски контролни структури за повторување **додека–извршувај**, **извршувај–додека** и **за–до–чекор** и контролни наредби за повторување while, do-while и for.
- Користење на операторите за инкрементирање (++) и за декрементирање (––).
- Алгоритамски контролни структури за скок (**продолжи**, **прекин**, **излез**, **скок**) и контролни наредби за скок (continue, break, exit(), goto).
- Библиотечни функции во програмскиот јазик C++.
- Кориснички функции: дефинирање, декларирање и повикување.
- Декларирање и користење на глобалните и на локалните променливи.
- Вградени функции.

Клучни зборови

:: оператор за разрешување на подрачјето на видливост.	Корисничка функција
?: условен оператор	Кориснички дефинирана функција
break	Листа на аргументи
continue	Листа на параметри
do-while	Локална променлива
exit()	Корисничка функција
for	Модуларно програмирање
goto	Параметри
if	Подалгоритам
if-else	Подзадача
inline	Подрачје на видливост
switch	Подрачје на пристап
void	**почеток–крај**
while	**прекин**
ако–тогаш	Пренесување на аргумент по вредност
ако–тогаш–инаку	Пренесување на аргумент по референца
Алгоритамска контролна структура	Програмирање одгоре надолу
Алгоритамската контролна структура за избор од две можности	**продолжи**
Алгоритамски блок	Процедура
Аргумент	Процедурален подалгоритам
Аргумент пренесен по вредност	Редоследна алгоритамска контролна структура (секвенца)
Аргумент пренесен по референца	Референтна променлива
Вградена функција	Референца
Вистински аргумент	Референцирање
Влезен параметар	Стандардна библиотека
Влезен формален аргумент	**скок**
Влезно-излезен параметар	Скриена променлива
Влезно-излезен формален аргумент	**случај**
врати	Статичка променлива

Глобална променлива	Формален аргумент
додека–извршувај	Формален параметар на типот
Живот на променлива	Функција
за–до–чекор	Функција без повратна вредност
за–зголемувај–до	Функција со повратна вредност
за–намалувај–до	Функциска потпрограма
извршувај–додека	Функциски подалгоритам
излез	Функциски потпис
Излезен параметар	Функциски прототип
Излезен формален аргумент	Циклично повторување
Константен параметар	Циклус

Во потточките **1.1**, **1.2** и **1.3** од **МОДУЛ 1: Вовед во програмскиот јазик C++**, се запознавме со поимите алгоритам, алгоритамски чекори, начини на претставување на алгоритмите и структурирано програмирање. За опис на текот на алгоритмите при структурираното програмирање, се користат посебни т.н. **алгоритамски контролни структури**. Со нив не се извршуваат никакви пресметки, туку се одредува редоследот на извршување на алгоритамските чекори, т. е. се управува со редоследот на извршување на чекорите.

Во теоријата за програмирање е познато дека текот на секој алгоритам може да се контролира со користење на следниве алгоритамски контролни структури:

- **Редоследна контролна структура** или **секвенца** (англ. sequence).
- **Контролна структура за избор** или **селекција** (англ. selection).
- **Контролна структура за повторување** или **итерација** (англ. iteration).

Првата алгоритамска контролна структура е позната и под името **линиска контролна структура**, втората е позната и под името **разгранета контролна структура**, а третата е позната и под името **циклична контролна структура**.

Секоја алгоритамска контролна структура има само една влезна точка (●) и само една излезна точка (⊙), ***слика 2.1***.

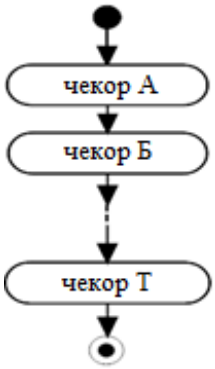
Слика 2.1

2.1 Редоследна контролна структура

Оваа контролна структура се состои од алгоритамски чекори кои се извршуваат по оној редослед по кој се наведени. Неа ја користевме во сите досегашни примери.

Графички, структурата е претставена на ***слика 2.1.1***.

Секоја редоследна контролна структура претставува посебна целина и може да се појави каде било во алгоритамот. Затоа, е потребно да се означи почетокот и крајот на структурата. Тоа се прави со зборовите **почеток** (англ. begin) и **крај** (англ. end), ***слика 2.1.2.*** Редоследната алгоритамска контролна структура се нарекува **почеток–крај**.



Слика 2.1.1

```
почеток
      чекор А;
      чекор Б;
      ...
      чекор Т;
крај
```

Слика 2.1.2

Знакот ; (точка и запирка) се користи за одделување на чекорите во редоследната контролна структура **почеток–крај**. Содржината на редоследната контролна структура е еден **блок** од чекори, а за поголема прегледност чекорите се пишуваат малку вовлечено надесно од зборовите **почеток** и **крај**. Зборовите **почеток** и **крај** се пишуваат во иста вертикала. Ова важи за сите алгоритамски контролни струк-

тури. Овој начин на пишување на алгоритмите и програмите со вовлекување (најчесто за една табулаторна позиција), се нарекува **назабување** (англ. indentation). Назабувањето се користи за поголема прегледност и за полесно распознавање на алгоритамските контролни структури и на програмите.

Алгоритамската контролна структура **почеток–крај** најчесто се користи во чекорите за задавање на почетни вредности на податоците и при пресметувања.

На пример:

почеток {Почетни вредности}
    a ← 3;
    b ← –5.74;
    c ← '@';
крај

Притоа стрелката налево ← се користи како симбол за доделување вредности на променливите.

Во сите програми што ги напишавме во **Модул 1: Вовед во програмскиот јазикот C++**, користевме алгоритамска контролна структура **почеток–крај** бидејќи наредбите ги пишувавме една по друга.

Во некои случаи, ако треба да се означи група наредби со кои се извршува еден алгоритамски чекор (на пример, чекорот: читање податоци), групата се става во големи загради {}. Заградите {} го означуваат блокот наредби кои треба да се извршат.

```
{
    naredba A;
    naredba B;
    . . .
    naredba P;
}
```

Слика 2.1.

Така, алгоритамската контролна структура **почеток–крај** од ***слика 2.1.2***, ќе се напише како на ***слика 2.1.*** .

На пример, алгоритамскиот сегмент за задавање почетни вредности на податоци, ќе се запише со следниов програмски сегмент:

```
{       // Pocetni vrednosti
        a = 3;
        b = -5.74;
        c = '@';
}
```

Напомена: По заградите {}, не се става знакот точка и запирка.

По извршувањето на следниов програмски сегмент

```
{
        a = 2;
        b = 3;
        p = a;
        a = b;
        b = p;
}
```

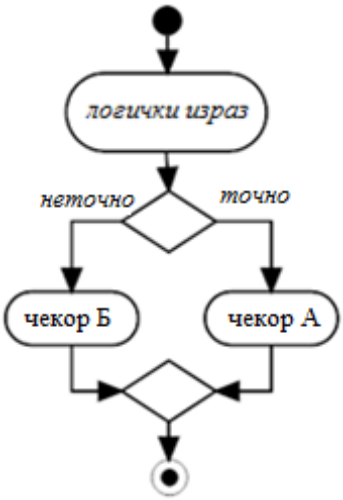
вредностите на променливите a и b ќе бидат 3 и 2.

2.2 Алгоритамски контролни структури за избор и контролни наредби за избор

Алгоритамска контролна структура за избор од две можности ако–тогаш–инаку

Алгоритамската контролна структура за избор од две можности се користи за избор на една од двете можни насоки на продолжување на дејството во алгоритамот, во зависност од некој услов. Условот е некој логички израз. Логичкиот израз може да има само две вредности: *точно* и *неточно*. Тие се нарекуваат *логички вредности* и затоа, изразот се нарекува *логички израз*.

Контролната структура за избор од две можности графички е претставена на ***слика 2.2.1***.



Слика 2.2.1

Ако вредноста на *логичкиот израз* е *точно*, **тогаш** се извршува чекорот А, **инаку** (ако вредноста е *неточно*) се извршува чекорот Б. Затоа, оваа контролна структура се нарекува **ако–тогаш–инаку** (англ. if-then-else).

Текстуалниот запис на структурата во псевдојазик е даден на ***слика 2.2.2***.

```
ако логички израз
        тогаш
                чекор А;
        инаку
                чекор Б;
крај_ако {логички израз}
```

Слика 2.2.2

Структурата почнува со **ако**, а завршува со **крај_ако** {*логички израз*}.

На пример, на ***слика 2.2.*** е користена оваа структура за наоѓање на поголемиот од броевите a и b.

```
ако a > b
        тогаш
                pogolem ← a
        инаку
                pogolem ← b;
крај_ако {a > b}
```

Слика 2.2.

Кажавме дека чекорите во еден алгоритам може да бидат поопшти и да се состојат од повеќе други чекори или дури и од цели контролни структури. Ако чекорот А и чекорот Б од ***слика 2.2.2*** се состојат од други чекори, тогаш тие претставуваат еден **алгоритамски блок** и најчесто се означуваат со зборовите **почеток** и **крај**, како на ***слика 2.2.4***. Значи, чекорите А1, А2... Аn и Б1, Б2... Бm се алгоритамски блокови во можноста **тогаш**, односно во можноста **инаку**.

```
ако логички израз
        тогаш
                почеток
                        чекор А1;
                        чекор А2;
                        ...
                        чекор Аn;
                крај
        инаку
                почеток
                        чекор Б1;
                        чекор Б2;
                        ...
                        чекор Бm;
                крај
крај_ако {логички израз}
```

Слика 2.2.4

На пример, за наоѓање на поголемиот од два дадени броја a и b, со користење на оваа контролна структура, ќе запишеме:

```
pogolem ← b;
ако a > b
        тогаш
                pogolem ← a;
крај_ако {a > b}
```

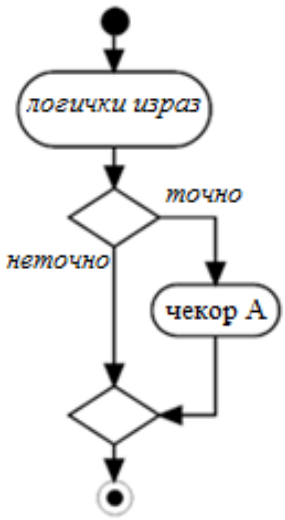
Контролната структура за избор од две можности може да се користи и кога една од можностите на логичкиот израз не содржи извршни чекори. Во тој случај, се продолжува на следниот чекор по контролната структура. Текстуалниот запис е даден на ***слика 2.2.5.***

```
ако логички израз
        тогаш
                чекор А;
крај_ако {логички израз}
```

Слика 2.2.5

Ваквата контролна структура се нарекува **ако–тогаш** (англ. if-then).

Графичкото ретставување на контролната структура **ако–тогаш** е дадено на ***слика 2.2.6***.



Слика 2.2.6

Контролна наредба за избор од две можности if

Со контролната наредбата за избор if, се реализира алгоритамската контролна структура за избор од две можности **ако–тогаш**, при што едната можност нема наредби, ***слика 2.2.5*** и ***слика 2.2.6***. Логичкиот израз е некој *услов* за извршување на naredba A или за неизвршување, ***слика 2.2.*** .

```
if(uslov)
        naredba A;
```
Слика 2.2.

При извршување на контролната наредба за избор if, прво се испитува *uslov* и ако е исполнет (има вредност true), се извршува naredba A, а ако не е исполнет (има вредност false), не се извршува naredba A, туку извршувањето на програмата продолжува со следната наредба.

Ако треба да се извршат повеќе наредби, тие се стават во блок со големи загради, ***слика 2.2.8***.

```
if(uslov) {
        naredba 1;
        ...
        naredba m;
}
```
Слика 2.2.8

Ако *uslov* има вредност true, тогаш се извршуваат наредбите од блокот {naredba 1... naredba m}, инаку наредбите од блокот не се извршуваат.

uslov може да биде само логички израз, кој може да има вредност true или false.

Логичките изрази во C++ се градат со релациски и со логички оператори[1].

Примери за логички изрази се:

```
(a > b) && (a >= c)
((alfa < 90) && (beta > 0)) || (alfa – beta > 0)
!(x <= y)                    исто со    x > y
!((a > b) && (a >= c))       исто со    (a <= b) || (a < c)
!((x != y) || (x > y))       исто со    (x == y) && (x <= y)
```

Следниве три примери ја илустрираат контролната наредба за избор **if**:

а)
```
int x;
if(x < 0)
        xaps = –x;
```

б)
```
int x;
if (x != 0)
     x = 0;
```

в)
```
int x, y, pom;
if (true) {
     pom = x;
     x = y;
     y = pom;
}
```

[1] За нив зборувавме во **Логички тип на податоци** во потточката **1.7. Типови на податоци.**

Примери

Пример 2.2.1

Да се напише програмски сегмент за утврдување дали внесениот broj е позитивен.

```
int broj;
cout << "Vnesete broj: ";
cin >> broj;
if(broj > 0) {
	cout << "Vneseniot broj e pozitiven." << endl;
}
```

Пример 2.2.2

Да се напише програмски сегмент за пресметување на апсолутната вредност на број.

```
int broj;
cout << "Vnesete cel broj: ";
cin >> broj;
cout << "Apsolutnata vrednost na brojot " << broj;
if (broj < 0) {
	broj = -broj;
}
cout << " e " broj << endl;
```

Контролна наредба за избор од две можности if-else

Со контролната наредба за избор if-else, се реализира алгоритамската контролна структура за избор од две можности **ако–тогаш–инаку**, ***слика* 2.2.1** и ***слика* 2.2.2**.

При извршување на контролната наредба за избор if-else (***слика 2.2.9***), прво се испитува *uslov* и ако е исполнет (има вредност true), се извршува naredba A, а ако не е исполнет (има вредност false), се извршува naredba B.

Ако треба да се извршат повеќе наредби кога е условот исполнет или кога тој не е исполнет, тие наредби се стават во блок со големи загради, ***слика 2.2.10***.

Ако *услов* има вредност true, тогаш се извршуваат наредбите од блокот {naredba A1... naredba Ap}, а ако *услов* има вредност false, се извршуваат наредбите од блокот {naredba B1... naredba Bq}.

```
if(uslov)
	naredba A;
else
	naredba B;
```

Слика 2.2.9

```
if(uslov) {
	naredba A1;
	...
	naredba Ap;
}
else {
	naredba B1;
	...
	naredba Bq;
}
```

Слика 2.2.10

Примери

Пример 2.2.3

Ако при точен одговор на некое прашање треба да се одговори со една порака, а при неточен со друга, тогаш тоа можеме да го запишеме со следниов програмски сегмент:

```
char odgovor;
cout << "Vnesete odgovor D za tocno: ";
cin >> odgovor;
if (odgovor == 'D')
	cout << "Odgovorot e tocen." << endl;
else
	cout << "Odgovorot ne e tocen." << endl;
```

Пример 2.2.4

Аголот alfa е остар ако е поголем од 0^o и помал од 90^o. Тоа може да се запише со следниов програмски сегмент:

```
int alfa;
cout << "Vnesete agol vo stepeni: ";
cin >> alfa;
if ((alfa > 0) && (alfa < 90))
	cout << "Agolot e ostar." << endl;
else
	cout << "Agolot ne e ostar." << endl;
```

Пример 2.2.5

Следниот програмски сегмент е исправен (преведувачот не јавува грешка), но работи погрешно:

```
bool daNe;
cin >> daNe;
if (daNe = true) {
	cout << "Odgovorot e tocen." << endl;
	cout << "Ama segmentot e pogresen." << endl;
}
else
	cout << "Odgovorot ne e tocen." << endl;
```

Која било вредност да ја внесеме за променливата `daNe`, сегментот секогаш печати `Odgovorot e tocen.`

Грешката е во условот `daNe = true` бидејќи тоа е наредба за доделување (со операторот =), а не услов за еднаквост (со оператор ==). Резултатот од нејзиното извршување секогаш е true и затоа, секогаш се печати `Odgovorot e tocen.`

Пример 2.2.6

Честопати е потребна проверка на вредноста на именителот при делење на два броја бидејќи не е дозволено делење со 0. Следниот програмски сегмент илустрира како треба да се избегне делење со 0:

```
int n;
cout << "Vnesete cel broj: ";
cin >> n;
if (n != 0)
     cout << "Reciprocnata vrednost na " << n << "e: " << 1./ n << endl;
else
       cout << "Ne e dozvoleno delenje so 0." << endl;
```

Вежби

Вежба 2.2.1

Кои од следниве програмски сегменти даваат ист резултат?

а)

```
if(a > b)
      cout << a;
Else
      cout << b;
```

б)

```
if(a > b)
      cout << a;
```

в)

```
if(a <= b)
      cout << b;
Else
      cout << a;
```

Вежба 2.2.2

Која е разликата меѓу следниве програмски сегменти?

а)

```
if((a == b) || (a == c))
       cout << a << endl;
else
       cout << b << ' ' << c << endl;
```

б)

```
if((a != b) && (a != c))
       cout << b << ' ' << c << endl;
else
       cout << a << endl;
```

Вежба 2.2.3

Што работи следниов програмски сегмент?

```
if(a < b)
      min = a;
else
      min = b;
 cout << min << endl;
```

Решени задачи

Задача 2.2.1

Да се провери дали внесениот природен број е парен или непарен.

```
1   // Da se proveri dali vneseniot broj e
2   // paren ili neparen.
3   #include <iostream>
4   using namespace std;
5
6   int main() {
7       int broj;
8       cout << "Vnesete priroden broj: ";
9       cin >> broj;
10      if( broj % 2 == 0 )
11          cout << "Vneseniot broj " << broj << " e paren." << endl;
12      else
13          cout << "Vneseniot broj " << broj << " e neparen." << endl;
14
15      cout << endl;
16      system( "Color 17" );
17      system( "pause" );
18      return 0;
19  }
```

Слика 2.2.11

```
Vnesete priroden broj: 123
Vneseniot broj 123 e neparen.

Press any key to continue . . .
```

Задача 2.2.2

Да се провери дали внесениот број е позитивен, негативен или нула.

```
1   // Da se proveri dali vneseniot broj e
2   // pozitiven, negativen ili nula.
3   #include <iostream>
4   using namespace std;
5
6   int main() {
7       float broj;
8       cout << "Vnesete broj: ";
9       cin >> broj;
10      cout << "Vneseniot broj " << broj << " e: ";
11      if (broj < 0) {
12          cout << " negativen." << endl;
13      }
14      if (broj > 0) {
15          cout << " pozitiven." << endl;
16      }
```

Слика 2.2.12

```
17      if (broj == 0) {
18          cout << " nula." << endl;
19      }
20
21      cout << endl;
22      system("Color 17");
23      system("pause");
24      return 0;
25  }
```

Слика 2.2.12 продол ение

Еден излез од извршување на програмата е:

```
Unesete broj: -12.345
Uneseniot broj -12.345 e:  negativen.

Press any key to continue . . .
```

Задача 2.2.3

Да се провери дали внесениот број е во опсегот на соодветниот тип броеви.

Забелешка: Во поднасловот **Читање и печатење бројни податоци** од потточката **1.8 Читање и печатење податоци**, објаснивме како настанува преполнување (англ. overflow). За да не дојде до преполнување, бидејќи може да се добијат погрешни резултати, честопати е потребна проверка на некој податок дали е тој во опсегот на вредности на типот од кој е декларирана променливата.

Во следниов пример е илустрирано преполнување на цели броеви.

```
1   // Prepolnuvanje na celi broevi
2   #include <iostream>
3
4   using namespace std;
5
6   int main() {
7
8       int celBroj;
9       double na3;
10      cout << "Opsegot na tip int e: [" << INT_MIN
11          << ", " << INT_MAX << "]" << endl;
12      cout << "Vnesete cel broj pogolem od 1290: ";
13      cin >> celBroj;
14      na3 = 1.0 * celBroj * celBroj * celBroj;
15      cout << "Celiot broj " << celBroj << " na kub e: ";
16      if( ( INT_MIN <= na3 ) && ( na3 <= INT_MAX ) )
17          cout << int( na3 ) << endl;
18      else {
```

Слика 2.2.1

```
19          cout << "Dojde do prepolnuvanje." << endl;
20          cout << "Bidejkji " << celBroj << " na kub ne e "
21              << celBroj * celBroj * celBroj << endl;
22      }
23
24      cout << endl;
25      system( "Color 17" );
26      system( "pause" );
27      return 0;
28  }
```

Слика 2.2.1 продол ение

Еден излез од извршување на програмата е:

```
Opsegot na tip int e: [-2147483648, 2147483647]
Vnesete cel broj pogolem od 1290: 1291
Celiot broj 1291 na kub e: Dojde do prepolnuvanje.
Bidejkji 1291 na kub ne e -2143282125

Press any key to continue . . .
```

Задачи за вежбање

Да се напишат програми за следниве задачи:

1. Да се пресметаат квадратот и кубот на природниот број n.
2. Да се пресметаат обиколката на кружницата и плоштината на кругот со радиус r.
3. Да се прочита трицифрен природен број и да се отпечати средната цифра.
4. Да се прочитаат три броја и да се утврди дали може да бидат страни на триаголник.
5. Да се подредат три броја по големина.
6. Да се реши линеарната равенка: $ax + b = 0$, за $a \neq 0$.
7. Да се реши линеарната неравенка: $ax + b > 0$, за $a \neq 0$.
8. Да се прочита агол помал од 180^o и да се отпечати дали е остар или тап.
9. Да се провери дали внесениот датум е исправен. (Датумот да се внесе како три цели броја за ден, месец и година).
10. Да се пресмета возраста на човек како разлика на денешниот датум и датумот на раѓање. Датумите да се внесуваат како цели броеви за ден, месец и година.

Вгнездување на контролните наредби if и if-else

Контролните наредби if и if-else може да се вгнездат (англ. nested) било во изборот if било во изборот else.

Кога вгнездувањето е во изборот else, таквата наредба се нарекува if-else-if.

Да го разгледаме следниов пример.

Пример 2.2.7

Дали три цели броја a, b и c може да бидат страни на правоаголен триаголник?

Решение: Броевите a, b и c може да бидат страни на правоаголен триаголник ако $a^2 + b^2 = c^2$ или ако $a^2 + c^2 = b^2$ или ако $b^2 + c^2 = a^2$.

Оваа реченица можеме да ја изразиме со следниов програмски сегмент:

```
if(a * a + b * b == c * c)
       cout << "Da";
else   if(a * a + c * c == b * b)
       cout << "Da";
else   if (b * b + c * c == a * a)
       cout << "Da";
else
       cout << "Ne";
```

Вгнездувањето може да биде и во изборот if.

Да го разгледаме следниов пример.

Пример 2.2.8

Да се најде најголемиот од три дадени броја a, b и c.

Решение:

Ако a > b и a > c, тогаш е најголем бројот a.

Ако b > a и b > c, тогаш е најголем бројот b.

Ако c > a и c > b, тогаш е најголем бројот c.

Програмскиот сегмент ќе биде:

```
if(a > b)
       if(a > c)
              cout << "Najgolem broj e " << a;
if(b > a)
       if(b > c)
              cout << "Najgolem broj e " << b;
if(c > a)
       if(c > b)
              cout << "Najgolem broj e " << c;
```

Напомена: Не е дозволено вгнездената програмска наредба за избор да биде и во изборот **if** и во изборот **else**.

При вгнездување на контролните наредби за избор if, if-else и if-else-if, треба да се внимава дека преведувачот во C++ секое else го поврзува со претходното if. На пример, во следниот програмски сегмент, за a < b, нема ништо да се отпечати бидејќи else е врзано со второто if, а не со првото.

```
if(a > b)
	if(a > c)
		cout << "Najgolem broj e " << a;
	else
		cout << "Najgolem broj ne e " << a;
```

Ако сакаме за a < b да се изврши наредбата

```
cout << "Najgolem broj ne e " << a;
```

тогаш треба else да се поврзе со првото if со загради.

```
if(a > b) {
	if(a > c)
		cout << "Najgolem broj e " << a;
}
else
	cout << "Najgolem broj ne e " << a;
```

Ова е наречено **проблем на висечко else** (англ. dangling-else problem).

Вежби

За **Пример 2.2.7** и **Пример 2.2.8** да се напишат посебни програми во C++.

Решени задачи

Задача 2.2.4

Да се отпечати општиот успех на ученик ако е познат просекот на оценките од предметите, и тоа:

Просек	Општ успех
1 до 1.5	Недоволен
1.6 до 2.5	Доволен
2.6 до 3.5	Добар
3.6 до 4.5	Многу добар
4.6 до 5.0	Одличен

Програмата во C++ ќе биде како на ***слика 2.2.14.***

```
1   // Da se ispecati opstiot uspeh na ucenik ako e poznata prosecnata ocena.
2   #include <iostream>
3   using namespace std;
4
5   int main() {
6       float prosek;
7       cout << "Vnesete prosecna ocen: ";
8       cin >> prosek;
9       cout << "Opst uspeh za prosecna ocena " << prosek << " e ";
10      if (prosek <= 1.5)
11          cout << "Nedovolen" << endl;
12      else    if (prosek <= 2.5)
13          cout << "Dovolen" << endl;
14      else    if (prosek <= 3.5)
15          cout << "Dobar" << endl;
16      else    if (prosek <= 4.5)
17          cout << "Mnogu Dobar" << endl;
18      else
19          cout << "Odlicen" << endl;
20
21      cout << endl;
22      system("Color 17");
23      system("pause");
24      return 0;
25  }
```

Слика 2.2.14

```
Vnesete prosecna ocen: 3.56
Opst uspeh za prosecna ocena 3.56 e Mnogu Dobar

Press any key to continue . . .
```

Да се потсетиме дека кога има само една наредба во if или во else, тогаш може, но не мора, да се стават големи загради.

Задача 2.2.5

Да се одреди најголемиот од три дадени броја.

Програмата во C++ е дадена на ***слика 2.2.15***.

```
1   // Da se odredi najgolemiot od tri broja.
2
3   #include <iostream>
4   using namespace std;
5
6   int main() {
7       double a, b, c;
8       cout << "Vnesete tri broja: \n";
9       cout << "a="; cin >> a;
```

Слика 2.2.15

```
10      cout << "b="; cin >> b;
11      cout << "c="; cin >> c;
12      cout << "Najgolem od trite broja e ";
13      if (a > b) {
14          if (a > c)
15              cout << a << endl;
16          else
17              cout << c << endl;
18      }
19      else {
20          if (b > c)
21              cout << b << endl;
22          else
23              cout << c << endl;
24      }
25
26      cout << endl;
27      system("Color 17");
28      system("pause");
29      return 0;
30  }
```

Слика 2.2.15 продол ение

```
Unesete tri broja:
a=1.23
b=4.56
c=3.45
Najgolem od trite broja e 4.56

Press any key to continue . . .
```

Условен оператор ?:

Условниот оператор се користи за претставување на едноставни наредби if-else. Неговата синтакса е

uslov? izraz T : izraz N;

Прво се пресметува *uslov* чија вредност може да биде true или false. Ако е true, тогаш се пресметува izraz T, а ако е false се пресметува izraz N.

На пример, програмскиот сегмент:

```
if(m > n)
        max = m;
else
        max = n;
```

може да се запише пократко

```
max = (m > n) ? m : n;
```

Со следната наредба ќе се отпечати „paren“ ако променливата broj има парна вредност или „neparen“ ако променливата broj има непарна вредност:

```
cout << "Brojot " << broj << " e "
             << ((broj % 2 == 0) ? "paren" : "neparen") << endl;
```

На пример, за broj = 23 ќе се отпечати:

```
Brojot 23 e neparen
```

Задачи за вежбање

Да се напишат програми за решавање на следните задачи со вгнездување на контролни наредби за избор:

1. Да се внесат два броја и оператор за аритметичката операција (+, –, * или /) и да се изврши операцијата.
2. Да се одреди арапскиот еквивалент на римска цифра. (Римски цифри се: I = 1, V = 5, X = 10, L = 50, C = 100, D = 500 и M = 1 000).
3. Да се определи оценката за ученик според освоените бодови ако оценувањето се врши според следнава скала на бодови: помалку од 60 – недоволен, од 60 до 69 – доволен, од 70 до 79 – добар, од 80 до 89 – многу добар и повеќе од 90 – одличен.
4. Да се определи дали ученикот добил преодна оценка ако условот за положување е освоени 60 бода од вкупно 100, со користење на условниот оператор ?:.
5. Да се определи дали кружниците $O(x_1, y_1, r_1)$ и $O(x_2, y_2, r_2)$ се сечат, со користење на условниот оператор ?:. (Растојание меѓу две точки се пресметува по формулата $d = \sqrt{(x_2 - x_1)^2 + (y_2 - y_1)^2}$).

Алгоритамска контролна структура за избор од повеќе можности случај и контролна наредба за избор од повеќе можности switch

Кога има повеќе можности за продолжување на дејството во алгоритамот, се користи **алгоритамската контролна структура за избор од повеќе можности**, ***слика 2.2.16***. Изборот на една од можностите се врши, во зависност од вредноста на некој податок или на некој аритметички *израз*.

Вредноста на *израз* може да биде a, b... k или може *израз* да не добие ниту една од тие вредности. Во случај *израз* да добие вредност a, дејството продолжува со чекорот А, во случај вредноста на *израз* да е b, дејството продолжува со чекорот Б итн. Во случај *израз* да не добие ни-

случај *израз*
    a: чекор А;
       прекин;
    b: чекор Б;
       прекин;
    . . .
    k: чекор К;
       прекин;
    инаку
       чекор Х;
крај_случај {*израз*}

Слика 2.2.16

ту една од вредностите a, b... k, тогаш дејството продолжува со чекорот X.

Од досега кажаното, се гледа дека со оваа алгоритамска контролна структура се врши избор на еден од повеќе можни случаи на продолжување на дејството. Затоа, таа се нарекува **избор во случај** или, пократко, само **случај** (англ. case).

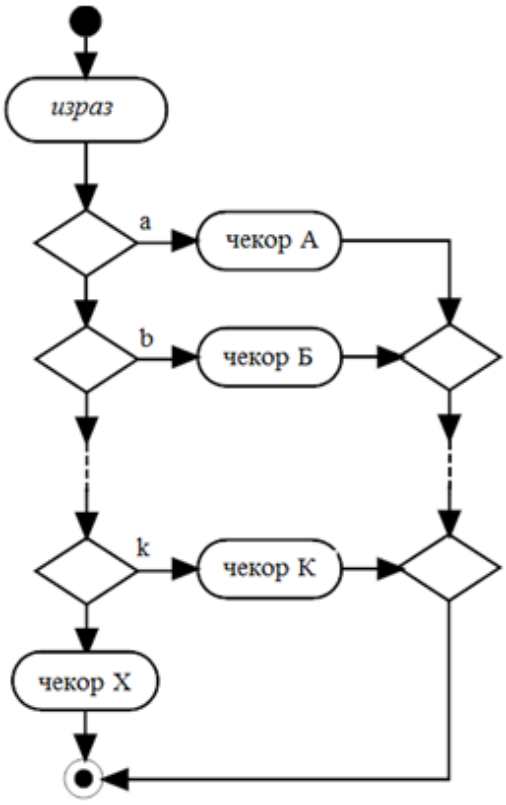
По чекорите a, b... k, е наведена алгоритамската контролна структура **прекин**, со која се прекинува извршувањето на останатите чекори во контролната структура **случај** и дејството продолжува со чекорот или контролната структура по **случај**. (За алгоритамската контролна структура **прекин** ќе зборуваме повеќе во потточката **2.4 Алгоритамски контролни структури за скок и контролни наредби за скок**).

Оваа контролна структура графички е претставена на ***слика 2.2.1*** .

Кога е потребно да се изврши ист чекор за повеќе различни вредности на *израз*, тогаш структурата **случај** се запишува како на ***слика 2.2.18***.

```
случај израз
      a, c, d:   чекор A;
                 прекин;
      b, g:      чекор B;
                 прекин;
      . . .
      k, i, f, j: чекор K;
                 прекин;
      инаку
                 чекор X;
крај_случај {израз}
```

Слика 2.2.18



Слика 2.2.1

Со контролната наредба **switch** се реализира алгоритамската контролна структура за избор од повеќе можности **случај**, а нејзината синтакса е дадена на ***слика 2.2.19***.

Изразот во заграда (*izraz)* може да биде кој било израз чии вредности се од целобројен тип (short, int, long, long long), знаковен тип (char), логички тип (bool) или набројив тип (enum). Ознаките a, b... k се константи од целобројен, знаковен, логички или набројив тип. Тие не може да бидат променливи.

```
switch(izraz) {
       case   a: naredba A;
                 break;
       case   b: naredba B;
                 break;
       ...
       case   k: naredba K;
                 break;
       default:
                naredba X;
}
```

Слика 2.2.19

За секој случај (case), се извршуваат сите наредби сè до првата контролна наредба break, чие дејство е скок (излез) на крајот на контролната наредба switch. Затоа, мора да се користи контролната наредба break по секое case. Ако не се наведе контролната наредба break, извршувањето продолжува со следното case[2]. По naredba X, не се става break.

Ако *izraz* не добие ниту една вредност од наведените константи по зборот case, тогаш се извршува naredba X по зборот default. Одделот default е опционален, не мора да се наведе. Ако не постои, по извршување на кое било case, се скока на следната наредба по контролната наредба switch.

Примери

Пример 2.2.9

Програмскиот сегмент за потврда дали сте млад/а или сте возрасен/на е:

```
bool DaNe;
char DN;
cout << "Dali ste roden/i vo 21 vek? (D, N): "; cin >> DN;
DaNe = true;
if(DN == 'N')
       DaNe = false;
switch(DaNe) {
       case true:    cout << "Vie ste mlad/a." << endl;
              break;
       case false:   cout << "Vie ste vozrasen/na." << endl;
}
```

Пример 2.2.10

Да се внесе знак за аритметичката операција: +, –, * или / и да се отпечати резултатот.

```
switch(znak) {
case '+': cout << znak << "plus " << endl;
       break;
case '-': cout << znak << "minus " << endl;
       break;
case '*': cout << znak << "po " << endl;
```

[2] Ова е многу честа грешка, затоа треба да се внимава да се стави break на крајот на секое case.

```
        break;
case '/': cout << znak << " vrz " << endl;
        break;
default:
        cout << znak << " ne e znak za aritmeticka operacija." << endl;
}
```

Иста група наредби може да се извршат за различни случаи (cases) сè до првата наредба break.

Пример 2.2.11

Да се внесе буква и да се отпечати дали е самогласка или е согласка.

```
switch(bukva) {
case 'a':
case 'e':
case 'i':
case 'o':
case 'u':
        cout << "Bukvata " << bukva << " e samoglaska." << endl;
        break;
default:
        cout << "Bukvata " << bukva << " e soglaska." << endl;
}
```

Пример 2.2.12

Ако некој месец во годината е зададен со неговиот реден број, тогаш со контролната наредба switch, може да се одреди колку дена има тој месец.

Напомена: Во престапната година месецот февруари има 29 дена. Престапна година е онаа која е делива со 4 и не е делива со 100 или која е делива со 400.

```
switch(mesec){
        case 1: case 3: case 5: case 7: case 8: case 10: case 12:
                denovi = 31;
                break;
        case 4: case 6: case 9: case 11:
                denovi = 30;
                break;
        case 2:
                denovi = (((godina % 4 == 0) && (godina % 100 != 0)) ||
                (godina % 400 == 0)) ? 29 : 28;
}
        cout << mesec << "-ot mesec vo " << godina << " godina ima "
        << denovi << " denovi." << endl;
```

Пример 2.2.13

Програмскиот сегмент за претходната задача кога се месеците од набројв тип ќе биде:

```
enum meseci {JANUARI, FEVRUARI, MART, APRIL, MAJ, JUNI,
             JULI, AVGUST, SEPTEMVRI, OKTOMVRI, NOEMVRI, DEKEMVRI
}
meseci mesecEnum = FEVRUARI;
switch(mesecEnum) {
       case JANUARI: case MART: case MAJ: case JULI: case AVGUST:
       case OKTOMVRI: case DEKEMVRI:
              denovi = 31;
              break;
       case APRIL: case JUNI: case SEPTEMVRI: case NOEMVRI:
              denovi = 30;
              break;
       case FEVRUARI:
              denovi = (((godina % 4 == 0) && (godina % 100 != 0)) ||
              (godina % 400 == 0)) ? 29 : 28;
}
       cout << mesecEnum << "-ot mesec vo " << godina << "godina ima "
       << denovi << " denovi." << endl;
```

Напишете ги претходните 5 примери во една програма и извршете ги.

Решени задачи

Задача 2.2.6

Да се напише програма со која ќе се отпечати општиот успех на ученик кога е даден бројниот успех 5, 4, 3, 2 или 1.

Програмата е дадена на ***слика 2.2.20***.

```
1   // Opst uspeh na ucenik
2   #include <iostream>
3   using namespace std;
4
5   int main() {
6       int ocena;
7       cout << "Vnesete ocena (1,2,3,4,5): ";
8       cin >> ocena;
9       cout << "Opstiot uspeh na ucenikot e: ";
10      switch( ocena ) {
11          case 5:     cout << "Odlicen" << endl;
12              break;
13          case 4:     cout << "Mnogu Dobar" << endl;
14              break;
```

Слика 2.2.20

```
15          case 3:     cout << "Dobar" << endl;
16              break;
17          case 2:     cout << "Dovolen" << endl;
18              break;
19          case 1:     cout << "Nedovolen" << endl;
20              break;
21          default:    cout << "Toa ne e ocena" << endl;
22      }
23
24      cout << endl;
25      system( "Color 17" );
26      system( "pause" );
27      return 0;
28  }
```

Слика 2.2.20 продол ение

Еден излез при извршување на програмата е:

```
Vnesete ocena (1,2,3,4,5): 5
Opstiot uspeh na ucenikot e: Odlicen

Press any key to continue . . .
```

Задача 2.2.7

Да се напише програма со која за внесена цифра меѓу 1 и 9 ќе се отпечати дали е парна или непарна. Ако цифрата не е меѓу 1 и 9 да се отпечати коментар.

```
1   // Parni i neparni cifri
2   #include <iostream>
3   using namespace std;
4
5   int main() {
6       short cifra;
7       cout << "Vnesete cifra od 1 do 9: ";
8       cin >> cifra;
9       switch( cifra ) {
10          case 1: case 3: case 5: case 7: case 9:
11              cout << "Vnesena e neparnata cifra " << cifra << endl;
12              break;
13          case 2: case 4: case 6: case 8:
14              cout << "Vnesena e parnata cifra " << cifra << endl;
15              break;
16          default: cout << "Vneseniot znak ne e cifra: " << cifra << endl;
17      }
18
19      cout << endl;
20      system( "Color 17" );
21      system( "pause" );
22      return 0;
23  }
```

Слика 2.2.21

Еден излез од извршување на програмата е:

```
Vnesete cifra od 1 do 9: 5
Vnesena e neparnata cifra 5

Press any key to continue . . .
```

Задачи за вежбање

За следниве задачи да се напишат програми со користење на контролната наредба за избор од повеќе можности switch:

1. Да се внесе природен број помал од 10 и да се отпечати дали е прост[3], делив со 2, делив со 3 или совршен[4].
2. Да се отпечати следново мени:

```
a. Profesor po makedonski jazik
b. Profesor po informatika
c. Profesor po matematika
d. Profesor po fizika

Izberete: a
```

   Потоа, во зависност од внесената буква (a, b, c, d), да се отпечати името на предметниот професор.

1. Да се внесе цифра 1, 2, 3... 9, 0 и да се отпечати цифрата со зборови. На пример: за 1 да се отпечати зборот „еден“, за 2 да се отпечати зборот „два“ итн.
2. Да се одреди арапскиот еквивалент на римска цифра. (Римски цифри се: I = 1, V = 5, X = 10, L = 50, C = 100, D = 500 и M = 1 000).
3. Да се определи оценката за ученик според освоените бодови ако оценувањето се врши според следнава скала на бодови: помалку од 60 – недоволен, од 60 до 69 – доволен, од 70 до 79 – добар, од 80 до 89 – многу добар и од 90 до 99 – одличен.

Прашања за проверка на знаењето

1. Кога (најчесто) се користи алгоритамската редоследна контролна структура?
2. Кои алгоритамски контролни структури за избор ги знаете? Напишете ја нивната текстуална форма.
3. Со која алгоритамска контролна наредба во C++ се реализира алгоритамската контролната структура за избор од две можности **ако–тогаш–инаку**? Наведете неколку примери.

[3] Прост број е оној кој е делив само со 1 и со самиот себеси. Прости броеви се: 2, 3, 5, 7, 11 итн.

[4] Совршен број е оној кој е еднаков на збирот на своите делители, без самиот број. Совршени броеви се: 6 (= 1 + 2 + 3), 28 (= 1 + 2 + 4 + 7 + 14), 496 итн.

4. Претставете ја графички алгоритамската контролна структура **ако–тогаш–инаку**.
5. Каков е типот на изразот што се користи како услов во алгоритамските контролнии структури за избор и контролните наредби за избор?
6. Како се запишува контролната наредба за избор од две можности, кога по if и/или по else има повеќе од една наредба? Наведете примери.
7. Напишете ја синтаксата на контролната наредба за избор од две можности ако по едната можност нема наредби. Наведете примери.
8. Кои начини на вгнездување на контролните наредби за избор ги знаете?
9. Како се нарекува контролната наредба за избор во која има вгнездување во изборот else?
10. Објаснете што е висечко else. Наведете пример.
11. Напишете ја синтаксата на условен оператор. Наведете примери.
12. Претставете ја текстуално и графички алгоритамската контролна структура за избор од повеќе можности.
13. Каков може да биде типот на изразот во контролната наредба за избор од повеќе можности?
14. Која контролна наредба се става на крајот од секое case во контролната наредба switch?
15. Што ќе се случи ако на крајот на некое case нема наредба break?

Задачи

1. Што ќе се отпечати по извршување на следниот програмски сегмент ако вредноста на променливата n е 78?

```
if((n % 2 == 0) && (n % 3 == 0))
   cout << n << " e deliv so 6." << endl;
else
   cout << n << " ne e deliv so 6." << endl;
```

2. Која е грешката во следниов програмски сегмент:

```
if(x = a)
   cout << "x ima ista vrednost so a.";
else
   cout << "x ima razlicna vrednost od a.";
```

3. Зошто следниот програмски сегмент секогаш печати „false“, независно од вредноста на n?

```
cin >> n;
if(n = 0)
          cout << "true" << endl;
   else
          cout << "false" << endl;
```

4. Што ќе се отпечати со следниов програмски сегмент?

```
int i = 0, n;
float a = 1;
char c = 'a';
a++;
if(c == a && i == 0)
   n = 1;
else
   n = 2;
cout << "n = " << n << endl;
```

5. Дали следниве два програмски сегменти даваат исти резултати?

```
if(a > b)
   cout << a << " > " << b << endl;
else if(b < c)
   cout << a << " <= " << b << "<" << c << endl;
else
   cout << a << " <= " << b << ">=" << c << endl;
```

```
if(a > b)
   cout << a << " > " << b << endl;
if(b < c)
   cout << a << " <= " << b << " < " << c << endl;
else
   cout << a << " <= " << b << " >= " << c << endl;
```

6. Да се напише програмски сегмент за логички izraz од две променливи:
   - izraz е точен ако или prom1 или prom2 е точна.
   - izraz е неточен ако и prom1 и prom2 се точни.
   - izraz е неточен ако и prom1 и prom2 се неточни.
7. Да се внесат три цели броја и да се отпечати најголемиот: еднаш ако е поголем од другите два, двапати ако двата поголеми од третиот имаат иста вредност, трипати ако се сите три броја еднакви.
8. Да се дополни **Задача 2.2.4** со проверка дали внесената просечна оценка е во интервалот [0.0, 5.0].
9. Да се дополни **Задача 2.2.6** со проверка дали внесената оценка е во интервалот [1, 5].
10. Да се напише програма за печатење на следниов излез, зависно од променли-
11. Да се провери дали три реални броеви може да бидат страни на триаголник?
12. Напишете програма која ќе симулира едноставен калкулатор со следните операции: +, –, *, %, /, x^2.

2.3 Алгоритамски контролни структури за повторување и контролни наредби за повторување

Контролните структури за повторување се користат тогаш кога е потребно една група алгоритамски чекори да се извршат повеќепати. Едно извршување на чекорите се нарекува еден **циклус**, ***слика 2. .1***. Затоа, таквото повторување на извршувањето на група алгоритамски чекори се нарекува **циклично повторување**.

```
циклус
    чекор А;
    чекор Б;
      ...
    чекор М;
крај_циклус
```

Слика 2. .1

За полесно да се разберат овие контролни структури, ќе ја разгледаме следнава задача.

Задача: Да се најде збирот на првите 10 природни броеви.

Да замислиме дека имаме некој сад и во него ги ставаме броевите. На почетокот садот е празен, т. е. збирот е 0. Прв природен број кој го ставиме во садот е 1, значи сега збирот е 0 + 1 = 1. Потоа, во садот го ставаме следниот природен број 2, значи во садот ќе има два броја (1 и 2), а збирот ќе биде 0 + 1 + 2. Потоа, ги ставаме броевите 3, 4, 5, 6, 7, 8, 9 и 10, а збирот ќе се зголемува со додавање на секој број. Крајниот збир ќе биде 0 + 1 + 2 + 3 + 4 + 5 + 6 + 7 + 8 + 9 + 10.

Може да забележиме дека во постапката за наоѓање на збирот се извршуваат само две дејства:

- Додавање број на збирот.
- Зголемување на бројот за 1.

Бидејќи бројот се менува од 1 до 10, а збирот е 0, 1, 3, 6, 10 итн., можеме да воведеме две променливи broj и zbir. На почетокот, содржината на променливата zbir е 0, а почетна вредност на broj е 1. Затоа, променливите broj и zbir ги иницијализираме на вредности 1 и 0. Алгоритамот може да се запише текстуално како на ***слика 2. .2.***

```
zbir ← 0;
broj ← 1;
циклус 10 пати
        – додади го broj на zbir;
        – зголеми го broj за 1;
крај_циклус
```

Слика 2. .2

Зависно од начинот на прекин на повторувањето, разликуваме три контролни структуру за повторување, и тоа:

- Контролна структура со броење на циклусите.
- Контролна структура со излез на почетокот од циклусот.
- Контролна структура со излез на крајот од циклусот.

Алгоритамски контролни структури за повторување со броење на циклусите и контролна наредба for

Во оваа контролна структура, бројот на повторувања на циклусот е однапред познат. Повторувањата, т. е. циклусите се бројат со посебен brojac, за кој се задаваат почетната и крајната вредност, ***слика 2. .*** .

Циклусите се извршуваат **за** секоја вредност на променливата brojac од pocetna, со **зголемување** по 1, **до** krajna. Променливата brojac се зголемува за 1 пред секој циклус. Затоа, ваквата контролна структура се нарекува **за–зголемувај–до**.

```
за brojac ← pocetna зголемувај до krajna
        чекор А;
        чекор Б;
        ...
        чекор К;
крај_за {brojac}
```

Слика 2. .

Променливите brojac, pocetna и krajna мора да бидат променливи од ист тип. Променливата pocetna има помала или еднаква вредност од променливата krajna. Во спротивно, не се извршува ниту еден циклус. Вредноста на brojac не смее да се менува во алгоритамските чекори од кои се состои циклусот.

Ако е потребно повторување во кое почетната (pocetna) вредност трба да биде поголема од крајната (krajna) вредност, тогаш се користи слична контролна структура. Во неа, циклусите се извршуваат **за** секоја вредност на променливата brojac од pocetna, со **намалување** по 1, **до** krajna. Променливата brojac се намалува за 1 пред секој циклус. Затоа. ваквата контролна структура се нарекува **за–намалувај–до**, ***слика 2. .4***.

```
за brojac ← pocetna намалувај до krajna
        чекор А;
        чекор Б;
        ...
        чекор J;
крај_за {brojac}
```

Слика 2. .4

Во алгоритми (задачи) каде што brojac не се зголемува или намалува по 1, туку по некој iznos на чекорот, контролната структура се нарекува **за–до–чекор**, ***слика 2. .5.***

```
за brojac ← pocetna до krajna чекор iznos
        чекор А;
        чекор Б;
        ...
        чекор М;
крај_за {brojac}
```

Слика 2. .5

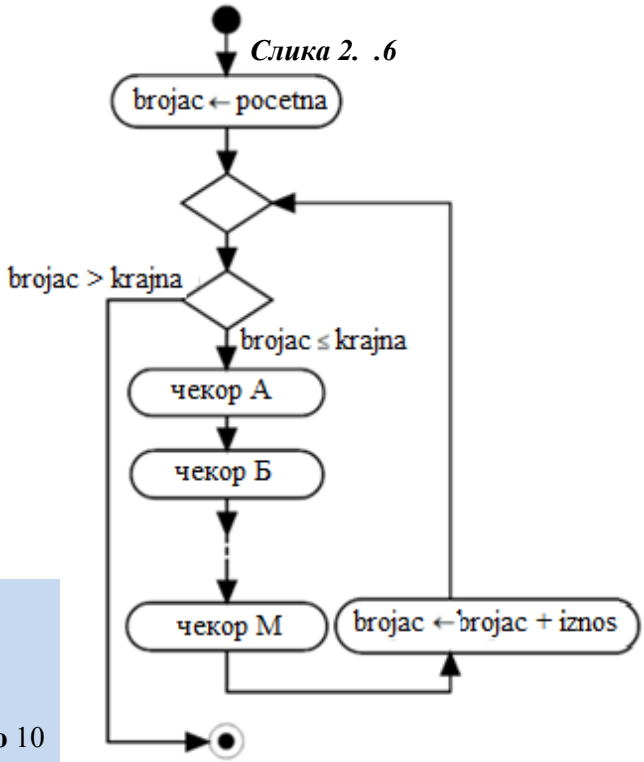
Во оваа контролна структура, **за** некоја pocetna вредност на brojac, **до** некоја krajna вредност, тој се зголемува во секој циклус со **чекор** за одреден iznos. Затоа, оваа контролна структура за повторување се нарекува **за–до–чекор.**

Ако износот (iznos) на чекорот во контролната структура **за–до–чекор** е +1, се добива контролната структура **за–зголемувај–до**, а ако iznos на чекорот е –1, се добива контролната структура **за–намалувај–до**.

Графичкиот приказ на контролната структура **за–до–чекор** е даден на ***слика 2. .6***.

Алгоритамот за задачата за наоѓање на збирот на првите 10 природни броеви, изразен со контролната структура **за–зголемувај–до** е даден на ***слика 2. .*** .

Слика 2. .6




```
алгоритам ЗбирДо10
почеток
        zbir ← 0;
        broj ← 1;
        за brojac ← 1 зголемувај до 10
                zbir ← zbir + broj;
                broj ← broj + 1;
        крај_за {brojac}
        печати zbir;
крај {ЗбирДо10}
```

Слика 2. .

За реализација на трите контролни структури: **за–зголемувај–до, за–намалувај–до** и **за–до–чекор**, во C++ се користи контролната наредба for.

Општа форма на контролната наредба for е дадена на ***слика 2. .8.***

```
for (иницијализација; услов; ажурирање){
                naredba A;
                naredba B;
                ...
                naredba R;
}
```

Слика 2. .8

Иницијализација е дел во кој се задава почетната вредност на бројачот. Овој дел се извршува еднаш пред да започне повторувањето на циклусите.

Услов е делот за испитување на условот пред повторно да се изврши циклусот. Тој е логички израз. Ако *услов* има вредност true, тогаш циклусот повторно ќе се изврши, а ако има вредност false, тогаш циклусот не се извршува, туку дејството продолжува со следната наредба. Бидејќи условот може да не биде исполнет и пред започнување на првиот циклус, може да не се изврши ниту еден циклус.

Делот *ажурирање* на контролната наредба for се извршува по извршување на секој циклус, а најчесто во него се ажурира бројачот.

Формата на контролната наредба for за контролната структура **за–до–чекор** е дадена на ***слика 2. .9***:

```
for (brojac = pocetna; brojac <= krajna; brojac = brojac + iznos) {
                    naredba A;
                    naredba B;
                    ...
                    naredba L;
}
```

Слика 2. .9

Во делот за иницијализација, променливата brojac се иницијализира на pocetna вредност. Тој дел се извршува само еднаш. Испитувањето на *услов* дали вредноста на brojac е помала од или еднаква на крајната (krajna) вредност (brojac <= krajna), е вградено во самата наредба. Условот (*услов*) brojac <= krajna се испитува пред почетокот на секој циклус и ако има вредност true, циклусот се извршува, а ако има вредност false, циклусот не се извршува.

Програмата за задачата за наоѓање на збирот на првите 10 природни броеви, според алгоритамот од ***слика 2. .*** , со користење на наредбата for, е дадена на ***слика 2. .10***.

```
1   #include <iostream>
2   using namespace std;
3
4   int main() {    // Zbir na prvite 10 prirodni broevi so for
5
6       int brojac, broj, zbir;
7       zbir = 0;
8       broj = 1;
9       for( brojac = 1; brojac <= 10; brojac = brojac + 1 ) {
10          zbir = zbir + broj;
11          broj = broj + 1;
12      }
13      cout << "Zbirot na prvite 10 prirodni broevi e " << zbir << endl;
14
15      cout << endl;
16      system( "Color 17" );
17      system( "pause" );
18      return 0;
19  }
```

Слика 2. .10

```
Zbirot na prvite 10 prirodni broevi e 55

Press any key to continue . . .
```

Примери

Пример 2.3.1

Да се најде збирот на првите 10 природни броеви, со собирање од 10 до 1, т. е. 10 + 9 + 8 + 7 + 6 + 5 + 4 + 3 + 2 + 1.

Програмата е дадена на ***слика 2. .11***. Гледаме дека програмата е иста со онаа на ***слика 2. .10***, со таа разлика што променливата broj се иницијализира на 10, бројачот brojac се иницијализира на 10, условот е сменет во brojac >= 1 и во делот за ажурирање на наредбата for, бројачот се намалува за 1.

```
#include <iostream>
using namespace std;

int main() {    // Zbir na prvite 10 prirodni broevi so for

    int brojac, broj, zbir;
    zbir = 0;
    broj = 10;
    for( brojac = 10; brojac >= 1; brojac = brojac - 1 ) {
        zbir += broj;
        broj += 1;
    }
    cout << "Zbirot na prvite 10 prirodni broevi e " << zbir << endl;

    cout << endl;
    system( "Color 17" );
    system( "pause" );
    return 0;
}
```

Слика 2. .11

Пример 2.3.2

Да се отпечати таблицата за множење до 10 со внесен број.

Програмата е дадена на ***слика 2. .12***. Во неа е вклучена библиотеката <iomanip> од која се користи функцијата setw(), со која се задава бројот на места во кои ќе се отпечатат следната променлива и функцијата right, со која вредноста што се печати се наслонува на десната страна.

На пример, ако вредноста на x е 1 234 со наредбата:

```
cout << setw(10 << right << x << endl;
```

вредноста на x ќе се отпечати десно наслонета во поле од 10 места:

␣␣␣␣␣␣123.

```cpp
#include <iostream>
#include <iomanip>
 using namespace std;

int main() {    // Tablica mnozenje so for

    int brojac, broj;
    cout << " Tablica mnozenje so (1,2,...,9): ";
    cin >> broj;
    for( brojac = 1; brojac <= 10; brojac = brojac + 1 ) {
        cout << "\n" << setw( 3 ) << right << brojac << " x " << broj << " = "
            << setw( 3 ) << brojac * broj;
    }

    cout << endl;
    system( "Color 17" );
    system( "pause" );
    return 0;
}
```

Слика 2. .12

```
 Tablica mnozenje so (1,2,...,9): 7

  1 x 7 =   7
  2 x 7 =  14
  3 x 7 =  21
  4 x 7 =  28
  5 x 7 =  35
  6 x 7 =  42
  7 x 7 =  49
  8 x 7 =  56
  9 x 7 =  63
 10 x 7 =  70
Press any key to continue . . .
```

Пример 2.3.3

Да се отпечатат буквите од G до g без знаците помеѓу големите и малите букви.

Декларацијата со иницијализација на бројачот bukva, може да се направи и во делот за иницијализација на наредбата for, како што е во следната програма.

```cpp
#include <iostream>
using namespace std;

int main() {    // Pecatenje na bukvite od G do g (so for)

    for( char bukva = 'G'; bukva <= 'g'; bukva = bukva + 1 ) {
        cout << bukva << " ";
    }

```

Слика 2. .1

```
10      cout << endl;
11      system( "Color 17" );
12      system( "pause" );
13      return 0;
14  }
```

Слика 2. .1 продол ение

```
g f e d c b a Z Y X W V U T S R Q P O N M L K J I H G
Press any key to continue . . .
```

Пример 2.3.4

Да се табелира функцијата $f(x) = 3x^2 – 2x + 4$ за $x \in [–30,30]$ со износ на чекорот 4, како што е прикажано во добиениот излез. Програмата е дадена на ***слика 2. .14.***

```
1   #include <iostream>
2   #include <iomanip>
3   using namespace std;
4
5   int main() {    // Tabeliranje na funkcija (so for)
6
7       double x, y;
8       cout << setw( 7 ) << "x" << setw( 15 ) << "y=3x^2-2x+4" << endl << endl;
9       for( int brojac = -30; brojac <= 30; brojac += 4 ) {
10          x = brojac;
11          y = 3 * x * x - 2 * x + 4;
12          cout << setw( 10 ) << x << setw( 10 ) << y << endl;
13      }
14
15      cout << endl;
16      system( "Color 17" );
17      system( "pause" );
18      return 0;
19  }
```

Слика 2. .14

Излезот од програмата е:

```
      x     y=3x^2-2x+4

    -30          2764
    -26          2084
    -22          1500
    -18          1012
    -14           620
    -10           324
     -6           124
     -2            20
      2            12
      6           100
     10           284
     14           564
     18           940
     22          1412
     26          1980
     30          2644
```

Оператори за инкрементирање и за декрементирање ++ , —

Во C++ се воведени посебни оператори за зголемување и за намалување на вредноста на податокот за 1. Тие се наречени **оператор за инкрементирање** (англ. increment operator) и **оператор за декрементирање** (англ. decrement operator). Операторот за инкрементирање е ++, а за декрементирање е – –. Инкрементирањето или декрементирањето на вредноста на податокот x може да се врши во префиксна (++x или – –x) или постфиксна (x++ или x– –) форма. Во првиот случај, инкрементирањето или декрементирањето се извршува пред пресметување на изразот (прединкремент) во кој учествува податокот x, а во вториот по пресметување на изразот (постинкремент).

Оператор	Форма	Инкремент	Декремент
++	++x	префиксен	
	x++	постфиксен	
– –	– –x		префиксен
	x– –		постфиксен

Пример:

Вредност на x пред пресметувањето	Израз	Вредност на изразот	Вредност на x по пресметувањето
5	++x	6	6
5	x++	5	6
5	– –x	4	4
5	x– –	5	4

Инкрементирањето и декрементирањето може да се користат како наредба.

Примери

Пример 2.3.5

Збирот на првите 10 природни броеви од претходните примери може да се пресмета со следниов програмски сегмент:

```
int broj = 1;
int zbir = 0;
for (int brojac = 1; brojac <= 10; brojac++) {
       zbir += broj;
       ++broj;       // isto so broj = broj + 1; ili broj += 1;
}
```

Пример 2.3.6

Следниов програмски сегмент илустрира употреба на операторот за инкрементирање:

```
int a, b, c;
a = b = c = 1;
cout << "a = " << a << ", b = " << b << ", c = " << c << endl;
b = ++a;
cout << "b = ++a \na = " << a << ", b = " << b << ", c = " << c << endl;
c = a++;
cout << "c = a++ \na = " << a << ", b = " << b << ", c = " << c << endl;
c = ++ ++b;
cout << "c = ++ ++b \na = " << a << ", b = " << b << ", c = " << c << endl;
```

Неговиот излез е:

```
a = 1, b = 1, c = 1
b = ++a
a = 2, b = 2, c = 1
c = a++
a = 3, b = 2, c = 2
c = ++ ++b
a = 3, b = 4, c = 4

Press any key to continue . . .
```

Операторите за инкрементирање и за декрементирање може да се користат само за линеарно подредени множества, како што се знаците ASCII и целите броеви. (Линеарно подредени множества се оние во кои секој елемент има свој претходник (освен првиот) и свој следбеник (освен последниот)).

Во претходните примери, место наредбите

```
brojac = brojac + 1;
brojac = brojac - 1;
```

може да се користат наредбите

```
brojac++;     или    ++brojac;
brojac--;     или    --brojac;
```

а место наредбите

```
bukva = bukva + 1;
bukva = bukva - 1;
```

може да се користат наредбите

```
bukva++;      или    ++bukva;
bukva--;      или    --bukva;
```

Користење на бројачот во телото на контролната наредба for

Во програмата за збирот на првите 10 природни броеви од 1 до 10, може да се забележи дека променливата broj добива почетна вредност 1 и се инкрементира за 1 во секој циклус. Исто така, и бројачот (brojac) се иницијализира на 1 и во секој циклус се инкрементира за 1. Ако ги отпечатиме нивните вредности во секој циклус, ќе видиме дека се исти.

```
brojac  broj

  1       1
  2       2
  3       3
  4       4
  5       5
  6       6
  7       7
  8       8
  9       9
  10      10
 Zbirot na prvite 10 prirodni broevi e 55
```

Во ваквите случаи, променливата brojac може да се искористи место променливата broj. Меѓутоа, почетниците во програмирањето треба да бидат многу внимателни при користењето на бројачот во телото на контролните наредби за повторување бидејќи лесно се прават грешки. И, уште нешто. Бројачот не може секогаш да се користи во телото на контролните наредби за повторување.

Програмата за збирот на првите 10 природни броеви, со користење на brojac во телото на наредбата for, ќе биде како на ***слика 2. .15***.

Исто така, во неа е покажано дека декларацијата на бројачот може да се врши и во делот за иницијализација.

```
1   #include <iostream>
2   using namespace std;
3
4   int main() {    // Zbir na prvite 10 prirodni broevi so for
5
6       int zbir = 0;
7       for( int brojac = 1; brojac <= 10; brojac++ ) {
8           zbir = zbir + brojac;
9       }
10      cout << "Zbirot na prvite 10 prirodni broevi e " << zbir << endl;
11
```

Слика 2. .15

```
12      cout << endl;
13      system( "Color 17" );
14      system( "pause" );
15      return 0;
16  }
```

Слика 2. .15 продол ение

Специфичности на контролната наредба for

а) Иницијализација и ажурирање на повеќе променливи

Контролната наредба **for** во C++ дозволува иницијализација и ажурирање на повеќе променливи. Притоа тие се одделуваат со запирки.

Пример 2.3.7
Да се провери дали збирот на броевите од 1 до n и од n до 1 е ист?

Во програмата (***слика 2. .16***), во делот за иницијализирање и во делот за ажурирање на наредбата for, се иницијализираат и се ажурираат две променливи.

Условот за прекин на повторувањето може да биде кој било логички израз во кој може да се користат релациски и логички оператори. Во оваа програма условот е i <= n && j >= 1.

```
1   #include <iostream>
2   using namespace std;
3
4   int main() {
5       // Dali zbirot 1+2+...+n e ist so zbirot n+(n-1)+(n-2)+...+2+1? (so for)
6
7       int n, zbir1doN = 0, zbirNdo1 = 0;
8       cout << "Do koj broj, n=";
9       cin >> n;
10      for( int i = 1, j = n; i <= n && j >= 1; i++, j-- ) {
11          zbir1doN += i;
12          zbirNdo1 += j;
13      }
14      cout << "Zbirot na prirodnite broevi od 1 do " << n << " e " << zbir1doN << endl;
15      cout << "Zbirot na prirodnite broevi od " << n << " do 1 e " << zbirNdo1 << endl;
16
17      cout << endl;
18      system( "Color 17" );
19      system( "pause" );
20      return 0;
21  }
```

Слика 2. .16

Еден излез од извршување на програмата е:

```
Do koj broj, n=100
Zbirot na prirodnite broevi od 1 do 100 e 5050
Zbirot na prirodnite broevi od 100 do 1 e 5050

Press any key to continue . . .
```

б) Делот за иницијализација и ажурирање може да биде и празен

Пример 2.3.8

Програмата од **Пример 2.3.7** може да се напише и како на ***слика 2. .1*** . Гледаме дека иницијализацијата се врши пред почетокот на наредбата for, а ажурирањето се врши во телото на наредбата for.

```
1   #include <iostream>
2   using namespace std;
3
4   int main() {
5       // Dali zbirot 1+2+...+n e ist so zbirot n+(n-1)+(n-2)+...+2+1? (so for)
6
7       int i, j, n, zbir1doN = 0, zbirNdo1 = 0;
8       cout << "Do koj broj, n=";
9       cin >> n;
10      i = 1; j = n;
11      for( ; i <= n && j >= 1;) {
12          zbir1doN += i;
13          zbirNdo1 += j;
14          i++;
15          j--;
16      }
17      cout << "Zbirot na prirodnite broevi od 1 do " << n << " e " << zbir1doN << endl;
18      cout << "Zbirot na prirodnite broevi od " << n << " do 1 e " << zbirNdo1 << endl;
19
20      cout << endl;
21      system( "Color 17" );
22      system( "pause" );
23      return 0;
24  }
```

Слика 2. .1

```
Do koj broj, n=99
Zbirot na prirodnite broevi od 1 do 99 e 4950
Zbirot na prirodnite broevi od 99 do 1 e 4950

Press any key to continue . . .
```

Вежби

Вежба 2.3.1

Што прави следниов програмски сегмент?

```
zbir = 0;
for(int broj = 2, broj <= 10; broj += 2)
	zbir += broj;
```

Вежба 2.3.2

Да се утврди што ќе се отпечати како резултат на извршување на следниов програмски сегмент:

```
for(int x = 1, i = 0; x <= 10; x++)
	i += x;
cout << x << endl;
```

Вежба 2.3.3

Да се утврди што ќе се отпечати како резултат на извршување на следниов програмски сегмент:

```
for(int i = 0, j = 0; i < 5; i++, j--)
	cout << i + j << endl;
```

Решени задачи

Задача 2.3.1

Да се најдат сите *Питагорини броеви* помали од природниот број n.

Објаснува е Питагорини броеви се секоја тројка природни броеви x, y и z што го задоволуваат равенството $x^2 + y^2 = z^2$. За да не се повторуваат тројките, како 3, 4, 5 и 4, 3, 5, променливата x ќе се менува од 1 до вредност не поголема од y, т. е. до (ако x = y) $\left[\sqrt{\frac{n^2}{2}}\right]$. Средните загради означуваат цел дел од децимален број. На пример, [3.56] = 3.

Напомена* 1:** Во следнава програма (слика 2. .18***) се користат две помошни променливи pom1 и pom2, со кои се пресметува вредноста на $\left[\sqrt{\frac{n^2}{2}}\right]$ и на $x^2 + y^2$. Бидејќи вредноста $\left[\sqrt{\frac{n^2}{2}}\right]$ е константа, ако ја оставиме во наредбата for, ќе се пресметува во секој циклус, што е непотребно. Исто така, изразот $x^2 + y^2$ се пресметува двапати во секој циклус, а со воведување на pom2, само еднаш. На тој начин, времето за пресметување на изразот $x^2 + y^2$ се скратува двојно.

***Напомена* 2:** Бидејќи резултатот од квадратниот корен на реален број е реален број, а во циклусот ни треба цел број, со (int) sqrt(n * n / 2), вршиме принудна конверзија (наречена кастирање) на реалниот број во цел број.

```
#include <iostream>
#include <iomanip>
using namespace std;

int main() {
    // Pitagorini trojki (so for)

    int n, x, y, z, pom1, pom2, brojac = 0;
    cout << "Do koj priroden broj: ";   cin >> n;
    cout << "\nPitagorini trojki do brojot " << n << " se:" << endl;
    pom1 = ( int ) sqrt( n * n / 2 );
    for( x = 1; x <= pom1; x++ ) {
        for( y = x + 1; y <= n; y++ ) {
            pom2 = x * x + y * y;
            z = ( int ) sqrt( pom2 );
            if( ( z <= n ) && ( pom2 == z * z ) ) {
                ++brojac;
                cout << setw( 10 ) << brojac << ": " << setw( 2 ) << x << ", "
                    << setw( 2 ) << setw( 2 ) << y << ", " << setw( 2 ) << z << endl;
            }
        }
    }

    cout << endl;
    system( "Color 17" );
    system( "pause" );
    return 0;
}
```

Слика 2. .18

Еден излез од програмата е:

```
Do koj priroden broj: 33

Pitagorini trojki do brojot 33 se:
         1:  3,  4,  5
         2:  5, 12, 13
         3:  6,  8, 10
         4:  7, 24, 25
         5:  8, 15, 17
         6:  9, 12, 15
         7: 10, 24, 26
         8: 12, 16, 20
         9: 15, 20, 25
        10: 18, 24, 30
        11: 20, 21, 29

Press any key to continue . . .
```

Задача 2.3.2
Да се најдат сите совршени броеви помали од природниот број n.

Објаснува е Природните броеви кои се еднакви на збирот на своите делители, без самиот број, се нарекуваат совршени броеви. Совршени броеви се: 6, 28, 496 итн.

```
1   #include <iostream>
2   #include <iomanip>
3   using namespace std;
4
5   int main() {
6       // Sovrseni broevi (so for)
7
8       int n, broj, delitel, zbir;
9       cout << "Do koj broj, n=";  cin >> n;
10      cout << "\nSovrseni broevi do " << n << " se:" << endl;
11      for( broj = 2; broj <= n; broj++ ) {
12          zbir = 1;
13          for( delitel = 2; delitel <= broj / 2; delitel++ ) {
14              if( broj % delitel == 0 )
15                  zbir += delitel;
16          }
17          if( zbir == broj )
18              cout << broj << ", ";
19      }
20      cout << endl;
21
22      cout << endl;
23      system( "Color 17" );
24      system( "pause" );
25      return 0;
26  }
```

Слика 2. .19

Резултатот од извршување на програмата дадена на ***слика 2. .19*** е:

```
Do koj broj, n=12345

Sovrseni broevi do 12345 se:
6, 28, 496, 8128,

Press any key to continue . . .
```

Задачи за вежбање

За следниве задачи да се напишат програми со користење на контролната наредба for:

1. Да се пресмета збирот 1 + 4 + 7 + 11 +… + n.
2. Да се пресмета збирот $1^2 + 2^2 + 3^2 + \ldots + n^2$.
3. Да се пресмета збирот $1+\frac{1}{2}+\frac{1}{3}+\ldots+\frac{1}{n}$.
4. Да се пресмета аритметичката средина на природните броеви од 1 до n.
5. Да се пресмета n! = 1 · 2 · 3 ·… · n. По дефиниција 0! = 1 и 1! = 1. (Ознаката n! се чита „n факториел“).
6. Да се пресмета производот: $1\cdot\frac{1}{2}\cdot\frac{1}{3}\cdot\ldots\cdot\frac{1}{n}$.
7. Да се отпечатат буквите од абецедата од A до Z и од Z до A.
8. Да се отпечатат следниве три форми:

```
*
**
***
****
*****
******
*******
********
*********
```

```
1
2 1
3 2 1
4 3 2 1
5 4 3 2 1
6 5 4 3 2 1
7 6 5 4 3 2 1
8 7 6 5 4 3 2 1
9 8 7 6 5 4 3 2 1
```

```
                1
              1 2
            1 2 3
          1 2 3 4
        1 2 3 4 5
      1 2 3 4 5 6
    1 2 3 4 5 6 7
  1 2 3 4 5 6 7 8
1 2 3 4 5 6 7 8 9
```

9. Да се пресмета збирот:
   1 + (1 + 2) + (1 + 2 + 3) + (1 + 2 + 3 + 4) +… + (1 + 2 +… + n).
10. Да се најде двоен факториел на ненегативниот број n.
    n!! = n(n – 2)(n – 4) ·.... ·5 · 3 · 1 за n непарен,
    n!! = n(n – 2)(n – 4) ·.... · 6 · 4 · 2 за n парен.
    По дефиниција 0!! = 1 и 1!! = 1.
11. Да се внесат n броеви и да се изброи колку од нив се парни, а колку се непарни.
12. Да се најдат делителите на природниот број n.

Алгоритамска контролна структура за повторување со излез на почетокот од циклусот додека–извршувај и контролна наредба while

Во задачата за наоѓање на збирот на првите 10 природни броеви од претходниот поднаслов, рековме дека се повторуваат само две дејства:

- Додавање broj на zbir.
- Зголемување на broj за 1.

Променливата broj се зголемува во секој циклус до вредност 11. Затоа, можеме да кажеме: **додека** broj има помала или еднаква вредност на 10, се **извршуваат** циклуси со овие две дејства. Ваквата алгоритамска контролна структура се нарекува **додека–извршшувај**. Таа е претставена на ***слика 2. .20***.

```
zbir ← 0;
broj ← 1;
додека broj ≤ 10 извршувај
        – додади го broj на zbir;
        – зголеми го broj за 1;
крај_ додека {broj ≤ 10}
```

Слика 2. .20

Општа форма на оваа контролна структура може да се запише како на ***слика 2. .21***.

```
додека услов извршувај
        чекор А;
        чекор Б;
          ...
        чекор М;
крај_додека {услов}
```

Слика 2. .21

Структурата се состои од чекорите *А, Б... М*. Едно извршшување на чекорите од *А* до *М* е еден циклус. Пред почетокот на секој циклус, се испитува дали е исполнет условот (*услов*), кој претставува логички израз со две вредности: *точно* (true) или *неточно* (false). Додека е исполнет (точен) *услов*, се извршуваат чекорите од циклусот и дејството се враќа на почетокот од циклусот. Кога во некое испитување на *услов*, ќе се констатира дека тој не е исполнет (не е точен), се прескокнува контролната структура и дејството продолжува со следниот чекор или со следната алгоритамска контролна структура по **крај_додека** {*услов*}.

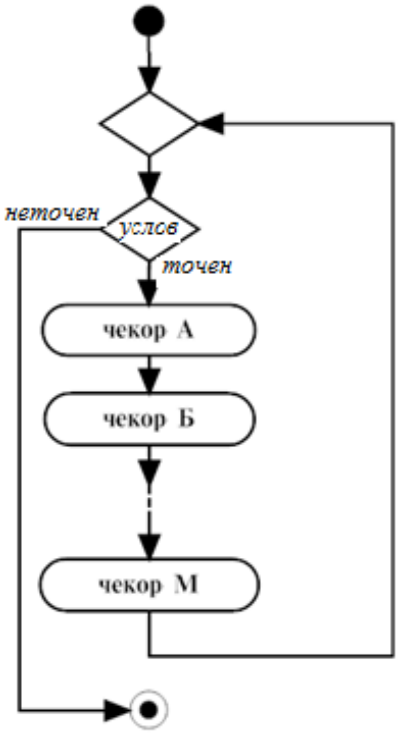
Бидејќи *услов* се испитува пред почетокот на секој циклус, може да се случи *услов* да не е исполнет уште при првото испитување и притоа, *е можно да не се изврши ниту еден циклус.*

Графичкото претставување на алгоритамската контролна структура **додека–извршшувај** е дадено на ***слика 2. .22***.

За реализација на алгоритамската контролна структура **додека–извршшувај**, во C++ се користи контролната наредба while. Нејзиниот запис е даден на ***слика 2. .2*** .

```
while(uslov) {
        naredba A;
        naredba B;
        ...
        naredba H;
}
```

Слика 2. .2



Слика 2. .22

За контролната наредба while важи дека ако *uslov* не е исполнет уште пред првиот циклус, тогаш наредбата while не се извршува ниту еднаш.

Примери

Пример 2.3.9

Со користење на алгоритамската контролна структура **додека–извршувај**, алгоритамот за задачата за наоѓање на збирот на првите 10 природни броеви од претходниот поднаслов, ќе биде како на ***слика 2. .24***, а програмата напишана со наредбата while е дадена на ***слика 2. .25.***

алгоритам *ЗбирДо10*
почеток
zbir ← 0;
broj ← 1;
додека broj ≤ 10 **извршувај**
zbir ← zbir + broj;
broj ← broj + 1;
крај_додека {broj ≤ 10}
печати zbir;
крај {*ЗбирДо10*}

Слика 2. .24

```
1   #include <iostream>
2   using namespace std;
3
4   int main() {    // Zbir na prvite
5
6       int broj, zbir;
7       zbir = 0;
8       broj = 1;
9       while( broj <= 10 ) {
10          zbir = zbir + broj;
11          broj = broj + 1;
12      }
13      cout << "Zbirot na prvite 10 prirodni broevi e " << zbir << endl;
14
15      cout << endl;
16      system( "Color 17" );
17      system( "pause" );
18      return 0;
19  }
```

Слика 2. .25

Еден излез од извршување на програмата е:

```
Zbirot na prvite 10 prirodni broevi e 55

Press any key to continue . . .
```

Пример 2.3.10

Да се пресмета колку пари сме потрошиле на пазар.

Бидејќи е потребно да знаеме колку производи сме купиле, за секој производ ќе го внесиме износот на сумата што сме го платиле. За крај на внесувањето, треба да поставиме некоја контрола. Знаеме дека за секој производ сме платиле одредена сума, која е позитивен број. Затоа, на крајот од внесувањето ќе внесеме некој број кој не може да претставува платен износ за некој производ. На пример, за износ ќе внесеме –1, –99 или 9 999 (мала е веројатноста да сме го платиле баш овој износ за некој производ) и слично. Ваквото контролирање на повторувањето со некоја однапред позната вредност се нарекува *повторување контролирано со стражар* (англ. Sentinel-Controlled Repetition).

Програмата изразена со наредбата while, е дадена на ***слика 2. .26.***

```
1   #include <iostream>
2   using namespace std;
3
4   int main() {
5       // Pazar (so while)
6
7       double iznosNaSmetka, vkupnoPotroseno = 0;
8       cout << "Za kraj, vnesete iznos na smetkata -99" << endl;
9       cout << "\nVnesete go iznosot na prvata smetka: ";
10      cin >> iznosNaSmetka;
11      while( iznosNaSmetka != -99 ) {
12          vkupnoPotroseno += iznosNaSmetka;
13          cout << "Vnesete go iznosot na slednata smetka: ";
14          cin >> iznosNaSmetka;
15      }
16      cout << "\nVkupno e potroseno " << vkupnoPotroseno << " denari." << endl;
17
18      cout << endl;
19      system( "Color 17" );
20      system( "pause" );
21      return 0;
22  }
```

Слика 2. .26

По извршување на програмата, се добива:

```
Za kraj, vnesete iznos na smetkata -99

Vnesete go iznosot na prvata smetka: 123.50
Vnesete go iznosot na slednata smetka: 100
Vnesete go iznosot na slednata smetka: 565.5
Vnesete go iznosot na slednata smetka: -99

Vkupno e potroseno 789 denari.

Press any key to continue . . .
```

Вежби

Вежба 2.3.4

Да се најде грешката во следниов програмски сегмент:

```
float x = 5, suma = 0;
while(x >= 0)
	suma = suma + x;
```

Вежба 2.3.5

Да се утврди што ќе се отпечати по извршување на следниов програмски сегмент:

```
int i = 0;
while(i < 5) {
	i = i + 2;
	cout << i << "\n";
}
```

Вежба 2.3.7

Што печати следнава програма?

```
#include <iostream>
using namespace std;

int main() {
	int n, s = 0, i = 1;
	double x;
	cout << "Vnesete priroden broj: "; cin >> n;
	while(i <= n) {
		cout << "Vnesete decimalen broj: "; cin >> x;
		s += x;
		i++;
	}
	cout << "s = " << s << endl;
	return 0;
}
```

Решени задачи

Задача 2.3.3
Да се утврди колку цифри има некој природен број.

Објаснува е Нека е даден бројот n = 174.
Ако n (= 174) го поделиме со 10, количникот ќе биде 17, а остатокот 4.
Количникот ќе му го доделиме на n (n = 17).
Ако n (= 17) го поделиме со 10, количникот ќе биде 1, а остатокот 7.
Количникот ќе му го доделиме на n (n = 1).
Ако n (= 1) го поделиме со 10, количникот ќе биде 0, а остатокот 1.
Количникот ќе му го доделиме на n (n = 0).
Гледаме дека може да формираме циклус со две дејства:

- Делење на n со 10.
- Доделување на добиениот количник на n.

Повторувањето прекинува кога остатокот од делењето ќе биде 0 (n = 0).
Со оваа постапка се губи почетната (внесена) вредност на n. Ако е потребно и по извршување на алгоритамот да се користи почетната вредност на n, потребно е по внесувањето да се запамти во друга променлива.

```
#include <iostream>
using namespace std;

int main() {
    // Broj cifri na cel broj (so while)

    int celBroj, brojCifri, kolicnik;
    cout << "Vnesete cel broj, n=";
    cin >> celBroj;
    cout << "Celiot broj " << celBroj << " ima ";
    kolicnik = celBroj / 10;
    brojCifri = 1;
    while( kolicnik != 0 ) {
        celBroj = kolicnik;
        kolicnik = celBroj / 10;
        brojCifri++;
    }
    cout << brojCifri << " cifri." << endl;

    cout << endl;
    system( "Color 17" );
    system( "pause" );
    return 0;
}
```

Слика 2. .2

По извршувањето на програмата, се добива:

```
Vnesete priroden broj, n=2123456789
Prirodniot broj 2123456789 ima 10 cifri.

Press any key to continue . . .
```

Задача 2.3.4

Да се најде најголемиот заеднички делител за два природни броја.

Објаснува е Најголемиот заеднички делител (НЗД) за два природни броја може да биде помалиот од нив ако се броевите деливи еден со друг или првиот број помал од помалиот, со кој се деливи и двата броја. Затоа, прво претпоставуваме дека НЗД е помалиот од нив, а ако не е, тогаш го намалуваме НЗД за 1 во секој циклус додека не најдеме број со кој се деливи двата дадени броја.

```
1   #include <iostream>
2   using namespace std;
3
4   int main() {
5       // NZD na dva broja (so while)
6
7       int a, b, nzd;
8       cout << "Vnesete dva prirodni broja" << endl;
9       cout << "a = "; cin >> a;
10      cout << "b = "; cin >> b;
11      if( a < b )
12          nzd = a;
13      else
14          nzd = b;
15      while( ( a % nzd != 0 ) || ( b % nzd != 0 ) )
16          nzd--;
17      cout << "Najgolem zaednicki delitel za broevite "
18          << a << " i " << b << " e " << nzd << endl;
19
20      cout << endl;
21      system( "Color 17" );
22      system( "pause" );
23      return 0;
24  }
```

Слика 2. .28

Пример на излез по извршување на програмата е:

```
Vnesete dva prirodni broja
a = 1248
b = 2364
Najgolem zaednicki delitel za broevite 1248 i 2364 e 12

Press any key to continue . . .
```

Задача 2.3.5
Да се погоди природниот број што го замислил компјутерот.

На почетокот од програмата се генерира случаен број со функцијата rand(). (Оваа функција е обработена во поднасловот **Библиотечни функции** од потточката **2.5 Функции**). Со rand() се генерира случаен број од 0 до 32 767. Со rand() % 100 множеството случајни броеви се сведува на интервалот од 0 до 99, а со 1 + rand() % 100, се сведува на множеството од 1 до 100.

```
1   #include <iostream>
2   using namespace std;
3
4   int main() {    // Zamislen broj (so while)
5
6       int broj, zamislenBroj, obidi;
7       zamislenBroj = 1 + rand() % 100;
8       cout << "Pogodete koj broj go zamisliv megju 1 i 100: ";
9       cin >> broj;
10      obidi = 1;
11      while( broj != zamislenBroj ) {
12          if( broj > zamislenBroj )
13              cout << "Zamisleniot broj e pomal od " << broj;
14          else
15              cout << "Zamisleniot broj e pogolem od " << broj;
16          cout << "\nPogoduvajte pak: ";
17          cin >> broj;
18          obidi++;
19      }
20      cout << "Bravo! Pogodivte vo " << obidi << "-ot obid." << endl;
21
22      cout << endl;
23      system( "Color 17" );
24      system( "pause" );
25      return 0;
26  }
```

```
Pogodete koj broj go zamisliv megju 1 i 100: 50
Zamisleniot broj e pomal od 50
Pogoduvajte pak: 25
Zamisleniot broj e pogolem od 25
Pogoduvajte pak: 37
Zamisleniot broj e pomal od 37
Pogoduvajte pak: 30
Zamisleniot broj e pomal od 30
Pogoduvajte pak: 26
Zamisleniot broj e pogolem od 26
Pogoduvajte pak: 28
Bravo! Pogodivte vo 6-ot obid.

Press any key to continue . . .
```

Задачи за вежбање

За следниве задачи да се напишат програми со користење на контролната наредба while:

1. Да се пресмета производот на целите броеви внесени преку тастатурата додека не се внесе број 0.
2. Да се пресмета збирот на броевите од a до b со чекор c = (b – a) / n, т. е. вредноста на изразот a + (a + c) + (a + 2c) + (a + 3c) +...
3. Да се пресмета збирот на цифрите на природниот број n.
4. Да се најде најмалиот заеднички содржател (НЗС) за два природни броја.
5. Да се провери дали некој природен број е палиндром. (Палиндром е оној природен број кој има иста вредност ако се чита одназад, како и однапред. На пример, бројот 123 454 321 е палиндром).
6. Да се провери дали природниот број n е прост или не е. (Прости броеви се оние кои се деливи само со 1 и со самите себеси).
7. Да се најдат сите пријателски броеви до 10 000. (Два броја се пријателски ако збирот на делителите на едниот е еднаков на другиот и збирот на делителите на другиот е еднаков на првиот. Во делители на бројот не спаѓа самиот број).
8. Да се претвори целиот ненегативен декаден број n во бинарен број.
9. Да се претвори целиот ненегативен бинарен број b во декаден број.
10. Да се пресмета вредноста на бројот e = 2.718281 со дадена точност ε (се чита „епсилон“), преку изразот $e = 1 + \frac{1}{1!} + \frac{1}{2!} + \frac{1}{3!} + \frac{1}{4!} + \ldots$ Точноста ε се постигнува кога последниот собирок (кој не се додава во збирот) $\frac{1}{k!} < \varepsilon$. Бројот ε е многу мала вредност, на пример, ε = 0.001, ε = 0.0000000001 итн.
11. Да се пресмета вредноста на бројот π = 3.14159 со дадена точност ε, преку изразот $\pi = 4 - \frac{4}{3} + \frac{4}{5} - \frac{4}{7} + \frac{4}{9} - \ldots$, односно додека $\frac{4}{2k+1} < \varepsilon$.
12. Да се отпечатат следниве форми:

```
@@@@@@@@@@
@@@@@@@@@
@@@@@@@
@@@@@
@@@
@
```

```
&&&&&&&&&&
&&&&&&&&
&&&&&&
&&&&
&&
&&
&&&&
&&&&&&
&&&&&&&&
&&&&&&&&&&
```

```
$
###
$$$$$
#######
$$$$$$$$$
###########
$$$$$$$$$
#######
$$$$$
###
$
```

Алгоритамска контролна структура за повторување со излез на крајот од циклусот извршувај–додека и контролна наредба do-while

Во задачата за наоѓање на збирот на првите 10 природни броеви од претходните поднаслови, рековме дека се повторуваат само две дејства:

- Додавање broj на zbir.
- Зголемување на broj за 1.

Можеме да кажеме дека циклусите со овие две дејства се **извршуваат додека** broj има помала или еднаква вредност на 10. Затоа, оваа алгоритамска контролна структура се нарекува **извршувај–додека**. Тоа можеме да го претставиме како на ***слика 2. . 0***.

```
zbir ← 0;
broj ← 1;
извршувај
        – додади го broj на zbir;
        – зголеми го broj за 1;
додека broj ≤ 10;
крај_ додека {broj ≤ 10}
печати zbir;
```

Слика 2. . 0

Во оваа алгоритамска контролна структура, условот дали следниот циклус ќе се изврши или не, се испитува на крајот по извршување на секој циклус. Ова значи дека мора да се изврши барем еден циклус.

Текстуалниот запис на алгоритамската контролна структура **извршувај–додека** и графичкиот приказ се дадени на ***слика 2. . 1*** и ***слика 2. . 2***.

За реализација на алгоритамската контролна структура **извршувај–додека**, во C++ се користи контролната наредба do-while, ***слика 2. .*** .

```
извршувај
        чекор А;
        чекор Б;
        ...
        чекор М;
додека услов;
крај_извршувај {услов}
```

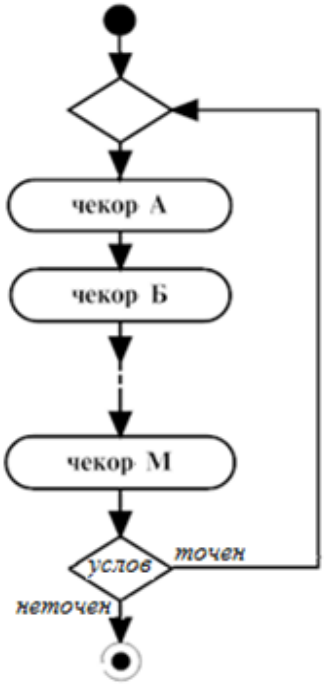
Слика 2. . 1

Наредбата do-while се користи кога некоја група од наредби треба да се извршат барем еднаш, а нивното повторно извршување зависи од некој услов.

Извршувањето на циклусите се повторува додека е исполнет *uslov*. Кога *uslov* нема да биде исполнет, повторувањето завршува и дејството продолжува со следната наредба.

```
do{
        naredba A;
        naredba B;
        ...
        naredba T;
}while(uslov);
```

Слика 2. .



Слика 2. . 2

Примери

Пример 2.3.11

Алгоритамот за задачата за збирот на првите 10 природни броеви, изразен текстуално со алгоритамската контролна структура **извршувај–додека** е даден на ***слика 2. . 4***, а програмата напишана со контролната наредба do-while е дадена на ***слика 2. . 5***.

алгоритам *Збир*
почеток
zbir ← 0;
broj ← 1;
извршувај
zbi ← zbir + broj;
broj ← broj + 1;
додека broj ≤ 10;
крај_извршувај {broj ≤ 10}
печати zbir;
крај {*Збир*}

Слика 2. . 4

```
1   #include <iostream>
2   using namespace std;
3
4   int main() {
5       // Zbir na prvite 10 prirodni broevi (so do-while)
6
7       int broj, zbir;
8       zbir = 0;
9       broj = 1;
10      do {
11          zbir = zbir + broj;
12          broj = broj + 1;
13      } while( broj <= 10 );
14      cout << "Zbirot na prvite 10 prirodni broevi e " << zbir << endl;
15
16      cout << endl;
17      system( "Color 17" );
18      system( "pause" );
19      return 0;
20  }
```

Слика 2. . 5

Еден излез по извршување на програмата е:

```
Zbirot na prvite 10 prirodni broevi e 55

Press any key to continue . . .
```

Примери 2.3.12

Да се пресмета колку пари сме потрошиле на пазар.

Објаснува е: Програмата од **Пример 2.3.10** поприродно е да ја напишеме со контролната наредба do-while бидејќи пресметка за трошокот на пазар ќе правиме и ако сме купиле барем еден продукт, т. е. ако имаме барем една сметка. Значи, мора да се изврши барем еден циклус.

Програмата е дадена на ***слика 2. . 6.***

```
1   #include <iostream>
2   using namespace std;
3
4   int main() {
5       // Pazar (so do-while)
6
7       double iznosNaSmetka, vkupnoPotroseno = 0;
8       cout << "Za kraj, vnesete iznos na smetkata -99" << endl;
9       cout << "Vnesete go iznosot na prvata smetka: ";
10      cin >> iznosNaSmetka;
11      do {
12          vkupnoPotroseno += iznosNaSmetka;
13          cout << "Vnesete ja slednata smetka: ";
14          cin >> iznosNaSmetka;
15      } while( iznosNaSmetka != -99 );
16      cout << "\nVkupno e potroseno " << vkupnoPotroseno << " denari." << endl;
17
18      cout << endl;
19      system( "Color 17" );
20      system( "pause" );
21      return 0;
22  }
```

Слика 2. . 6

Еден излез од извршување на програмата е:

```
Za kraj, vnesete iznos na smetkata -99
Vnesete go iznosot na prvata smetka: 12
Vnesete ja slednata smetka: 34.5
Vnesete ja slednata smetka: 56.789
Vnesete ja slednata smetka: -99

Vkupno e potroseno 103.289 denari.

Press any key to continue . . .
```

Вежби

Вежба 2.3.8

Да се утврди што работи следниов програмски сегмент:

```
int x = 0, i = 0;
do {
	x++;
	i += ++x;
} while(x < 10);
```

Вежба 2.3.9

Да се утврди што ќе се отпечати по извршување на следниов програмски сегмент:

```
int i = 0;
do {
	i += i++;
	cout << i << ", ";
} while(i < 10);
```

Вежба 2.3.10

Да се утврди што ќе се отпечати по извршување на следниов програмски сегмент:

```
int i = 12345;
do {
	cout << i % 10 << '\n';
	i /= 10;
} while(i > 0);
```

Решени задачи

Задача 2.3.6

Да се утврди колку цифри има некој природен број.

Објаснува е: Задачата е иста со **Задача 2.3.3**. Меѓутоа, поприродно е да ја изразиме со контролната наредба do-while бидејќи и да е бројот едноцифрен, мора да извршиме барем едно делење со 10 за да видиме дали остатокот е 0.

Програмата, напишана со наредбата do-while, е дадена на ***слика 2. . .***

Еден излез од извршување на програмата е:

```
Vnesete priroden broj, n=1234567890
Prirodniot broj 1234567890 ima 10 cifri.

Press any key to continue . . .
```

```
#include <iostream>
using namespace std;

int main() {
    // Broj cifri na priroden broj (so do-while)

    int prirodenBroj, brojCifri, kolicnik;
    cout << "Vnesete priroden broj, n=";
    cin >> prirodenBroj;
    cout << "Prirodniot broj " << prirodenBroj << " ima ";
    brojCifri = 0;
    do {
        kolicnik = prirodenBroj / 10;
        brojCifri++;
        prirodenBroj = kolicnik;
    } while( kolicnik != 0 );
    cout << brojCifri << " cifri." << endl;

    cout << endl;
    system( "Color 17" );
    system( "pause" );
    return 0;
}
```

Слика 2. .

Задача 2.3.7

Да се пресмета вредноста на Ојлеровиот[5] број (Leonhard Euler, швајцарски математичар) e = 2.718281... преку изразот

$$e \approx 1+\frac{1}{1!}+\frac{1}{2!}+\frac{1}{3!}+\frac{1}{4!}+\ldots+\frac{1}{n!}$$

со точност ε.

Објаснување: Рековме дека производот 1 · 2 · 3 ·. . . · n се означува со n! и се нарекува „n факториел“. По дефиниција, 0! = 1 и 1! = 1, а факториел за поголеми броеви од 1 се пресметува по формулата n! = (n – 1)! · n.

Така, 2! = 1 · 2, 3! = 2! · 3 = 1 · 2 · 3 итн.

Оваа особина на факториелите можеме да ја искористиме за решавање на задачата. Имено, $\frac{1}{2!}=\frac{1}{1!}\cdot\frac{1}{2}$, $\frac{1}{3!}=\frac{1}{2!}\cdot\frac{1}{3}$, $\frac{1}{4!}=\frac{1}{3!}\cdot\frac{1}{4}$, итн., т. е. $\frac{1}{n!}=\frac{1}{(n-1)!}\cdot\frac{1}{n}$.

Ова значи дека секој нареден собирок во збирот (на пример, к-тиот) се добива кога претходниот ((к – 1)-виот) собирок ќе го помножиме со $\frac{1}{k}$.

[5] Некои го нарекуваат Неперов број (*John Napier, XVII век, шкотски тематичар*).

Точноста ε ќе ја користиме за крајот на повторувањето на циклусите кога собирокот што треба да се додаде на збирот ќе биде помал од зададената точност ($\frac{1}{n!} < \varepsilon$). Притоа, не се знае однапред колку собироци треба да се соберат за да се постигне зададената точност.

Програмата е дадена на ***слика 2. . 8***.

Од точната вредност во програмата, се гледа дека за точност $\varepsilon = 10^{-50}$ се добива вредност со 15 точни децимали.

```
1   #include <iostream>
2   #include <iomanip>
3   using namespace std;
4
5   int main() {
6       // Presmetuvanjena brojot e=2.71828182845904523536028747135266249775724709369999...
7       // so tocnost epsilon. (so do-while)
8
9       int k;
10      double eBroj, epsilon, sobirok;
11      cout << "Vnesete tocnost, epsilon = ";
12      cin >> epsilon;
13      eBroj = 0;
14      k = 0;
15      sobirok = 1;
16      do {
17          eBroj += sobirok;
18          ++k;
19          sobirok /= k;
20      } while( sobirok >= epsilon );
21      cout << fixed << setprecision( 50 );
22      cout << "Brojot e=2.71 so tocnost " << epsilon << "\ne " << eBroj << endl;
23
24      cout << endl;
25      system( "Color 17" );
26      system( "pause" );
27      return 0;
28  }
```

Слика 2. . 8

Еден излез од извршување на програмата е:

```
Vnesete tocnost, epsilon = 1e-50
Brojot e=2.71 so tocnost 0.00000000000000000000000000000000000000000000000001
e 2.71828182845904553488480814849026501178741455078125

Press any key to continue . . .
```

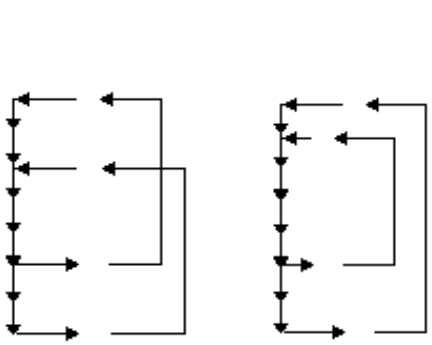
Задачи за вежбање

Да се провери кои задачи од **Задачи за вежбање** од поднасловот **Алгоритамска контролна структура за повторување со излез на почетокот од циклусот и контролна наредба while** може да се напишат со користење на контролната наредба do-while.

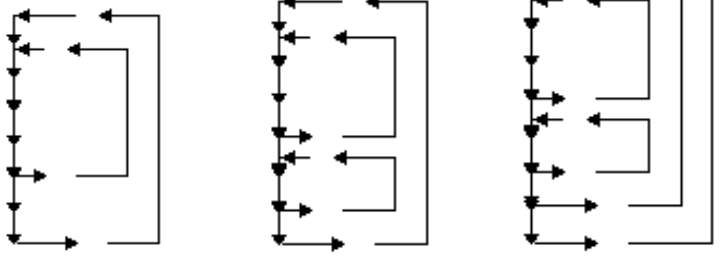
Вгнездување на контролните наредби за повторување

Во некои програми потребно е да се употребат повеќе контролни наредби за повторување. Притоа, некоја контролна наредба може да се наоѓа во друга. Тоа се нарекува **вгнездување** (англ. nesting).

При вгнездување на контролните наредби за повторување, сите наредби на едната контролна наредба мора да се наоѓаат во другата, односно не смее да има сечење (преклопување) на контролните наредби, како на ***слика 2. . 9***. На ***слика 2. .40 а, б*** и ***в*** се дадени дозволени начини на вгнездување на контролните наредби.



Слика 2. . 9



а *б* *в*

Слика 2. .40

Примери

Пример 2.3.13

Да се напише програма за таблицата за множење до 10 за броевите од 1 до 10.

Програмата е дадена на ***слика 2. .41***.

```cpp
#include <iostream>
#include <iomanip>
using namespace std;

int main() {
    // Tablica mnozenje do 10, za broevite od 1 do 10. (so for)

    for( int i = 1; i <= 10; i++ ) {
        cout << "Mnozenje so " << i << endl << endl;
        for( int j = 1; j <= 10; j++ ) {
            int ipoj = i * j;
            cout << setw( 2 ) << i << " x " << setw( 2 ) << j << " = "
                << setw( 3 ) << ipoj << endl;
        }
        cout << endl;
    }

    cout << endl;
    system( "Color 17" );
    system( "pause" );
    return 0;
}
```

Слика 2. .41

```
Mnozenje so 1

 1 x  1 =   1
 1 x  2 =   2
 1 x  3 =   3
 1 x  4 =   4
 1 x  5 =   5
 1 x  6 =   6
 1 x  7 =   7
 1 x  8 =   8
 1 x  9 =   9
 1 x 10 =  10

Mnozenje so 2

 2 x  1 =   2
 2 x  2 =   4
 2 x  3 =   6
 2 x  4 =   8
 2 x  5 =  10
 2 x  6 =  12
 2 x  7 =  14
 2 x  8 =  16
```

…

Пример 2.3.14

Да се најдат сите природни броеви помали од n чиј збир на кубовите на цифрите е еднаков на бројот.

```
#include <iostream>
using namespace std;

int main() {
    // Prirodni broevi ednakvi na kubovite na svoite cifri.
    system( "Color 17" );
    long long n, cifra, broj, zbir, brojN;
    bool imaNema = false;
    cout << "Do koj broj, n = ";    cin >> n;
    cout << "\n Prirodni broevi pomali od " << n;
    cout << "\n i ednakvi na zbirot na kubovite na svoite cifri \n se: ";
    for( broj = 1; broj <= n; broj++ ) {
        zbir = 0;
        brojN = broj;
        do {
            cifra = brojN % 10;
            zbir += ( int ) pow( cifra, 3 );
            brojN /= 10;
        } while( brojN > 0 );
        if( zbir == broj ) {
            cout << broj << ", ";
            imaNema = true;
        }
    }
    if( !imaNema ) {
        cout << "\n Nema takvi broevi do " << n << endl;
    }

    cout << endl << endl;
    system( "Color 17" );
    system( "pause" );
    return 0;
}
```

Слика 2. .42

Еден излез по извршување на програмата е:

```
Do koj broj, n = 1000

 Prirodni broevi pomali od 1000
 i ednakvi na zbirot na kubovite na svoite cifri
 se: 1, 153, 370, 371, 407,

Press any key to continue . . .
```

Прашања за проверка на знаењето

1. Кога се користи алгоритамската контролна структура за повторување?
2. Како се нарекува едно извршување на чекорите во алгоритамската контролна структура за повторување?
3. Како се делат алгоритамските контролни структури за повторување, според начинот на прекинување на повторувањето?
4. Кои алгоритамски контролни структури за повторување со броење на циклусите ги знаете?
5. Претставете ја текстуално алгоритамската контролната структура **за–зголемувај–до.**
6. Од каков тип се променливите brojac, pocetna и krajna во алгоритамската контролната структура **за-зголемувај-до**?
7. Колку циклуси ќе се извршат во алгоритамската контролната структура **за–зголемувај–до** ако променливата pocetna има поголема вредност од променливата krajna?
8. Со која контролна наредба во C++ се реализираат алгоритамските контролни структури за повторување со броење на циклусите?
9. Кои делови ги има контролната наредба for? Што се прави во секој дел?
10. Кое е дејството на операторите за инкрементирање и за декрементирање?
11. Што значи префиксно, а што постфиксно инкрементирање на променлива? Наведете пример.
12. За какви множества податоци може да се користат операторите за инкрементирање и за декрементирање?
13. Дали може бројачот во контролната наредба for да се користи во нејзиното тело?
14. Колку променливи може да се иницијализираат во контролната наредба for?
15. Дали бројачот во контролната наредба for може да се иницијализира надвор од неа?
16. Дали може делот за услов во контролната наредба for да биде празен?
17. Што работи следниов програмски сегмент?

```
for(x = 1, i = 0; x <= 10; x++)
   i += x;
```

18. Што ќе се отпечати при извршување на следниов програмски сегмент?

```
for(int i = 0, j = 0; i < 5; i++, j--)
   cout << i + j << endl;
```

19. Што ќе се отпечати при извршување на следниов програмски сегмент?

```
for(int i = 1; i <= 5; i++)
   for(int j = 1; j <= 5; j++)
          for(int k = 1; k <= 5; k++)
                 cout << i + j + k << endl;
```

20. Што ќе се отпечати при извршување на следниов програмски сегмент?

```
for(int i = 1; i <= 5; i++) {
   for(int j = 1; j <= i; j++) {
          for(int k = 1; k <= j; k++)
                 cout << '?';
          cout << endl;
   }
   cout << endl;
}
```

21. Претставете ја текстуално и графички алгоритамската контролна структура **додека–извршувај**.
22. Со која контролна наредба во C++ се реализира алгоритамската контролна структура **додека–извршувај**?
23. Дали може со контролната наредба while да не се изврши ниту еден циклус?
24. Напишете ја синтаксата на контролната наредба while.
25. Со која контролна наредба во C++ се реализира алгоритамската контролна структура **извршувај–додека**?
26. Напишете ја синтаксата и објаснете како работи контролната наредба do-while.
27. Која е разликата помеѓу контролните наредби while и do-while?
28. Што ќе се отпечати како резултат на извршување на следниов програмски сегмент?

```
int i = 0;
while(i < 5) {
   i = i + 2;
   cout << i << endl;
}
```

29. Што работи следниов програмски сегмент?

```
int i, j;
i = 0;
while(i < 8) {
   j = 0;
   while(j < 8) {
          cout << setw(5) << (i + j) % 8;
          ++j;
   }
   cout << endl;
   ++i;
}
```

30. Што работи следниов програмски сегмент?

```
int x, i;
x = 0; i = 0;
do {
   x++;
   i += x;
} while(x < 10);
```

31. Што ќе се отпечати по извршување на следниов програмски сегмент?

```
int i = 0;
do {
   ++ ++i;
   cout << i << endl;
} while(i < 10);
```

32. Што ќе се отпечати по извршување на следниов програмски сегмент?

```
for(int i = 1; i <= 5; i++)
   switch(i) {
   case 2: cout << "den, "; break;
   case 4: cout << "ste? "; break;
   case 3: cout << "kako"; break;
   case 1: cout << "Dobar"; break;
   }
```

33. Што значи вгнездување на контролни наредби? Нацртајте шеми на дозволени вгнездувања на контролните наредби за повторување.

Задачи

1. Да се генерира следниов триаголник од n линии, а во секоја линија по к ѕвездички, к = 1, 2... n.

```
*
* *
* * *
* * * *
...
* * * * * * * * *
```

2. Да се генерира следниов триаголник:

```
1
2 4
3 6 9
4 8 12 16
...
n ...
```

3. Да се отпечати следниов цртеж:

```
>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>
<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<
>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>
<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<
>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>
```

4. Да се генерираат броевите на Fibonacci: 1, 1, 2, 3, 5, 8, 13, 21, 34 итн. помали од бројот n.
5. Да се внесе природен број n и да се отпечати цифра по цифра со зборови. На пример, ако се внесе бројот 231, да се отпечати dva-tri-eden.
6. Да се внесе природен број n и да се отпечати со зборови. На пример, ако се внесе бројот 12 345, да се отпечати „dvanaeset iljadi trista cetirieset i pet“.

7. Да се пресмета збирот $\frac{1}{1\cdot 2}+\frac{1}{2\cdot 3}+\frac{1}{3\cdot 4}+\cdots$ со точност ε, $\left(\frac{1}{k(k+1)}<\varepsilon\right)$.
8. Да се внесуваат броеви преку тастатурата и да се најдат најголемиот и најмалиот од внесените. За крај на внесувањето, да се внесе бројот 999 999.
9. Да се скрати дропката $\frac{a}{b}$, a и b се природни броеви.
10. Да се отпечатат сите броеви помали од n, кои се деливи со збирот на своите цифри.
11. Да се најде најголемиот непарен делител на природниот број n.
12. Да се изостави к-тата цифра на природниот број n, броејќи ги цифрите од цифрата на единиците.

2.4 Алгоритамски контролни структури за скок и контролни наредби за скок

Во програмските јазици постојат три вида алгоритамски контролни структури за скок, и тоа:

- *Скок на крајот на циклусот* – **продолжи**.
- *Скок на крајот на контролната структура* – **прекин**.
- *Скок на крајот на алгоритамот* – **излез**.
- *Скок на произволно место* – **скок**.

Соодветни контролни наредби за овие алгоритамски контролни структури во C++ се:

- continue за **продолжи**,
- break за **прекин**,
- exit() за **излез**,
- goto за **скок**.

Наредбите ќе ги објасниме на примери.

Контролна наредба continue

Оваа контролна наредба се користи за безусловен скок на крајот на циклусот. Тоа се прави кога во некоја наредба од циклусот, пред тој да заврши, е исполнет одреден услов и не е потребно да се извршат останатите наредби од циклусот. Со неа се излегува од тековниот циклус и дејството *продолжува* со следниот циклус.

Контролната наредба **continue** се користи за излез од тековниот циклус во контролните наредби за повторување while, do-while и for.

Пример 2.4.1

Да се пресмета квадратниот корен од внесените броеви. Ако се внесе негативен број, да не се пресметува квадратниот корен од него, туку да се продолжи со следниот внесен број. За крај на внесувањето, да се внесе бројот 999 999.

Програмата е дадена на ***слика 2.4.1***. Ако бројот е негативен, се печати порака дека нема корен од негативен број и се продолжува со следниот циклус, т. е. со читање на следниот број.

```
1   #include <iostream>
2   using namespace std;
3
4   int main() {
5       // Kvadraten koren od realen broj (so continue)
6
7       double x;
8       cout << "Za kraj, vnesete 999999 za broj." << endl;
9       do {
10          cout << "Vnesete realen broj, x=";
11          cin >> x;
12          if( x < 0 ) {
13              cout << "Brojot e negativen i nema kvadraten koren. \n";
14              continue;
15          }
16          cout << "Kvadraten koren od " << x << " e " << sqrt( x ) << endl;
17
18      } while( x != 999999 );
19
20      cout << endl;
21      system( "Color 17" );
22      system( "pause" );
23      return 0;
24  }
```

Слика 2.4.1

Излезот по извршување на програмата е:

```
Za kraj, vnesete 999999 za broj.
Vnesete realen broj, x=1
Kvadraten koren od 1 e 1
Vnesete realen broj, x=2
Kvadraten koren od 2 e 1.41421
Vnesete realen broj, x=-3
Brojot e negativen i nema kvadraten koren.
Vnesete realen broj, x=12345678900
Kvadraten koren od 1.23457e+09 e 35136.4
Vnesete realen broj, x=999999
Kvadraten koren od 999999 e 999.999

Press any key to continue . . .
```

Контролна наредба break

Контролната наредба break се користи во контролните наредби за повторување while, do-while и for[6], како безусловен скок на крајот од наредбата за повторување и прекин на повторувањата. Тоа се прави кога во некоја наредба од циклусот, пред тој да заврши, е исполнет одреден услов и не е потребно да се продолжи со повторување на циклусите, т. е. се *прекинува* повторувањето. Дејството продолжува со следната наредба.

Ако контролната наредба **break** се наоѓа во некоја од вгнездените контролни наредби while, do-while и for, тогаш со неа се излегува само од вгнездената контролна наредба во чие тело се наоѓа.

Оваа контролна наредба најчесто се користи за излез од контролните наредби со бесконечен број циклуси.

***Напомена**:* Бесконечно повторување може да се реализира и со други наредби.

Пример 2.4.2

Во следниов пример, повторувањето ќе прекине само ако го погодиме замисленият број од компјутерот (случајно генериран), инаку ќе биде бесконечно, ***слика 2.4.2***.

Во програмата се користи функцијата rand() за генерирање случаен број, објаснета во **Задача 2.3.5**. За да се генерира различна низа случајни броеви со rand(), се користи функцијата srand(), чиј аргумент е број внесен преку тастатурата. Функцијата srand() е објаснета во поднасловот **Функција за генерирање случајни броеви** од потточката **2.5 Функции**.

Еден излез од извршување на програмата е:

```
Vnesete priroden broj: 12345
Pogodete koj broj go zamisliv pomegju 1 i 10: 1
Pogodete koj broj go zamisliv pomegju 1 i 10: 2
Pogodete koj broj go zamisliv pomegju 1 i 10: 3
Pogodete koj broj go zamisliv pomegju 1 i 10: 4
Pogodete koj broj go zamisliv pomegju 1 i 10: 5
Bravo! Go pogodivte zamisleniot broj 5 vo 5-ot obid.

Press any key to continue . . .
```

[6] Контролната наредба break се користи и во контролната наредба switch, но таму е задолжителна според дефиницијата на наредбата switch.

```
1   #include <iostream>
2   using namespace std;
3
4   int main() {    // Pogoduvanje zamislen broj megju 1 i 10 (so break)
5
6       int broj, zamislenBroj;
7       cout << "Vnesete priroden broj: "; cin >> broj;
8       srand( broj );
9       zamislenBroj = 1 + rand() % 10;
10      for( int obidi = 1; ; obidi++ ) {
11          cout << "Pogodete koj broj go zamisliv pomegju 1 i 10: ";
12          cin >> broj;
13          if( broj == zamislenBroj ) {
14              cout << "Bravo! Go pogodivte zamisleniot broj "
15                  << zamislenBroj << " vo " << obidi << "-ot obid." << endl;
16              break;
17          }
18      }
19
20      cout << endl;
21      system( "Color 17" );
22      system( "pause" );
23      return 0;
24  }
```

Слика 2.4.2

Контролна наредба exit()

Понекогаш е потребно програмата принудно да заврши, што се постигнува со скок на крајот на програмата. Тоа е потребно кога на некое место во програмата се очекува операција со чие извршување може да дојде до прекин на програмата и до нејзино нерегуларно завршување. На пример, ако постои можност при некои пресметувања поткорената големина на квадратниот корен да добие негативна вредност, не треба да се дозволи пресметување на квадратниот корен бидејќи ќе се јави грешка и програмата ќе прекине, туку треба регуларно да се излезе од програмата, со скок на нејзиниот крај.

Оваа контролна наредба често се користи при детектирање на грешки кои може да доведат до погрешни резултати или до прекин на апликацијата, како што е случај да се чита од датотека која не постои.

За скок на крајот на програмата, во C++ се користи функцијата **exit**(), која има еден аргумент и тоа константите EXIT_SUCCESS и EXIT_FAILURE. Ако функцијата се повика со првиот аргумент, значи дека програмата завршила успешно, а ако се повика со вториот аргумент, значи дека програмата завршила неуспешно, т. е. се случила грешка.

exit(EXIT_SUCCESS) – за регуларно (успешно) завршување на програмата, или

exit(EXIT_FAILURE) – за нерегуларно (неуспешно) завршување на програмата, т. е. константата EXIT_FAILURE покажува дека се случила грешка при извршувањето.

Пример 2.4.3
Да се провери дали постои датотека со име Ucenici**.**

Објаснува е Во програмата се користи функцијата compare() за споредување на два стринга. Функцијата е објаснета во потточката **3.5 Стрингови**.

```
1   #include <iostream>
2   #include <string>
3   using namespace std;
4
5   int main() {
6       // Izlez od programata (so exit() )
7       string imeDatoteka;
8       while( true ) {
9           cout << "Vnesete ime na datotekata: ";
10          cin >> imeDatoteka;
11          if( imeDatoteka.compare( "Ucenici" ) == 0 ) {
12              cout << "Postoi datoteka so ime " << imeDatoteka;
13              cout << ", programata moze da prodolzi." << endl;
14              system( "pause" );
15              exit( EXIT_SUCCESS );
16          }
17          else {
18              cout << "Nema datoteka so ime " << imeDatoteka;
19              cout << ", programata zavrsuva." << endl;
20              system( "pause" );
21              exit( EXIT_FAILURE );
22          }
23      }
24
25      cout << endl;
26      system( "Color 17" );
27      system( "pause" );
28      return 0;
29  }
```

Слика 2.4.

Пример за излез од програмата е:

```
Vnesete ime na datotekata: ucenici
Nema datoteka so ime ucenici, programata zavrsuva.
Press any key to continue . . .
```

Контролна наредба goto

Контролната наредба за скок на произволно место goto се користи за скок од кое било место на кое било место во програмата. Притоа, местото на кое се скока мора да биде означено. Ознаката може да биде бројна или текстуална. Таа се запишува како идентификатор и задолжително на крајот се стават две точки.

Ознаката може да биде пред која било наредба во програмата.

Оваа контролна наредба се користела за неструктурирано програмирање. Нејзин недостаток е што го менува текот на извршување на програмата и затоа, е тешко таа да се анализира и разбере. Затоа, оваа контролна наредба не се користи во структурираното програмирање.

Пример 2.4.4

Контролната наредба goto е илустрирана со програмата за пресметување на квадратниот корен од број на **Пример 2.4.1**. Програмата го пресметува квадратниот корен од реален број и прекинува со извршување ако се внесе негативен број, ***слика 2.4.4***.

```
1   #include <iostream>
2   using namespace std;
3
4   int main() {
5       // Kvadraten koren od broj (so goto)
6
7       double x;
8       while( true ) {
9           cout << "Vnesete realen broj, x=";
10          cin >> x;
11          if( x < 0 )
12              goto kraj;
13          cout << "Kvadraten koren od " << x << " e " << sqrt( x ) << endl;
14      }
15  kraj:
16      cout << "Brojot e negativen.Prekinuvam." << endl;
17
18      cout << endl;
19      system( "Color 17" );
20      system( "pause" );
21      return 0;
22  }
```

Слика 2.4.4

Еден излез од извршување на програмата е:

```
Vnesete realen broj, x=2
Kvadraten koren od 2 e 1.41421
Vnesete realen broj, x=-3
Brojot e negativen.Prekinuvam.

Press any key to continue . . .
```

Прашања за проверка на знаењето

1. Кои алгоритамски контролни структури за скок ги знаете?
2. Со кои контролни наредби во C++ се реализираат алгоритамските контролни структури за скок?
3. Која е разликата меѓу контролните наредби continue и break?
4. Со која контролна наредба (функција) се скока на крајот на програмата и каков може да биде нејзиниот аргумент?
5. Објаснете зошто треба да се избегнува контролната наредба goto?

2.5 Функции

Библиотечни функции

Познато е од математиката дека **функциите** (англ. functions) имаат само ***еден излезен резултат***. На пример:

y = 5, y = 4x – 2, y = $2x^2$ – 3x + 1 итн.

Велиме дека вредноста на променливата y зависи од аргументот x и тоа се запишува со y = f(x).

Покрај математичките, постојат и многу други функции кои се користат во разни области. За да се олесни работата на програмерите да не пишуваат секогаш програми за исти функции, на пример за x^n, $\sqrt{x}$, e^x и други, во секој програмски јазик се напишани посебни програми за почесто користените функции. Тие програми се чуваат во т.н. **програмски библиотеки** на јазикот. Програмерите можат да ги користат тие програми, со претходно вклучување на библиотеката на почетокот од програмата, со директивата #include. Така, во сите досегашни програми ние ја вклучувавме библиотеката <iostream> бидејќи користевме програми од неа за влез и излез на податоци.

Практика е програмите од библиотеките да се нарекуваат функции.

Јазикот C++ има т.н. ***стандардна библиотека на C*** (англ. C++ Standard Library).

Функциите од библиотеките на C++ се нарекуваат и **вградени функции** (англ. build-in functions). Нивното име се пишува со мали букви бидејќи имињата се резервирани зборови. На пример, pow(), sqrt, rand() итн.

Стандардната библиотека на C++ содржи повеќе библиотеки кои се чуваат во датотеки, наречени **хедер-датотеки** (англ. header files), кои, пак, содржат групи од слични функции, како:

<iostream>	Функции за влез и за излез.
<iomanip>	Функции за форматирање.
<cstdlib>	Општа библиотека: контрола на програмата, случајни броеви, сортирање, барање итн.
<cmath>	Општи математички функции.
<string>	Функции за работа со стрингови.
<random>	Функции за генерирање случајни броеви.

За да се користи некоја функција од библиотеката на C++ во програмата, на почетокот на програмата треба да се вклучи библиотеката со директивата #include, како што го правевме тоа и досега. Притоа, библиотеката се става во аглести загради.

Во продолжение ќе разгледаме некои почесто користени библиотечни функции.

Математички функции

Да повториме кратко за математичките функции.

На пример, функцијата f(x) = x е линеарна функција, $f(x) = x^2$ е квадратна функција, $f(x) = \sqrt{x}$ е функција за наоѓање на квадратниот корен од x итн. f(x) означува правило по кое се пресметува некоја вредност. На пример, правилото за пресметување на квадратот на некој број е тој да се помножи сам со себе, а се запишува со $f(x) = x \cdot x$ или пократко x^2. Вредноста која ќе се добие за x = 5 е 25 и се означува со некој друг број, на пример со y, т. е. y = 25. За секој број x (наречен **аргумент на функцијата**), се добива вредност на y. Тоа се запишува и со y = f(x) или $y = x^2$.

Во програмирањето, аргументот и вредноста на функцијата се променливи од одреден тип. На пример:

```
int x, y;
x = 5;
y = x * x;
или
y = pow(x, 2);       // Funkcija za stepenuvanje x^2
```

Најчесто користени математички функции во C++ се:

pow(x, y) – Експоненцијална функција x^y.
exp(x) – Експоненцијална функција e^x, e = 2.7182[7].
sqrt(x) – Квардатен корен од x, $\sqrt{x}$.
cbrt(x) – Кубен корен од x, $\sqrt[3]{x}$.
log(x) – Логаритамска функција ln(x), со основа e = 2.7182.
log10(x) – Логаритамска функција log(x), со основа 10.
fabs(x) – Апсолутна вредност од x, |x|.
ceil(x) – Заокружување на x на најмалиот реален број не помал од x.
floor(x) – Заокружување на x на најголемиот реален број не поголем од x.
trunc(x) – Отсекување само на реалниот дел лево од децималната точка.
round(x) – Заокружување на најблискиот реален број до x (без децимали).
fmod(x, y) – Остаток од количникот x / y како реален број.
modf(x, &integralen) – Разделување на децималниот број x на интегрален и на децимален дел. (За знакот & ќе зборуваме подоцна).
sin(x) – Тригонометриска функција sin(x).
cos(x) – Тригонометриска функција cos(x).
tan(x) – Тригонометриска функција tg(x)

Овие функции му овозможуваат на корисникот да изврши низа од математички пресметки. Функциите се користат така што се пишуваат името на функцијата и листата со аргументи ставени во мали загради.

Во претходниот пример напишавме

```
y = pow(x, 2);
```

Десната страна од наредбата означува **повик на функцијата** (англ. function call или function invocation) со име pow и со аргументи x и 2. Со повикување на функцијата, се извршува програмата под име pow(), која се наоѓа во библиотеката <cmath> и резултатот се доделува на променливата y. Често се вели дека **функцијата враќа вредност**.

Со следната наредба за печатење на квадратниот корен од бројот 900.0, се повикува функцијата со име sqrt и со аргумент 900.0:

```
cout << sqrt(900.0);
```

Вредноста што ја враќа функцијата се печати.

Наведените математички функции при секој повик враќаат вредност. Затоа се наречени **функции со повратна вредност** (англ. value-returning functions).

Аргументите на функциите може да бидат константи, променливи или цели изрази. На пример, за c = 12, d = 3.0, g = 4.0 наредбата

```
cout << sqrt(c + d * f);
```

ќе ги пресмета и отпечати квадратниот корен од 12.0 + 3.0 * 4.0 = 25.0, т. е 5.00.

[7] Овој број е основа на природните логаритми, а се нарекува Ојлеров број или Неперов број.

Со следниов пример се илустрира користењето на математичките функции.

Пример 2.5.1

```
#include <iostream>
#include <string>
#include <iomanip>
using namespace std;

int main()
{   // Matematicki funkcii pow(), sqrt(), cbrt(), exp() i log()

    int a, b, c;
    cout << "Vnesete ja prvata vrednost za  x=";
    cin >> a;
    cout << "Vnesete ja vtorata vrednost za x=";
    cin >> b;
    cout << "Vnesete ja tretata vrednost za x=";
    cin >> c;
    cout << setprecision(4) << endl;
    cout << "-----------------------------------------------------------------
    cout << setw(5) << "x" << setw(7) << "preth." << setw(7) << "sledb."
        << setw(7) << "x^2" << setw(15) << "kv.koren(x)" << setw(15)
        << "kub.koren(x)" << setw(12) << "e^x" << setw(10) << "ln(x)" << endl;
    cout << "-----------------------------------------------------------------
    cout << setw(5) << a << setw(4) << a - 1 << setw(7) << a + 1 << setw(10)
        << pow(a, 2) << setw(15) << sqrt(a) << setw(15) << cbrt(a) << setw(12)
        << exp(a) << setw(10) << log(a) << endl;
    cout << setw(5) << b << setw(4) << b - 1 << setw(7) << b + 1 << setw(10)
        << pow(b, 2) << setw(15) << sqrt(b) << setw(15) << cbrt(b) << setw(12)
        << exp(b) << setw(10) << log(b) << endl;
    cout << setw(5) << c << setw(4) << c - 1 << setw(7) << c + 1 << setw(10)
        << pow(c, 2) << setw(15) << sqrt(c) << setw(15) << cbrt(c) << setw(12)
        << exp(c) << setw(10) << log(c) << endl;
    cout << "-----------------------------------------------------------------

    cout << endl;
    system("Color 17");
    system("pause");
    return 0;
}
```

Слика 2.5.1

При извршување на програмата се добива следниот излез:

```
    x preth. sledb.    x^2    kv.koren(x)   kub.koren(x)         e^x      ln(x)
-------------------------------------------------------------------------------
    2    1      3       4          1.414          1.26       7.389     0.6931
    3    2      4       9          1.732         1.442       20.09      1.099
   12   11     13     144          3.464         2.289    1.628e+05      2.485
```

Функција за генерирање на случајни броеви

Во програмирањето често се јавува потреба од генерирање случаен број. Затоа, во C++ постои посебна функција **rand**(), која генерира случаен број помеѓу 0 и 32 767. |[8]

На пример:

```
Slucajni broevi pomegju 0 i 32767:
19169 26500 6334 18467 41
```

Ако сакаме ова множество {0, 1, 2, 3... 32 767} да го сведеме на одреден интервал [0, max] тогаш ќе ја користиме формулата:

```
rand() % (max + 1);
```

На пример, случајни броеви од интервалот [0, 99]:

```
Slucajni broevi pomegju 0 i 99:
64 62 58 78 24
```

За да почнуваат случајните броеви од 1, формулата е

```
1 + rand() % (max + 1);
```

На пример, случајни броеви од претходниот пример во интервалот [1, 100] се:

```
Slucajni broevi pomegju 1 i 100:
62 28 82 46 6
```

Ако сакаме интервалот на случајно генерираните броеви да започнува од одреден број (на пример poceten), тогаш формулата е:

```
poceten + rand() % (max + 1);
```

На пример, интервалот [1, 100] може да биде [51, 150] со:

```
51 + rand() % 100;
```

```
Slucajni broevi pomegju 1 i 150:
87 78 93 146 142
```

При извршување на програмата во која се користи ист израз за генерирање случајни броеви, ќе се генерира иста низа на случајни броеви. Тоа е така бидејќи при секое повикување на функцијата rand() од *стандардната библиотека на C++*, се извршува иста програма. Затоа, генерираните случајни броеви не се навистина случајни, туку се нарекуваат **псевдослучајни броеви** (англ. pseudo-random numbers).

За да се генерира нова низа на случајни броеви при секое повикување на функцијата rand(), се врши **рандомизација** (англ. randomizing) со задавање раз-

[8] Постои константа RAND_MAX чија вредност може да се отпечати.

лична неозначена целобројна вредност како арагумент на функцијата **srand**(). Оваа функција треба да се изврши пред повикување на функцијата rand().

На пример, ако во претходниот пример, пред генерирање на случајните броеви, се изврши функцијата:

```
srand(123);
```

ќе се генерираат следниве случајни броеви:

```
Slucajni broevi pomegju 1 i 150:
114 55 126 104 91
```

Ако, повторно се изврши функцијата, но со друг аргумент:

```
srand(321);
```

ќе се генерираат други случајни броеви:

```
Slucajni broevi pomegju 1 i 150:
116 131 88 69 137
```

Забележуваме дека функцијата srand() се повикува само со своето име, без наредба за доделување. Тоа значи дека оваа функција не враќа вредност. Има и други такви функции кои се нарекуваат **функции без повратна вредност** (англ. non value-returning functions) или **процедури** (англ. procedures).

Пример 2.5.2

Еден од ефикасните начини за генерирање на различни низи случајни броеви е користење на компјутерскиот часовник. Часовникот се чита со функцијата time(0), која се наоѓа во библиотеката <**ctime**> и која мора да се вклучи во програмата, а враќа неозначен цел број на времето во тој момент изразено во секунди.

```
1   #include <iostream>
2   #include <string>
3   #include <ctime>
4
5   using namespace std;
6
7   int main()
8   {   // Slucajni broevi: funkcii random() i srand()
9
10      cout << "Slucajni broevi pomegju 0 i " << RAND_MAX << ": \n"
11          << rand() << " " << rand() << " " << rand() << " "
12          << rand() << " " << rand() << " " << endl;
13      int max = 100;
14      cout << "\nSlucajni broevi pomegju 0 i " << max - 1 << ": \n"
15          << rand() % max << " " << rand() % max << " " << rand() % max << " "
16          << rand() % max << " " << rand() % max << " " << endl;
17      cout << "\nSlucajni broevi pomegju 1 i " << max << ": \n"
18          << 1 + rand() % max << " " << 1 + rand() % max << " " << 1 + rand() % max
19          << " " << 1 + rand() % max << " " << 1 + rand() % max << " " << endl;
20      int poceten = 51;
21      cout << "\nSlucajni broevi pomegju 1 i " << poceten + max - 1 << ": \n"
22          << poceten + rand() % max << " " << poceten + rand() % max << " "
```

Слика 2.5.2

```
23          << poceten + rand() % max << " " << poceten + rand() % max << " "
24          << poceten + rand() % max << " " << endl;
25      srand(123);
26      srand(time(0));
27      cout << "\n Vreme vo sekundi: " << time(0) << endl;
28      cout << "Slucajni broevi pomegju 1 i " << poceten + max - 1 << ": \n"
29          << poceten + rand() % max << " " << poceten + rand() % max << " "
30          << poceten + rand() % max << " " << poceten + rand() % max << " "
31          << poceten + rand() % max << " " << endl;
32
33      cout << endl;
34      system("Color 17");
35      system("pause");
36      return 0;
37  }
```

Слика 2.5.2 продол ение

Еден излез од програмата е:

```
Slucajni broevi pomegju 0 i 32767:
19169 26500 6334 18467 41

Slucajni broevi pomegju 0 i 99:
64 62 58 78 24

Slucajni broevi pomegju 1 i 100:
62 28 82 46 6

Slucajni broevi pomegju 1 i 150:
87 78 93 146 142

 Vreme vo sekundi: 1469456833
Slucajni broevi pomegju 1 i 150:
61 72 108 132 137

Press any key to continue . . .
```

Со користење на функцијата time(0), во програмата од ***слика 2.5.2***, ќе се генерираат следниве низи случајни броеви при извршување на програмата два-пати:

```
 Vreme vo sekundi: 1469456809
Slucajni broevi pomegju 1 i 150:
56 108 52 114 59
 Vreme vo sekundi: 1469456833
Slucajni broevi pomegju 1 i 150:
61 72 108 132 137
```

Функции за работа со знаци

Во C++ се воведени повеќе функции за работи со знаци. Тие се наоѓаат во библиотеката <cctype>, која мора да се вклучи на почетокот од програмата.

isdigit(z)	– true ако z е цифра (0, 1... 9).
isalpha(z)	– true ако z е буква.
islower(z)	– true ако z е мала буква.
isupper(z)	– true ако z голема буква.
isblank(z)	– true ако z е празно место.
isspace(z)	– true ако z е празен простор: ' ','\t', '\n', '\v', '\f', '\r'[9].
isalnum(z)	– true ако z е буква или цифра.
ispunct(z)	– true ако z е интерпункциски знак: . , ; : '' „" ? / ! – ( ) [ ] { } < > … итн
isprint(z)	– true ако z е знак што се печати. (На пример, '\n', '\t' и други не се печатат.)
tolower(z)	– ако z е голема буква ја конвертира во мала буква.
toupper(z)	– ако z е мала буква ја конвертира во голема буква.

Пример 2.5.3

```
1   #include <iostream>
2   #include <string>
3   #include <iomanip>
4
5   using namespace std;
6
7   int main() {    // Funkcii za rabota so znaci
8
9       char z1 = '5';
10      char z2 = 'W';
11      char z3 = '@';
12      char z4 = '\n';
13      char z5 = ' ';
14      char z6 = '?';
15      bool daNe;
16      daNe = isdigit( z1 );
17      cout << "Dali e " << z1 << " cifra? " << boolalpha << daNe << endl;
18      daNe = isdigit( z2 );
19      cout << "Dali e " << z2 << " cifra? " << boolalpha << daNe << endl;
20      daNe = isalpha( z2 );
21      cout << "Dali e " << z2 << " bukva? " << boolalpha << daNe << endl;
22      daNe = isalpha( z3 );
23      cout << "Dali e " << z3 << " bukva? " << boolalpha << daNe << endl;
24      daNe = islower( z2 );
25      cout << "Dali e " << z2 << " mala bukva? " << boolalpha << daNe << endl;
```

Слика 2.5.

[9] '\t' – хоризонтален табулатор, '\n' – нова линија, '\v' – вертикален табулатор, '\f' – нова страница, '\r' – претходна линија.

```
26      daNe = isupper( z2 );
27      cout << "Dali e " << z2 << " golema bukva? " << boolalpha << daNe << endl;
28      daNe = isblank( z4 );
29      cout << "Dali e " << z4 << " prazno mesto? " << boolalpha << daNe << endl;
30      daNe = isblank( z5 );
31      cout << "Dali e " << z5 << " prazno mesto? " << boolalpha << daNe << endl;
32      daNe = isspace( z4 );
33      cout << "Dali e " << z4 << " prazen prostor? " << boolalpha << daNe << endl;
34      daNe = isspace( z5 );
35      cout << "Dali e " << z5 << " prazen prostor? " << boolalpha << daNe << endl;
36      daNe = isalnum( z1 );
37      cout << "Dali e " << z1 << " bukva ili cifra? " << boolalpha << daNe << endl;
38      daNe = isalnum( z2 );
39      cout << "Dali e " << z2 << " bukva ili cifra? " << boolalpha << daNe << endl;
40      daNe = isalnum( z6 );
41      cout << "Dali e " << z6 << " bukva ili cifra? " << boolalpha << daNe << endl;
42      daNe = ispunct( z6 );
43      cout << "Dali e " << z6 << " interpunciski znak? " << boolalpha << daNe << endl;
44      daNe = ispunct( z4 );
45      cout << "Dali e " << z4 << " interpunciski znak? " << boolalpha << daNe << endl;
46      daNe = isprint( z4 );
47      cout << "Dali e " << z4 << " znak sto se pecati? " << boolalpha << daNe << endl;
48      daNe = isprint( z6 );
49      cout << "Dali e " << z6 << " znak sto se pecati? " << boolalpha << daNe << endl;
50      cout << "Bukvata " << z2;
51      z2 = tolower( z2 );
52      cout << " pretvorena vo mala e " << z2 << endl;
53      cout << "Bukvata " << z2;
54      z2 = toupper( z2 );
55      cout << " pretvorena vo golema e " << z2 << endl;
56
57      cout << endl;
58      system( "Color 17" );
59      system( "pause" );
60      return 0;
61  }
```

```
Dali e 5 cifra? true
Dali e W cifra? false
Dali e W bukva? true
Dali e @ bukva? false
Dali e W mala bukva? false
Dali e W golema bukva? true
Dali e
 prazno mesto? false
Dali e   prazno mesto? true
Dali e
 prazen prostor? true
Dali e   prazen prostor? true
Dali e 5 bukva ili cifra? true
Dali e W bukva ili cifra? true
Dali e ? bukva ili cifra? false
Dali e ? interpunciski znak? true
Dali e
 interpunciski znak? false
Dali e
 znak sto se pecati? false
Dali e ? znak sto se pecati? true
Bukvata W pretvorena vo mala e w
Bukvata w pretvorena vo golema e W
```

Слика 2.5. *(продолжение)*

Излезот по извршување на програмата е:

Оператор sizeof()

Овој оператор се користи за пресметување на големината на меморијата (во бајти) што ја зафаќа некој тип податоци. Неговата синтакса е

sizeof(*тип*);

Тој може да се примени и на една променлива или израз, при што не се потребни загради

sizeof *израз;*

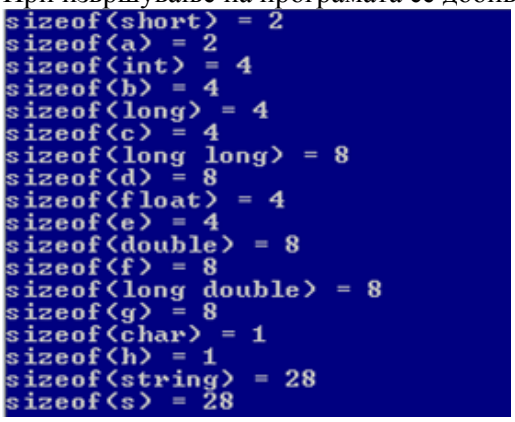
Пример 2.5.4

Со програмата на ***слика 2.5.4*** е илустрирано користењето на операторот sizeof().

```
1   // Operator sizeof()
2
3   #include <iostream>
4   #include <string>
5   using namespace std;
6
7   int main() {
8       short a;    int b;  long c; long long d;
9       float e;    double f;   long double g;
10      char h;     string s;
11
12      cout << "sizeof(short) = " << sizeof(short) << endl;
13      cout << "sizeof(a) = " << sizeof a << endl;
14      cout << "sizeof(int) = " << sizeof(int) << endl;
15      cout << "sizeof(b) = " << sizeof b << endl;
16      cout << "sizeof(long) = " << sizeof(long) << endl;
17      cout << "sizeof(c) = " << sizeof c << endl;
18      cout << "sizeof(long long) = " << sizeof(long long) << endl;
19      cout << "sizeof(d) = " << sizeof d << endl;
20      cout << "sizeof(float) = " << sizeof(float) << endl;
21      cout << "sizeof(e) = " << sizeof e << endl;
22      cout << "sizeof(double) = " << sizeof(double) << endl;
23      cout << "sizeof(f) = " << sizeof f << endl;
24      cout << "sizeof(long double) = " << sizeof(long double) << endl;
25      cout << "sizeof(g) = " << sizeof g << endl;
26      cout << "sizeof(char) = " << sizeof(char) << endl;
27      cout << "sizeof(h) = " << sizeof h << endl;
28      cout << "sizeof(string) = " << sizeof(string) << endl;
29      cout << "sizeof(s) = " << sizeof s << endl;
30
31      cout << endl;
32      system("Color 17");
33      system("pause");
34      return 0;
35  }
```

Слика 2.5.4

При извршување на програмата се добива:



Задачи за вежбање

Да се напишат програми за следниве задачи со користење библиотечни функции:

1. Да се внесе вредност за x и да се пресмета збирот $1 + x + x^2 + x^3 + x^4 + x^5$.
2. Да се пресмета збирот $e^0 + e^1 + e^2 + e^3 + e^4 + e^5$.
3. Да се внесе цел број n и да се пресмета $\left[\sqrt{\frac{n^2}{2}}\right]$, [] означува цел дел.
4. Да се внесат коефициентите a, b и c на квадратната равенка $ax^2 + bx + c = 0$ и да се пресметаат корените по формулата

   $$D = b^2 - 4ac, \qquad x_1 = \frac{-b + \sqrt{D}}{2a}, \qquad x_2 = \frac{-b - \sqrt{D}}{2a}.$$

5. Да се отпечати следнава табела со вредности за тригонометриските функции sin(x) и cos(x) за аглите од 0^o, 30^o, 45^o, 60^o, 90^o, 180^o, 270^o и 360^o. (Аголот се задава во радијани, $x^{rad} = \frac{\pi}{360} x^o$).

x	0^o	30^o	45^o	60^o	90^o	180^o	270^o	360^o
sin(x)								
cos(x)								

6. Да се генерира случаен непарен број од интервалот [31, 100].
7. Да се внесе буква и да се утврди дали е голема.
8. Да се внесе знак и да се утврди дали е интерпункциски знак.

Кориснички функции

Рековме дека во структурираното програмирање (објаснето во точка **1.2 Алгоритми**) се користат две техники на програмирање, и тоа:

- **Програмирање одгоре надолу** (англ. top-down programming).
- **Модуларно програмирање** (англ. modular programming).

Програмирањето одгоре надолу се врши со разделување (расчленување) на задачата на помали и поедноставни задачи, кои ќе ги наречеме **подзадачи**. Ако е потребно, и тие подзадачи понатаму се разделуваат на уште поедноставни додека не се добијат задачи што лесно се програмираат.

Секоја подзадача од така расчленетата задача може да се разгледува како посебна задача, независно од другите. За секоја подзадача може да се напише посебен алгоритам, кој ќе го наречеме **подалгоритам**.

На пример, алгоритам за наоѓање на најголемиот од три дадени броја може да се напише и на следниов начин, ***слика 2.5.5***:

```
алгоритам НајголемОдТриБроја
почеток
        читај a, b, c;
        ако a > b
                тогаш
                        pogolem ← a;
                инаку
                        pogolem ← b;
        крај_ако {a > b}
        p ← pogolem;
        ако p > c
                тогаш
                        pogolem ← p;
                инаку
                        pogolem ← c;
        крај_ако {p > c}
        n ← pogolem;
        печати n;
крај {НајголемОдТриБроја}
```

Слика 2.5.5

Во алгоритамот двапати се бара одредување на поголемиот од два броја: a и b, и p и c. Тоа може да се издвои како посебен алгоритам, односно подалгоритам.

Меѓутоа, се јавува проблем како да се напише тој подалгоритам за да може со него да се извршат двете споредувања: a и b во првото и p и c во второто

споредување. Исто така, подалгоритамот треба да врши споредување на кои било два броја. За да се овозможи тоа, подалгоритамот се пишува во општа форма.

Еден подалгоритам се запишува како и секој друг алгоритам. И во него може да се внесуваат и да се печатат податоци, да се декларираат променливи, да се дефинираат типови и константи, да се доделуваат вредности на променливи, да се извршуваат различни пресметувања и други операции кои може да се извршат и во алгоритмите.

Подалгоритмите, исто како и алгоритмите, се именуваат. На пример: PogolemBroj(), ProstBroj(), NajmalOdSite(), *Замена(), ПоместувањеНиза()* итн.[10]

На пример, подалгоритам за наоѓање на поголемиот од два броја може да се дефинира како на ***слика 2.5.6***:

```
подалгоритам PogolemBroj(broj1, broj2)
почеток
        ако broj1 > broj2
                тогаш
                        pogolem ← broj1
                инаку
                        pogolem ← broj2;
        крај_ако {broj1 > broj2}
        врати pogolem;
крај {Pogolem}
```

Слика 2.5.6

Името на подалгоритамот е PogolemBroj(), а има два влезни параметри broj1 и broj2.

Резултатот, кој се пресметува во подалгоритамот, се враќа со чекорот

```
        врати pogolem;
```

Ваквите подалгоритми се нарекуваат **функциски подалгоритми**.

Ако чекорите за наоѓање на поголемиот број во алгоритамот од ***слика 2.5.5*** ги замениме со подалгоритамот од ***слика 2.5.6***, алгоритамот ќе изгледа како на ***слика 2.5.*** :

```
алгоритам НајголемОдТриБроја
почеток
        читај a, b, c;
        p ← PogolemBroj(a, b);
        n ← PogolemBroj(p, c);
        печати n;
крај {НајголемОдТриБроја}
```

Слика 2.5.

[10] Имињата на подалгоритмите ќе ги пишуваме со првата буква голема. Ќе забележите дека некои имиња се напишани латиница, а некои кирилица. Тоа ќе го објасниме подоцна.

На десната страна од чекорите:

p ← PogolemBroj(a, b);
n ← PogolemBroj(p, c);

се повикува подалгоритамот PogolemBroj() за аргументите a, b и p, c. Ова значи дека функциски подалгоритам се повикува со името и аргументите во загради. Бидејќи функциски подалгоритам враќа само една вредност, тој може да се повика во чекор за доделување (како на ***слика 2.5.***), во израз или во друг чекор.

Општа форма на повик на функциски подалгоритам е

ImeNaPodalgoritamot(*листа_на_влезни_аргументи*);

Подалгоритамот PogolemBroj() може да се напише и така што повратната вредност да биде параметар, ***слика 2.5.6 а***:

```
подалгоритам ПоголемБрој(↓broj1, ↓broj2, ↑pogolem)
почеток
        ако broj1 > broj2
                тогаш
                        pogolem ← broj1
                инаку
                        pogolem ← broj2;
        крај_ако {broj1 > broj2}
крај {Pogolem}
```

Слика 2.5.6 а

Ваквите подалгоритми се нарекуваат **процедурални подалгоритми** или само **процедури** (англ. procedures). Нивната листа на параметри може да има влезни параметри, излезни параметри и влезно-излезни[11].

За да се разликуваат влезните, излезните и влезно-излезните параметри во процедуралните подалгоритми, пред влезните параметри ќе ставаме стрелка надолу ↓, а пред излезните параметри стрелка нагоре ↑. Ако некој параметар е влезно-излезен, тогаш пред него ќе ставаме двонасочна стрелка ↕. (Овие ознаки не се користат кај функциските подалгоритми бидејќи кај нив сите параметри се влезни).

Процедуралниот подалгоритам се повикува само со своето име, а во загради се наведуваат аргументите:

ИмеНаПодалгоритамот(листа_на_влезни_на_излезни_и_на_влезно_излезни_аргументи);

Забелешка: Името на процедуралните подалгоритми ќе го пишуваме со кирилица за да се познава дека процедурален подалгоритам не може да се повика во друг чекор од алгоритамот, како што може функциски подалгоритам.

[11] Во програмските јазици постојат различни начини за означување на влезните, излезните и влезно-излезните параметри. Најчест се означуваат со in, out и inout.

Според кажаното, алгоритамот за наоѓање на најголемиот од три дадени броја, со користење на процедуралниот подалгоритам *ПоголемБрој()*, ќе биде како на ***слика 2.5. а***:

```
алгоритам НајголемОдТриБроја
почеток
        читај a, b, c;
        ПоголемБрој(a, b, p);
        ПоголемБрој(p, c, n);
        печати n;
крај {НајголемОдТриБроја}
```

Слика 2.5. а

Процедуралните подалгоритми се користат најчесто во случаи кога подалгоритамот треба да врати повеќе од една вредност.

На пример, при решавање на систем од две линеарни равенки со две непознати, решението се состои од две вредности за двете непознати. Или, при решавање на квадратна равенка, решението се состои од две вредности за квадратните корени и слично.

Подалгоритмите кои се кодираат во програмските јазици се нарекуваат **потпрограми** (англ. subprograms). Потпрограмите може да бидат напишани како **функции** (англ. functions) или како **процедури** (англ. procedures). Функциите се потпрограми кои даваат (враќаат по извршување) само еден резултат. Процедурите се потпрограми кои не враќаат вредност. (Но постојат механизми кои овозможуваат со процедурите да се добијат повеќе резултати).

Суштината на потпрограмите е во тоа што тие може да се користат во различни програми, а и на повеќе места во иста програма. Тоа е овозможено преку механизмот на **формални аргументи** (англ. formal arguments) и **вистински аргументи** (англ. actual arguments). Во литературата, формалните аргументи често се нарекуваат **параметри** (англ. parameters), а вистинските аргументи само **аргументи**. Ние ќе ги користиме последните два термини.

Во C++ има само функции. Функциите (како и во математиката) даваат (враќаат) само еден резултат, т. е. една вредност. Затоа (како и библиотечните функции), се нарекуваат **функции со повратна вредност**. Со такви функции се реализираат функциските подалгоритми. Процедуралните подалгоритми се реализираат со процедури (како и библиотечните функции), кои во C++ се наречени **функции без повратна вредност**.

Функциите во C++ може да бидат:

- Библиотечни функции.
- Кориснички дефинирани функции.

Со библиотечните функции се запознавме во поднасловот **Библиотечни функции.**

Корисничките функции (англ. user functions) се дефинираат од корисниците (програмерите) и затоа, се наречени и **кориснички дефинирани функции** (англ. user-defined functions).

Кориснички функции со повратна вредност

Ќе започнеме со примерот за наоѓање на најголемиот од три дадени броја, објаснет претходно.

Функцискиот подалгоритам PogolemBroj() од ***слика 2.5.6***, може да се дефинира во C++ како функција, ***слика 2.5.8***:

```
int pogolemBroj(int broj1, int broj2) {
	int pogolem;
	if(broj1 > broj2)		// Ako broj1 e pogolem od broj2,
		pogolem = broj1;	// togas e pogolem broj1,
	else				// inaku
		pogolem = broj2;	// e pogolem broj2.
	return pogolem;
}
```

Слика 2.5.8

Името на функцијата е pogolemBroj[12]. Пред името се задава типот на резултатот што го враќа функцијата. Во примеров, резултатот од функцијата pogolemBroj() е од типот int. По името на функцијата, во загради, се задаваат имињата и типовите на параметрите broj1 и broj2. Параметрите се променливи кои се користат во дефиницијата на функцијата. Функцијата го дава (враќа) резултатот со наредбата

```
return pogolem;
```

Главната функција main() за наоѓање на најголемиот од три броја, е дадена на ***слика 2.5.9***:

```
int main() {
	int a; // prv cel broj
	int b; // vtor cel broj
	int c; // tret cel broj
	int p, n;
	cout << "Vnesete tri celi broja: \n";
	cout << "Prv broj: ";
	cin >> a;
	cout << "Vtor broj: ";
	cin >> b;
	cout << "Tret broj: ";
	cin >> c;
```

[12] Имињата на корисничките функции ќе ги пишуваме со првата буква мала. Ако името се состои од повеќе споени зборови, тогаш секој збор (освен првиот) ќе го пишуваме со голема буква.

```
        // a, b i c se argumenti
        p = pogolemBroj(a, b);      // Povik na funkcijata
        n = pogolemBroj(p, c);      // Povik na funkcijata
        cout << "Najgolem od broevite " << a << ", " << b << "i"
                << c << "e" << n << endl;

        cout << endl;
        system("Color 17");
        system("pause");
        return 0;
}
```

Слика 2.5.9

Ќе ја анализираме функцијата main().

По читање на трите броја a, b и c, се повикува функцијата со аргументите a и b на десната страна во наредбата за доделување

```
 p = pogolemBroj( a, b );
```

Притоа, вредностите на аргументите a и b се пренесуваат во параметрите broj1 и broj2. При извршување на функцијата, поголемата вредност од broj1 и broj2 се доделува на променливата pogolem и претставува резултат од функцијата.

Со наредбата

```
return pogolem;
```

функцијата `pogolemBroj()` го враќа резултатот, т. е. вредноста на променливата pogolem во главната програма. Затоа, велиме дека функцијата `pogolemBroj()` е функција со повратна вредност.

Со наредбата за доделување

```
p = pogolemBroj(a, b);
```

вратената вредност на променливата pogolem ѝ се доделува на променливата p.

Извршувањето на наредбата за доделување

```
p = pogolemBroj(a, b);
```

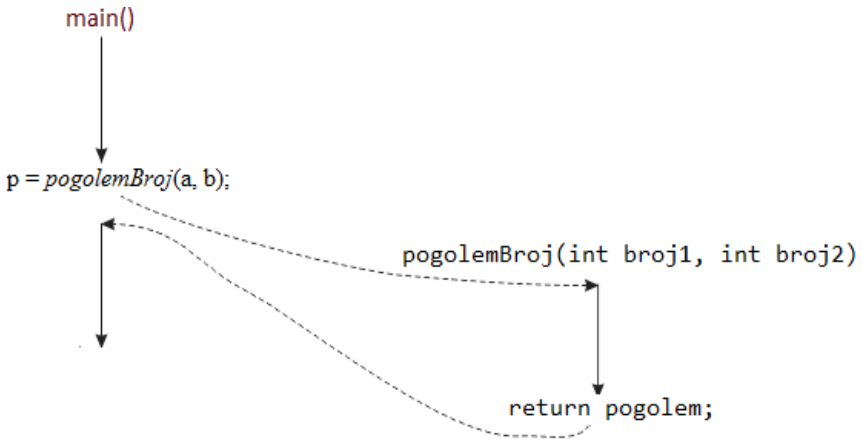
е претставено шематски на ***слика 2.5.10***.

При извршување на следната наредба за доделување, во која повторно се повикува функцијата pogolemBroj(), на променливата n ѝ се доделува вредноста на поголемиот од броевите p и c

```
n = pogolemBroj(p, c);
```

По извршување на оваа наредба, променливата n ја содржи најголемата вредност од променливите a, b и c.

Аргументите на функцијата може да бидат константи, променливи или изрази, чиј тип е ист или компатибилен со типот на соодветните параметри во дефиницијата на функцијата.



Слика 2.5.10

На пример, следните повици на функцијата pogolemBroj() се валидни:

```
p = pogolemBroj((int) (20 * 123 / 45), 6);
p = pogolemBroj((int) 1.2345f, (int) Math.sqrt(1.5));
```

При секој повик на функцијата, се проверуваат бројот и типот на аргументите за време на преведувањето на програмата.

- Ако бројот на аргументите е различен од бројот на параметрите, се јавува порака за грешка.
- Ако не се сложува типот на параметрите со типот на соодветните аргументи, прво преведувачот се обидува да го „преведе“ (конвертира) типот на аргументот во типот на соодветниот параметар. На пример, double во int. (Притоа, може да се изврши повикот, но резултатот да биде погрешен). Ако не успее „преведувањето“ на типот, тогаш преведувачот јавува порака за грешка.

Една функција со повратна вредност може да се повика на различни места во програмата, со различни аргументи и на различни начини.

На пример:

```
        // Eden nacin na povik na funkcija:
p = pogolemBroj(a, b);
        // Drug nacin na povik na funkcija:
System.out.println("\nPogolem e brojot: " + pogolemBroj(a, b));
        // Tret nacin na povik na funkcija:
if(pogolemBroj(a, b) > 99)
        System.out.print("\nPrva nagrada.");
```

Ако ги ставиме во една програма функцијата pogolemBroj() и главната функција main(), како на ***слика 2.5.11*** и ако ја извршиме, ќе го добиеме следниот излез:

```
Vnesete tri celi broja:
Prv broj: 123
Vtor broj: 321
Tret broj: 231
Najgolem od broevite 123, 321 i 231 e 321

Press any key to continue . . .
```

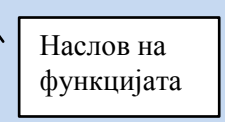
```
#include <iostream>
using namespace std;

// Definicija na funkcijata pogolemBroj().
// broj1 i broj2 se vlezni parametri.

      Функцијата pogolemBroj()

// Glavna funkcija

      Функцијата main()
```

Слика 2.5.11

Дефиниција и декларација на функција со повратна вредност

Дефиниција на функција со повратна вредност во C++ се врши на следниот начин:

тип име (листа_на_параметри) {
    тело на функцијата (наредби)
    return *израз;*
}



***Забелешки**:*

- Насловот на функцијата не завршува со **; (точка и запирка),**
- *тип* е типот на функцијата, кој е, всушност, тип на повратната вредност,
- *име* е името на функцијата.
- *листа_на_параметри* (англ. parameters list) мора да ги содржи и имињата и типовите на параметрите.
- *израз* е резултатот од функцијата кој може да биде израз (од променливи и/или константи и/или функции) или една променлива и кој/а мора да е од ист тип со *тип* на функцијата.
- Параметрите во *листа_на_параметри* може да бидат **влезни параметри** (англ. input parameters) и **влезно-излезни параметри** (англ. input-output parameters).

Декларација на функција во C++ се врши со наредба со форма

тип име (*листа_на_параметри*);

тип – типот на вредноста која ја враќа функцијата,
име – името на функцијата,

листа_на_параметри – ги содржи имињата на параметрите и нивните типови, одделени со запирки.

***Забелешки**:*

- Ако не се наведе *тип* на функцијата, се подразбира типот int.
- Имињата на функциите (по конвенција) ќе ги пишуваме како и променливите[13], т. е. со мала почетна буква. Ако името се состои од повеќе збора, секој следен збор ќе го пишуваме со голема буква.
- Листата на параметри мора да ги содржи типовите на параметрите, а имињата може, но не мора.
- Декларацијата на функција завршува со знакот ; (точка и запирка).

Декларацијата на функција се нарекува **функциски прототип** (англ. function prototype).

Со функцискиот прототип се опишува интерфејсот на функцијата, односно тој дава информации за бројот и типот на параметрите кои треба да бидат познати пред да се повика функцијата, како и за типот на резултатот.

Ќе наведеме неколку примери на функциски прототипи на кориснички функци:

```
int mnozi(int m, int n);          – име: mnozi
                                  – параметри: m и n
                                  – тип на функцијата: int
float koren(float x);             – типот на функцијата е float
int pomal(int , int);             – листа без имиња на параметрите
int g();                          – функција со празна листа на параметри,
                                    т. е. без параметри.
int f(int a, b);                  – неисправно, треба:
                                    int f(int a, int b);
long plostina(long dolzina,long sirina);  – тип на функцијата: long
int h(void);                      – функција  со листа без параметри [14]
void pecati(int brojPoraka);      – функција од типот  void[15]
```

Делот од функцискиот прототип кој ги содржи името и листата на параметри се нарекува **функциски потпис** (англ. function signature).

На пример, функциски потписи се:

```
mnozi(int m, int n);
koren(float b);
pomal(int, int);
plostina(long, long);
```

[13] Имињата на функциите може да се користат во изрази, а и како аргументи на други функции, т. е. исто како и променливите.

[14] Зборот void ќе го објасниме подоцна.

[15] Ќе објасниме подоцна.

Функција со повратна вредност се повикува со наредба за доделување:

променлива = име(листа_на_аргументи);

Листата_на_аргументи и *листата_на_параметри* на функцијата *име* мора да имаат:

- ист број на аргументи и параметри,
- ист редослед на аргументите и параметрите,
- ист или конвертибилен тип на аргументите и соодветните параметри.

Ќе наведеме примери за повик на функции:

```
proizvod = mnozi(a, b);
kvKoren = koren(12.345);
n = pomal(c, p);
P = plostina(dolzina, sirina);
```

При повик на функција, вредностите на аргументите (кои претходно мора да се дефинирани – имаат вредност) се пренесуваат (копираат) во соодветните параметри.

На пример, при повикот

```
proizvod = mnozi(a, b);
```

вредностите на аргументите a и b се пренесуваат (копираат) во параметрите m и n од дефиницијата на функцијата mnozi()

```
int mnozi(int m, int n) {...}
```

Функциите кои се дефинирани по главната функција main(), мора да се декларираат (со наведување на нивниот прототип) пред главната функција за да може да се повикуваат во главната функција, односно за да знае преведувачот кога ја преведува главната функција што претставуваат тие имиња.

Во дефиницијата на функцијата, мора да се наведе што се враќа како резултат по нејзиното извршување, било да се повикува од главната функција main() или од друга функција.

Една функција може само еднаш да се дефинира во истата програма.

Константни параметри на функција

Во функцијата pogolemBroj() од ***слика 2.5.8***, параметрите се broj1 и broj2. Но можно е во самата функција да се промени вредноста на еден од овие параметри.

На пример, ако во функцијата ја додадеме наредбата:

```
broj1 = 99999;
```

тогаш во програмата од ***слика 2.5.11***, без оглед на вредностите што ги внесуваме за broj1, broj2 и broj3, а што се помали од 99 999, резултатот ќе биде 99 999:

```
Vnesete tri celi broja:
Prv broj: 12345
Vtor broj: 23456
Tret broj: 4567

Najgolem od broevite 12345, 23456 i 4567 e 99999

Press any key to continue . . .
```

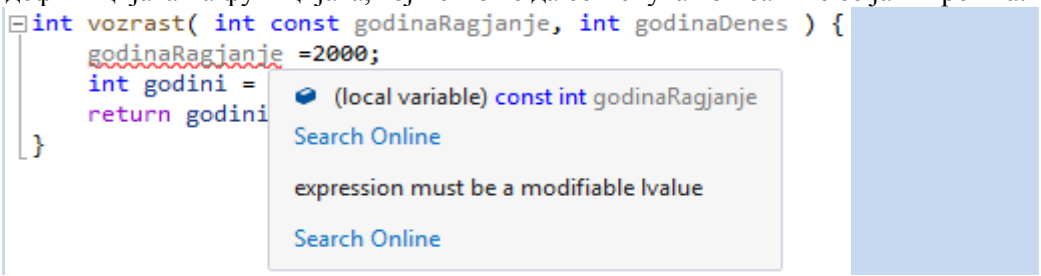
За да се заштитат параметрите во функцијата од измени во неа, тие се означуваат како **константни параметри** со квалификаторот (англ. qualifier) **const**.

Во следнава функција (***слика 2.5.12***) се пресметува возраста на човек, како разлика од тековната година и годината на раѓање. Бидејќи годината на раѓање е непроменлива, треба да се означи како константна.

```
1   #include <iostream>
2   using namespace std;
3
4   int vozrast( int const, int );
5
6   int main() {
7       int godinaRagjanje, godinaDenes;
8       cout << "Vnesete ja godinata na ragjanje: ";
9       cin >> godinaRagjanje;
10      cout << "Vnesete ja segasnata godina: ";
11      cin >> godinaDenes;
12
13      int godini = vozrast( godinaRagjanje, godinaDenes );
14      cout << "\nVasata vozrast e: " << godini << " godini." << endl;
15
16      cout << endl;
17      system( "Color 17" );
18      system( "pause" );
19      return 0;
20  }
21
22  int vozrast( int const godinaRagjanje, int godinaDenes ) {
23      int godini = godinaDenes - godinaRagjanje;
24      return godini;
25  }
```

Слика 2.5.12

Бидејќи параметарот godinaRagjanje е квалификуван како константа во дефиницијата на функцијата, тој не може да се менува во неа – ќе се јави грешка.



Да напоменеме дека во примерот од ***слика 2.5.12***, функцијата е дефинирана по главната функција main() и затоа пред main() е наведен прототипот на функцијата.

Решени задачи

Задача 2.5.1

Да се напише функција за пресметување на x^n, без користење библиотечна функција.

Програмата е дадена на ***слика 2.5.1*** .

```
1   // Funkcija x^n.
2   #include <iostream>
3   using namespace std;
4
5   int XnaN( int n, double x ) {
6       double p = 1;
7       for( int i = 1; i <= n; i++ )
8           p = p * x;
9       return p;
10  }
11
12  int main() {
13      system( "Color 17" );
14      int n;
15      double a, stepen;
16      cout << "Vnesete osnova: a =";  cin >> a;
17      cout << "Vnesete stepen: n = "; cin >> n;
18      stepen = XnaN( n, a );
19      cout << "\n Vrednosta na stepenot " << a << "^" << n
20          << " = " << stepen << endl;
21
22      cout << endl;
23      system( "Color 17" );
24      system( "pause" );
25      return 0;
26  }
```

Слика 2.5.1

Задача 2.5.2

Да се напишат посебни функции за пресметување на бројот и на збирот на цифрите на природен број.

Програмата е дадена на ***слика 2.5.14***.

```
1   // Broj i zbir na cifrite na priroden broj.
2   #include<iostream>
3   using namespace std;
4
5   int brojNaCifri( int m );
6   int zbirNaCifri( int m );
7
8   int main() {
9       system( "Color 17" );
10      int n;
11      cout << "Vnesete priroden broj: ";  cin >> n;
12      cout << "\n Vneseniot broj e  " << brojNaCifri( n )
13          << " - cifren" << endl;
14      cout << "\n Zbirot na cifrite na vneseniot broj e " <<
15          zbirNaCifri( n ) << endl;
16
17      cout << endl;
18      system( "Color 17" );
19      system( "pause" );
20      return 0;
21  }
22
23  int brojNaCifri( int m ) {
24      int broj = 0;
25      do {
26          broj++;
27          m /= 10;
28      } while( m > 0 );
29      return broj;
30  }
31
32  int zbirNaCifri( int m ) {
33      int cifra, zbir = 0;
34      do {
35          cifra = m % 10;
36          zbir += cifra;
37          m /= 10;
38      } while( m > 0 );
39      return zbir;
40  }
```

Слика 2.5.14

Задачи за вежбање

За следниве задачи да се напишат функции со повратна вредност:

1. Пресметување на n!
2. Наоѓање на средна вредност на два броја.
3. Пресметување на периметарот на многуаголник, при тоа да се користи функција за пресметување на растојанието меѓу две точки во рамнина.
   (Ако е $A(x_a, y_a)$ и $B(x_b, y_b)$, растојанието $d = \overline{AB} = \sqrt{(x_b - x_a)^2 + (y_b - y_a)^2}$).

4. Кратење на дропката $\frac{a}{b}$ (a и b се цели броеви) со делење на a и b со својот НЗД. За наоѓање на НЗД од a и b, да се да се напише посебна функција.
5. Наоѓање на НЗС на два броја.
6. Наоѓање на најголемиот заеднички делител за n броеви. За наоѓање на НЗД за два броја, да се напише посебна функција.
7. Да се напишат посебни функции за пресметување на збирот и на разликата на два бинарни броја изразени како цели декадни броеви чии цифри се само 0 и 1. (На пример, за x = 10011101 и y = 01101001.)
8. Пресметување на збирот на два броја во броен систем со основа 8. (Цифрите во овој броен систем се: 0, 1, 2, 3, 4, 5, 6 и 7).
9. Проверка дали природниот број n е совршен. (Совршени броеви се оние кои се еднакви на збирот на своите делители без самиот број). Да се најдат сите совршени броеви помали од n.
10. Проверка дали природниот број n е прост или не е. (Бројот е прост ако е делив само со 1 и со самиот себеси). Да се нјдат сите прости броеви помали од природниот број n.
11. Да се пресмета збирот 1 + (1 + 2) + (1 + 2 + 3) +... + (1 + 2 + 3 +... + n) со употреба на функција за збирот на првите к природни броеви, т.е. 1 + 2 +...+ к.
12. Да се најде декадниот еквивалент на даден цел бинарен број.

Кориснички функции без повратна вредност

Функциите во C++ кои не враќаат вредност велиме дека се функции од типот **void**.

На пример, дефиницијата на функцијата pecatenje() може да биде:

```
void pecatenje(int broj, float rezultat) {
       cout << broj << rezultat << endl;
       // broj  е број од типот инт
       // rezultat е број од типот float
}
```

Повикот на оваа функција може да биде од функцијата main() или од друга функција. Повикот на функција без повратна вредност се разликува од повикот на функција со повратна вредност. Функција без повратна вредност се повикува само со името и листата на аргументи.

На пример:

```
pecatenje(n , x);
```

Функцијата од типот void може да биде и без параметри.

На пример:

```
void vlez() {
       cout << "Vnesuvanje na vlezni podatoci\n ";
}
```

или

```
void izlez(void) {   // So parametarot void e isto
                              // kako i bez nego
       cout << "Pecatenje na izleznite podatoci\n ";
}
```

Повикот на функција од типот void може да биде:

```
vlez();       vlez(void);          izlez();      izlez(void);
```

Од функција од типот void се излегува без наредбата return (како од функција со повратна вредност), автоматски по нејзиното извршување. Но може да се излезе и од кое било место во телото на функција од типот void, со наредбата return без параметар.

На пример:

```
void kvadratenKoren( double broj ) {
    if( broj < 0 ) {
        cout << "Brojot e negativen i nema kvadraten koren." << endl;
        return;
    }
    else
        cout << "Kvadraten koren od " << broj << " e "
        << sqrt( broj ) << endl;
    cout << "Kraj na void-funkcijata." << endl;
}
```

На пример, со

```
double broj = 2;
kvadratenKoren(broj);
```

ќе се отпечати:

```
Kvadraten koren od 2 e 1.41421
Kraj na void-funkcijata.

Press any key to continue . . .
```

Ако функција од типот void се повика со

```
double broj = -2;
kvadratenKoren(broj);
```

ќе се отпечати:

```
Brojot e negativen i nema kvadraten koren.

Press any key to continue . . .
```

Гледаме дека во второто повикување не се извршува наредбата

```
cout << "Kraj na procedurata." << endl;
```

како при првото повикување бидејќи функцијата завршува со наредбата return.

Функциите без повратна вредност се користат најчесто во случаи кога функцијата треба да врати повеќе од една вредност. На пример, кога треба две променливи да си ги разменат содржините, треба и двете вредности да се вратат. Кога треба да се сортираат n податоци, тие треба да се вратат и слично.

Општата синтакса на функција од тип void е:

void *ИмеНаФункцијата*(*листа_на_влезни_ на_излезни_ и_на_влезно_излезни _параметри*) {
тело на функцијата
}

Функциите без повратна вредност се повикуваат само со своето име, а во загради се наведуваат аргументите:

ИмеНаФункцијата(листа_на_влезни_на_излезни_и_на_влезно-излезни _аргументи);

Забелешка: Функција без повратна вредност не може да се повика во друга наредба од главната функција или од друга функција, како што може функција со повратна вредност.

Пример 2.5.5

Да се напише апликација во која ќе се користи функција од типот void за замена на вредностите на две променливи.

Програмата е дадена на ***слика 2.5.15***.

Еден излез од програмата е:

```
a = 3
b = 4
Zameneti  a = 3 i b = 4 se a = 4 i b = 3

Press any key to continue . . .
```

Ако во главната функција main(), по повикот на функцијата zamena() од типот void ја додадеме наредбата:

```
cout << "Vrednosti na a i b po vrakjanje od funkcijata void zamena() se: "
     << "a = " << a << "i b = " << b << endl;
```

ќе се отпечати:

```
a = 3
b = 4
Zameneti  a = 3 i b = 4 se a = 4 i b = 3
Vrednosti na a i b po vrakjanje od void funkcijata zamena() se: a = 3 i b = 4

Press any key to continue . . .
```

Гледаме дека во функцијата од типот void се печатат заменетите вредности на a и b, а по враќање во повикувачот (функцијата main()), нивните вредности не се заменети.

Да ги анализираме процесот на извршување на програмата од **Пример 2.5.5** (***слика 2.5.15***), т. е. повикување на функцијата zamena() од типот void од main(), извршувањето на функцијата и враќањето назад во main().

```
1   #include <iostream>
2   using namespace std;
3
4   void zamena( int, int );
5
6   int main() {
7       // Zamena na vrednosti na dve promenlivi (so void funkcija)
8
9       int a, b;
10      cout << "a = ";     cin >> a;
11      cout << "b = ";     cin >> b;
12      cout << "Zameneti  a = " << a << " i b = " << b << " se ";
13      zamena( a, b );
14
15      cout << endl;
16      system( "Color 17" );
17      system( "pause" );
18      return 0;
19  }
20
21  void zamena( int prv, int vtor ) {
22      int pom;
23      pom = prv;
24      prv = vtor;
25      vtor = pom;
26      cout << "a = " << prv << " i b = " << vtor << endl;
27  }
```

Слика 2.5.15

При извршување на програмата, прво во меморијата се резервира простор (***слика 2.5.16***) за променливите a и b при нивната декларација, а потоа се исполнува со вредностите што се внесени при читањето.

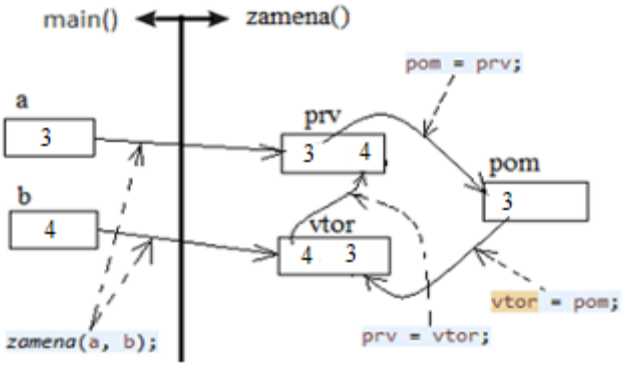
Со повикот на функцијата од типот void, дејството продолжува со извршување на функцијата. Прво, во меморијата се резервира простор за параметрите prv и vtor и во нив се пренесуваат (копираат) вредностите на аргументите a и b. Потоа, се резервира простор за променливата pom и се извршуваат наредбите:

```
pom = prv;
prv = vtor;
vtor = pom;
```

Тоа е покажано на десната страна на ***слика 2.5.16***.

На крајот на функцијата од типот void, се печатат вредностите на prv и vtor, кои ги содржат вредностите на b и a.

По извршување на функцијата zamena() од типот void, дејството се враќа во функцијата main(), каде што променливите a и b си ги задржале истите вредности, што се гледа и од ***слика 2.5.16.***



Слика 2.5.16

Овој механизам се нарекува **пренесување на аргументите по вредност**, а аргументите се нарекуваат **аргументи пренесени по вредност** (англ. pass-by-value arguments).

Битно е да се напомене дека вредностите на параметрите во функцијата од типот void може да се менуваат, но при враќање во повикувачот (главната функција или друга функција), тие не се копираат во соодветните аргументи. Затоа, аргументите ја задржуваат вредноста која ја имале пред повикот на функцијата од типот void. Значи, копирањето се врши еднонасочно, т. е. само на аргументите во параметрите, но не и спротивно. Ваквите параметри се нарекуваат **вредносни параметри** (англ. value parameters).

По извршување на функцијата zamena() од типот void, целокупната меморија која била зафатена при нејзиното извршување (променливите декларирани во неа и во нејзината листа на параметри) се ослободува. Во меморијата остануваат само променливите креирани при извршување на фунцијата main().

Честопати, потребно е да се вратат повеќе резултати од функцијата од типот void во повикувачот (функцијата main() или друга функција). Бидејќи функциите од типот void во C++ се функции кои не враќаат вредност, враќањето на повеќе резултати е овозможено со т.н. **референтни параметри** (англ. reference parameters).

Референтни параметри

Референцирање (англ. referencing), кажано кратко, е доделување друго име на иста променлива. На пример:

```
int b = 3;
int & r = b;
```

Со втората наредба, се декларира референцата r, која упатува на променливата b.

Знакот & (амперсанд) се чита „референца“. Променливата r е **референтна променлива** (англ. reference variable) или кратко **референца** (англ. reference).

Втората наредба се чита: „r е целобројна референца иницијализирана на b“ или „r е целобројна референца која упатува на b“.

При декларирање на референца, таа мора да се иницијализира така што ѝ се доделува променливата на која упатува. По декларирање и иницијализирање на референца, таа не може да се промени да упатува на друга променлива. Но може да се декларираат повеќе референци кои упатуваат на иста променлива.

Исто така, типот на референцата мора да е ист со типот на променливата на која упатува.

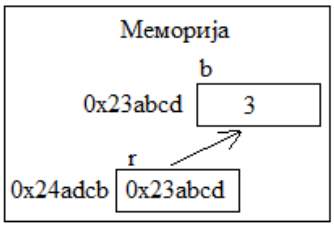
Бидејќи референцата упатува на некоја променлива, често велиме дека референцата е друго име за променливата. На пример, ако ги отпечатиме променливата b и референцата r која упатува на неа:

```
int b;
int& r = b;
b = 3;
cout << "Promenliva b = " << b << endl;
cout << "Referenca r = " << r << endl;
```

ќе се добие иста вредност:

```
Promenliva b = 3
Referenca r = 3

Press any key to continue . . .
```

На следнава скица (***слика 2.5.1***) е илустрирана референцата r која упатува на променливата b во меморијата. Бидејќи и b и r се променливи кои зафаќаат простор во меморијата (се наоѓаат на некои адреси), упатувањето се врши така што референцата r ја содржи адресата на променливата b.



Слика 2.5.1

Примената на референците како референтни параметри во функциите без повратна вредност, ќе ја објасниме на истиот **Пример 2.5.5**.

Бидејќи по враќање од функцијата zamena() од типот void, вредностите на променливите a и b не се променети, функцијата ќе ја напишеме со референтни параметри (***слика 2.5.18***) пред кои се става знакот &:

```
void zamena(int &prv, int &vtor) {
	int pom;
	pom = prv;
	prv = vtor;
	vtor = pom;
	cout << "a = " << prv << "i b = " << vtor << endl;
}
```

Слика 2.5.18

Исто така, и во прототипот на функцијата од типот void треба да се означи дека параметрите се референтни:

```
void zamena(int &, int &);
```

Резултатот по извршување на апликацијата ќе биде точен:

```
a = 3
b = 4
Zameneti  a = 3 i b = 4 se a = 4 i b = 3
Vrednosti na a i b po vrakjanje od void funkcijata zamena() se: a = 4 i b = 3

Press any key to continue . . .
```

Знакот & може да се стави каде било помеѓу типот и името на параметарот:

```
int & prv     int& prv      int &prv
```

Овој механизам на пренесување вредности од повикувачот во функцијата и спротивно е наречен **пренесување по референци** (англ. pass-by-references), а аргументите се нарекуваат **аргументи пренесени по референца** (англ. pass-by-reference arguments).

Повикот на функција од типот void, се врши со наредба која се состои од името на функцијата и листата на аргументи. На пример, функцијата zamena() од типот void ќе се повика со наредбата

```
zamena(a, b);
```

Притоа, бројот и редоследот на аргументите мора да одговара на бројот и редоследот на параметрите. Но

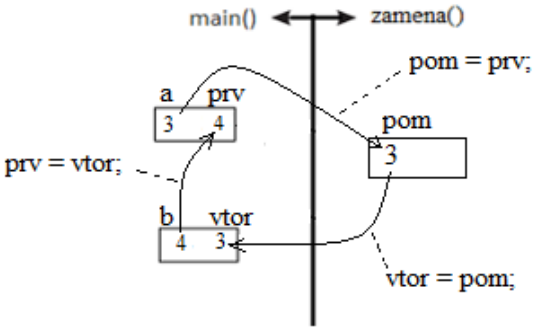
- типовите на аргументите кои се пренесуваат по вредност може да бидат константи, променливи или изрази, чии типови се исти или конвертибилни со типовите на соодветните вредносни параметри,
- типовите на аргументите кои се пренесуваат по референца мора да бидат променливи (не константи или изрази) и тоа од ист тип како соодветните референтни параметри.

При повикување на функција од типот void со референтни параметри, не се креираат нови променливи за референтните параметри, како што се прави за вредносните параметри. За време на извршувањето на функцијата од типот void, *референтните параметри се третираат како други имиња за соодветните аргументи од повикот*. Тоа се постигнува така што на референтните параметри им се доделува адресата на соодветните аргументи. Затоа, *сите измени во референтните параметри, всушност, се вршат врз соодветните аргументи* бидејќи тие се различни имиња на иста мемориска локација.

Скицата на ***слика 2.5.19*** го илустрира извршувањето на апликацијата од ***слика 2.5.15*** со направените измени за параметрите на функцијата од типот void како на ***слика 2.5.18***.

Со споредување на ***слика 2.5.16*** и ***слика 2.5.19***, се гледа разликата при извршување на функцијата zamena() кога е напишана како функција со повратна вредност (***слика 2.5.15***) и како функција без повратна вредност (***слика 2.5.18***).

Во поднасловот **Константни параметри на функција**, објаснивме дека константните параметри на функција со повратна вредност не може да се менуваат во функцијата. Истото важи и за функции без повратна вредност. Независно дали е параметарот вредносен или референтен, ако сакаме да не се менува во функцијата, треба да се означи како константен со квалификаторот const.



Слика 2.5.19

Решени задачи

Задача 2.5.13
Да се подредат три броја по големина.

Програмата е дадена на ***слика 2.5.20***.

```
1   // Podreduvanje na tri broja
2   #include <iostream>
3   using namespace std;
4
5   void zamena( double& prv, double& vtor ) {
6       double pom = prv;
7       prv = vtor;
8       vtor = pom;
9   }
10
11  int main() {
12      double a, b, c;
13      cout << "Vnesete gi broevite a, b i c." << endl;
14      cout << "a = "; cin >> a;
15      cout << "b = "; cin >> b;
16      cout << "c = "; cin >> c;
17      cout << "Podredeni broevite " << a << ", " << b << " i "
18          << c << " se: ";
19      if( a > b ) zamena( a, b );
20      if( a > c ) zamena( a, c );
21      if( b > c ) zamena( b, c );
22      cout << a << ", " << b << ", " << c << endl;
```

Слика 2.5.20

```
23      cout << endl;
24      system( "Color 17" );
25      system( "pause" );
26      return 0;
27  }
```

Слика 2.5.20 продол ение

Еден излез по извршување на програмата е:

```
Vnesete gi broevite a, b i c.
a = 76.5
b = 65.4
c = 54.3
Podredeni broevite 76.5, 65.4 i 54.3 se: 54.3, 65.4, 76.5

Press any key to continue . . .
```

Задача 2.5.14

Да се реши системот од две линеарни равенки со две непознати

$a_1x + b_1y = c_1$

$a_2x + b_2y = c_2$

Решение на системот равенки е:

$D = a_1b_2 - a_2b_1$; $D_1 = c_1b_2 - c_2b_1$; $D_2 = a_1c_2 - a_2c_1$;

ако $D \neq 0$

тогаш

$x = \frac{D_1}{D}$; $y = \frac{D_2}{D}$;

инаку

ако $D_1=0$ ***И*** $D_2=0$

тогаш

печати „Системот има ∞ решенија.“;

инаку

печати „Системот нема решение.“;

Функцијата и апликацијата се дадени на ***слика 2.5.21***.

Еден излез по извршување на апликацијата е:

```
Vnesete gi koeficientite a1, b1 i c1:
a1 = 3
b1 = -2
c1 = 5
Vnesete gi koeficientite a2, b2 i c2:
a2 = -4
b2 = 7
c2 = -1

Sistemot linearni ravenki:
        +3x -2y = +5
        -4x +7y = -1
Ima resenija : x = +2.53846, y = +1.30769

Press any key to continue . . .
```

```
1   // Sistem dve linearni ravenki so 2 nepoznati
2   #include <iostream>
3   using namespace std;
4
5   void sistemLinRavenki( int a1, int b1, int c1, int a2, int b2, int c2,
6                                                   double& x, double& y ) {
7       int D = a1 * b2 - a2 * b1;
8       int D1 = c1 * b2 - c2 * b1;
9       int D2 = a1 * c2 - a2 * c1;
10      if( D != 0 ) {
11          x = ( double ) D1 / D;
12          y = ( double ) D2 / D;
13          cout << "Ima resenija : x = " << x << ", y = " << y << endl;
14      }
15      else {
16          if( D1 == 0 && D2 == 0 )
17              cout << "Ima beskonecno mnogu resenija." << endl;
18          else
19              cout << "E protivrecen i nema resenie." << endl;
20      }
21  }
22
23  int main() {
24      int a1, b1, c1, a2, b2, c2;
25      double x, y;
26      cout << "Vnesete gi koeficientite a1, b1 i c1: \n";
27      cout << "a1 = ";    cin >> a1;
28      cout << "b1 = ";    cin >> b1;
29      cout << "c1 = ";    cin >> c1;
30      cout << "Vnesete gi koeficientite a2, b2 i c2: \n";
31      cout << "a2 = ";    cin >> a2;
32      cout << "b2 = ";    cin >> b2;
33      cout << "c2 = ";    cin >> c2;
34      cout << "\nSistemot linearni ravenki:" << endl;
35      cout << showpos << internal;
36      cout << "\t" << a1 << "x " << b1 << "y = " << c1 << endl;
37      cout << "\t" << a2 << "x " << b2 << "y = " << c2 << endl;
38      sistemLinRavenki( a1, b1, c1, a2, b2, c2, x, y );
39
40      cout << endl;
41      system( "Color 17" );
42      system( "pause" );
43      return 0;
44  }
```

Слика 2.5.21

Задачи за вежбање

Да се напишат функции од типот void за следниве задачи:

1. Да се пресметаат степените на броевите x и y, x^y и y^x.
2. Да се претворат поларните координати ρ и φ во Декартови координати x и y, по формулите: $x = \rho\cos\varphi,\ y = \rho\sin\varphi$.
3. Да се реши квадратната равенка $ax^2 + bx + c = 0$.
4. Да се подредат три броја по големина.
5. Да се најдат најголемиот и најмалиот број кои може да се формираат од цифрите на даден 5-цифрен природен број.
6. Да се најде возраста на човек како разлика на денешниот ден и роденденот. Датумите да се изразат со ден, месец и година.
7. Да се пресмета аритметичката, геометриската и хармониската средина на n внесени броеви.
8. Да се внесат n броеви и да се најдат најголемиот и најмалиот од нив.
9. Да се претвори децимален број во бинарен, така што функцијата да го врати бинарниот еквивалент на целиот дел и на децималниот дел посебно.
10. Да се генерира Паскаловиот триаголник за n = 5.

```
                    1                   n = 0
                1       1               n = 1
            1       2       1           n = 2
        1       3       3       1       n = 3
    1       4       6       4       1   n = 4
1       5       10      10      5       1 n = 5
```

Повикување на функции во функција

Една функција може да се повика во друга функција. Тоа ќе го покажеме со следниов пример.

Пример 2.5.6
Да се најде најголемиот од 5 дадени броја.

За решавање на задачата ќе ја користиме функцијата pogolemBroj() за наоѓање на поголемиот од два броја од ***слика 2.5.8***, а програмата од ***слика 2.5.9*** ќе ја напишеме како функција за наоѓање на најголемиот од три броја под име najgolemOd3Broja(). Потоа, најголемиот од 5 броја ќе го најдеме со повикување на овие функции на следниот начин:

```
pogolemBroj(pogolemBroj(a, b), najgolemOd3Broja(c, d, e));
```

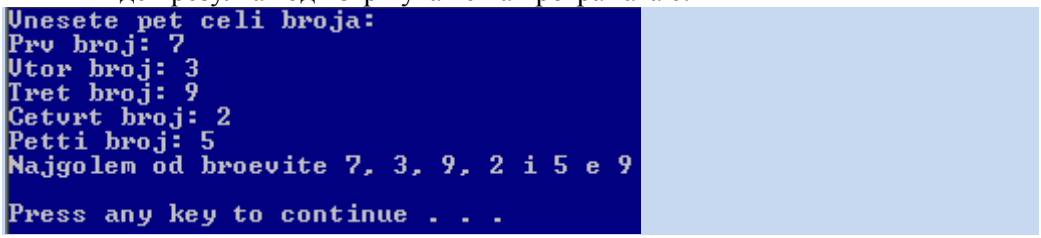
***Напомена**:* Со комбинирање на повици на овие две функции, може да се најде најголемиот од кои било броеви.

Програмата е дадена на ***слика 2.5.22***.

```
#include <iostream>
using namespace std;

int pogolemBroj( int broj1, int broj2 );
int najgolemOd3Broja( int broj1, int broj2, int broj3 );

int main() {
    int a, b, c, d, e, najgolem;
    cout << "Vnesete pet celi broja: \n";
    cout << "Prv broj: ";   cin >> a;
    cout << "Vtor broj: ";  cin >> b;
    cout << "Tret broj: ";  cin >> c;
    cout << "Cetvrt broj: ";    cin >> d;
    cout << "Petti broj: "; cin >> e;

    najgolem = pogolemBroj( pogolemBroj( a, b ),
                            najgolemOd3Broja( c, d, e ) );
    cout << "Najgolem od broevite " << a << ", " << b << ", "
        << c << ", " << d << " i " << e << " e "
        << najgolem << endl;

    cout << endl;
    system( "Color 17" );
    system( "pause" );
    return 0;
}

int pogolemBroj( int broj1, int broj2 ) {
    int pogolem;
    if( broj1 > broj2 )     // Ako broj1 e pogolem od broj2,
        pogolem = broj1;    // togas pogolem e broj1,
    else                    // inaku
        pogolem = broj2;    // pogolem e broj2.
    return pogolem;
}

int najgolemOd3Broja( int broj1, int broj2, int broj3 ) {
    int p, n;
    p = pogolemBroj( broj1, broj2 );    // Funkciski povik
    n = pogolemBroj( p, broj3 );        // Funkciski povik
    return n;
}
```

Слика 2.5.22

Еден резултат од извршување на програмата е:



Глобални и локални променливи

Променливите кои се креирани во програмите и во потпрограмите (функциите) имаат одредено **подрачје на видливост** (англ. scope of visibility) наречено и **подрачје на пристап** (англ. scope of access). Подрачје на видливост на некоја променлива се смета она подрачје во кое има пристап до променливата, т. е. може да се користи во операции.

Некои променливи може да се користат во целата апликација, некои во целата функција, а некои само во делови од функцијата.

Според подрачјето на видливост променливите може да бидат:

- **Глобални променливи**.
- **Локални променливи**.

Подрачјето на видливост на глобалните променливи, односно пристапот до нив е целата апликација, т. е. главната функција и сите кориснички дефинирани функции во истата датотека во која е главната функција.

Имињата на кориснички дефинираните функции во една апликација се третираат како глобални променливи и затоа, една функција може да се повика во која било друга функција во апликацијата. (Функција во C++ не може да се дефинира во телото на друга функција).

Локалните променливи може да имаат подрачје на пристап: само телото на функцијата во која се декларирани, само насловот на функцијата или само блокот помеѓу загради {}. Видливоста започнува со декларацијата на локалната променлива.

Параметрите на функција се третираат како локални променливи во целата функција. Исто така, локалните променливи во една функција се видливи во сите вгнездени наредби, а локалните променливи декларирани во телото на некоја наредба не се видливи надвор од нејзиното тело.

За една променлива велиме дека е **скриена** (прекриена со друга променлива) во некој блок или во цела функција ако не е видлива (не може да се користи) во тој блок или во таа функција. Тоа значи дека ако во подрачјето на видливост на некоја променлива (на пример, int x = 1;) има блок (во наредба) во кој е декларирана променлива со исто име (int x = 2;), во тој блок ќе биде видлива само променливата x со вредност 2.

Пример 2.5.7

Со овој пример ќе ја демонстрираме видливоста на променливите во една апликација.

Во програмата од ***слика 2.5.2*** , променливата ovaaGodina е декларирана како глобална променлива, а променливите lani и idnaGodina се декларирани како локални променливи во функциите main() и mojaFunkcija(). До глобалната променлива ovaaGodina има пристап и од главната функција main() и од функцијата mojaFunkcija(). До локалната променлива lani има пристап само од main(), а нема од mojaFunkcija(), а до локалната променлива idnaGodina има пристап само од mojaFunkcija(), а нема од main().

```
1   #include <iostream>
2   using namespace std;
3
4   void mojaFunkcija();        // Prototip
5
6   int ovaaGodina = 2020;      // Globalna promenliva
7
8   int main() {
9
10      int lani = 2019;    // Lokalna promenliva
11      cout << "\tPecatam od glavnata funkcija main()" << endl;
12      cout << "Ovaa godina e " << ovaaGodina << ", a lani bese " << lani << endl;
13      mojaFunkcija();
14      // Do promenlivata 'idnaGodina' nema pristap od main(),
15      // bidejki e lokalna vo mojaFunkcija()
16
17      cout << endl;
18      system( "Color 17" );
19      system( "pause" );
20      return 0;
21  }
22
23  void mojaFunkcija() {
24      int idnaGodina = 2021;; // Lokalna promenliva
25      cout << "\n\tPecatam od funkcijata mojaFunkcija()" << endl;
26      cout << "Ovaa godina e " << ovaaGodina << ", a idnata godina e "
27          << idnaGodina << endl;
28      int ovaaGodina = 2030;
29      cout << "Ottuka do kraj na funkcijata globalnata promenliva 'ovaaGodina' "
30          << "\ne prekriena so lokalnata promenliva 'ovaaGodina' " << endl;
```

Слика 2.5.2

```
31      cout << "Ovaa godina e " << ovaaGodina << ", a idnata godina e "
32          << idnaGodina << endl;
33
34      cout << "Sepak, pristap do prekrienata globalna promenliva 'ovaaGodina' "
35          << "\ne mozen so operatorot ::" << "\n Ovaa godina e "
36          << ::ovaaGodina << endl;
37      // Do promenlivata 'lani' nema pristap od mojaFunkcija(),
38      // bidejki e lokalna vo main()
39  }
```

Слика 2.5.2 продол ение

Во примерот, глобалната променлива ovaaGodina со вредност 2020 е прекриена (скриена) во mojaFunkcija() бидејќи е повторно декларирана како локална променлива со вредност 2030. Сето ова е јасно од излезот на програмата:

```
        Pecatam od glavnata funkcija main()
Ovaa godina e 2020, a lani bese 2019

        Pecatam od funkcijata mojaFunkcija()
Ovaa godina e 2020, a idnata godina e 2021
Ottuka do kraj na funkcijata globalnata promenliva 'ovaaGodina'
e prekriena so lokalnata promenliva 'ovaaGodina'
Ovaa godina e 2030, a idnata godina e 2021
Sepak, pristap do prekrienata globalna promenliva 'ovaaGodina'
e mozen so operatorot ::
 Ovaa godina e 2020

Press any key to continue . . .
```

Интересно е дека до глобалната променлива ovaaGodina (= 2020), која е прекриена во функцијата mojaFunkcija() со истоимената локална променлива ovaaGodina (= 2030), има пристап со т.н. **оператор за разрешување на подрачјето на видливост** (англ. scope resolution operator), чиј знак е **::**.

Времето од креирањето на една променлива во меморијата (веднаш по извршување на наредбата за декларација) до нејзиното исчезнување (крај на блокот, крај на функцијата или крај на апликацијата) се нарекува **живот на променливата** (англ. lifetime).

Глобалните променливи живеат цело време додека се извршува апликацијата.

Локалните променливи живеат од моментот на креирање до крајот на подрачјето на видливост, кое може да биде целата функција или само блок меѓу загради {}.

Може да се постават следниве правила за подрачјето на видливост на променливите:

- Локалните променливи не може да се користат надвор од подрачјето на видливост.
- Глобалните променливи може да се користат во целата апликација.
- Променливите декларирани во еден блок (функција, тело на наредба – for, while, do-while) може да се користат само во него.

- Променлива декларирана во една функција не може да се користи во друга функција.
- Променливата може да биде скриена во некој дел од своето подрачје на видливост ако во тој дел е повторно декларирана под исто име.
- Две функции со исто име може да имаат исто подрачјето на видливост само ако имаат различни листи на аргументи.

Статички променливи

Променливите може да бидат и **статички** (англ. static). Тие се декларираат со зборот **static**.

На пример:

```
static int broj = 7;
static double iznos;
iznos = 123.45;
```

Локалните статички променливи се однесуваат како глобални променливи. Тие живеат од нивната декларација до завршување на апликацијата. Притоа, се иницијализираат само еднаш при декларацијата или при доделување вредност во првиот повик на функцијата во која се локални променливи. По извршување на функцијата во која се декларирани и враќање на дејството во повикувачот (во main() или друга функција), статичките локални променливи ја задржуваат добиената вредност, т. е. не се уништуваат како нестатичките локалните променливи. При секое следно повикување на функцијата, ја имаат состојбата (вредноста) добиена во претходното повикување.

Глобалните променливи и локалните статички променливи почнуваат да живеат со првото извршување на нивната дефиниција.

Пример 2.5.8

Во апликацијата од ***слика 2.5.24***, функцијата f() се повикува 5 пати.

Во првото повикување на функцијата f(), статичката променлива pocetok се иницијализира на вредност 0, а нестатичката променлива brojac на вредност 100. Со наредбите за инкрементирање, тие добиваат вредност 1 и 101.

При второто повикување на функцијата f(), статичката променлива pocetok не се иницијализира повторно (иако пак се извршува наредбата за декларација и иницијализација), туку (бидејќи е статичка) си ја задржува вредноста 1 од првото повикување на функцијата. Затоа, со наредбата за инкрементирање, добива вредност 2. Нестатичката променливата brojac повторно се иницијализира на вредност 100 и потоа се инкрементира на 101.

Истото се повторува при секое повикување на функцијата f().

Ова значи дека статичката променлива brojac се иницијализира само еднаш (при првиот повик на функцијата f()) и живее како глобална променлива до завршување на апликацијата.

```
1   #include <iostream>
2   using namespace std;
3
4   void f();
5
6   int main() {
7       for( int k = 1; k <= 5; k++ )
8           f();
9
10      cout << endl;
11      system( "Color 17" );
12      system( "pause" );
13      return 0;
14  }
15
16  void f() {
17      static int pocetok = 0; // pocetok e staticka promenliva
18      int brojac = 100;
19
20      pocetok += 1;
21      brojac += 1;
22      cout << "pocetok=" << pocetok << " brojac=" << brojac << endl;
23  }
```

Слика 2.5.24

Излезот од извршување на апликацијата е:

```
pocetok=1 brojac=101
pocetok=2 brojac=101
pocetok=3 brojac=101
pocetok=4 brojac=101
pocetok=5 brojac=101

Press any key to continue . . .
```

Ако променливата brojac не е статичка, тогаш излезот ќе биде:

```
pocetok=1 brojac=101
pocetok=1 brojac=101
pocetok=1 brojac=101
pocetok=1 brojac=101
pocetok=1 brojac=101

Press any key to continue . . .
```

Користењето на глобалните и на статичките променливи треба да е многу внимателно бидејќи тие се пристапни во сите функции во апликацијата. Можно е да дојде до измена на нивната содржина во една функција, а во друга тоа да предизвика грешка.

Вградени функции

При извршување на една апликација, за сите функции во неа се прави по една копија во меморијата. Познато е дека извршувањето на секоја апликација започнува со функцијата main(). Ако во неа се повикува друга функција, тогаш дејството прекинува на местото на повикот и се скока на локацијата во меморијата каде што се наоѓа повиканата функција. По нејзиното извршување, дејството се враќа во main() и продолжува со следната наредба по наредбата за повик на функцијата.

Овој механизам за повик на функција и враќање во повикувачот (кој може да биде и друга функција освен main()), бара одредено време за извршување. Во случај кога повиканата функција е мала, може да се случи времето за повик на функцијата и враќањето од неа да биде поголемо од времето на самото извршување на функцијата. Губењето време е поизразено ако мала функција се повикува почесто, на пример, во наредба за повторување.

Во таквите случаи, попрактично е функцијата да се вгради во кодот на повикувачот (на пример во main()) при неговото преведување. (Спомнавме дека библиотечните функции во C++ се вградени функции).

За таа цел, во C++ се користи квалификаторот **inline**, кој се става пред дефиницијата на функцијата за да му се каже на преведувачот да ја вгради во програмите кои ја повикуваат, на местата на кои се повикува.

Пример 2.5.9

Во програмата на ***слика 2.5.25*** функцијата igraBonus() е мала и се извршува повеќепати, затоа може да се квалификува како вградена функција.

```
1   #include <iostream>
2   #include <string>
3   #include <ctime>
4   using namespace std;
5
6   inline int igraBonus( string imeIPrezime ) {
7       srand( time( 0 ) );
8       int bonus = 1 + rand() % 100;
9       return bonus;
10  }
11
12  int main() {
13      string imeIPrezime;
14      for( int i = 1; i <= 3; i++ ) {
15          cout << "Vnesete go vaseto ime i prezime: ";
16          getline( cin, imeIPrezime );
```

Слика 2.5.25

```
17          cout << "\tPocituvan/a " << imeIPrezime
18              << " vasiot bonus za ovaa igra e "
19              << igraBonus( imeIPrezime ) << endl;
20      }
21
22      cout << endl;
23      system( "Color 17" );
24      system( "pause" );
25      return 0;
26  }
```

Слика 2.5.25 продол ение

Еден излез од програмата е:

```
Vnesete go vaseto ime i prezime: Aleksandar Makedonski
        Pocituvan/a Aleksandar Makedonski vasiot bonus za ovaa igra e 41
Vnesete go vaseto ime i prezime: Car Samoil
        Pocituvan/a Car Samoil vasiot bonus za ovaa igra e 67
Vnesete go vaseto ime i prezime: Goce Delcev
        Pocituvan/a Goce Delcev vasiot bonus za ovaa igra e 90

Press any key to continue . . .
```

Задачи за вежбање

Да се напишат апликации и функции за следниве задачи:

1. Да се пресмета сумата $S_n = a_1 + (a_1 + a_2) + (a_1 + a_2 + a_3) + ... + (a_1 + a_2 + ... + a_n)$. Да се користи функција за пресметување на сумата $S_k = a_1 + a_2 + ... + a_k$.
2. Да се пресмета збирот $S_n = \frac{1}{1!} + \frac{1}{2!} + \frac{1}{3!} + \frac{1}{4!} + \cdots + \frac{1}{n!}$, со користење на функција за пресметување k!.
3. Да се напишат посебни функции за пресметување на бројот и на збирот на цифрите на природниот број n.
4. Да се најдат простите броеви помали од n. Да се напише функција за одредување дали некој природен број е прост или не е.
5. Да се напише функција од типот void за конверзија на реален декаден број во бинарен.
6. Да се напише функција од типот void за решавање на квадратната равенка $ax^2 + bx + c = 0$.
7. Да се напише функција од типот void за наоѓање на бинарниот, октални от и хексадецималниот запис на декадниот број n.
8. Да се напише функција од типот void за наоѓање на најголемиот и најмалиот број кои можат да се формираат од цифрите на природниот број n.

9. Да се најдат сите трицифрени природни броеви кои се еднакви на збирот на факториелите на своите цифри. За пресметување на факториел на цифра, да се користи посебна функција.
10. Да се најдат сите пријателски броеви во интервалот од m до n. (За два броја велиме дека се пријателски ако збирот на делителите на првиот број е еднаков на вториот број, а збирот на делителите на вториот број е еднаков на првиот број). За збир на делители на број да се напише посебна функција.

Прашања за проверка на знаењето

1. Како се нарекува библиотеката со функции во C++?
2. Наведете некоја хедер-датотека од библиотеката на C++.
3. Што треба да се вклучи во програмата ако сакаме да користиме функција од некоја хедер-датотека?
4. Во која библиотека на C++ се наоѓаат функциите за читање податоци внесени преку тастатурата?
5. Со која библиотечна функција се пресметува x^n, а со која $\sqrt{x}$?
6. Која е функцијата во C++ за генерирање случаен број?
7. Со која функција се претвораат сите букви од еден збор во големи?
8. Со која функција се проверува дали некој знак е цифра, а со која дали е буква?
9. Со кој оператор се одредува големината на меморијата што ја зафаќа некоја променлива?
10. Која е предноста на пишување делови од алгоритмите во подалгоритми?
11. Какви може да бидат подалгоритмите? Која е нивната главна карактеристика?
12. Како се повикува функциски подалгоритам, а како процедурален?
13. Што значи чекорот

    врати ←100;

    ако е напишан во функциски подалгоритам?
14. Какви видови функции постојат во C++, според тоа дали враќаат една или повеќе вредности?
15. Напишете општа форма на дефиниција на функција со повратна вредност.
16. Како се врши декларирање на функција со повратна вредност? Напишете општа форма.
17. Што е тип на функција?
18. Која е улогата на функцискиот прототип?
19. Од што се состои функцискиот потпис?
20. Дали за функциите дефинирани пред функцијата main() треба да се наведе функциски прототип?
21. Наведете неколку примери на функциски прототипови.
22. Наведете неколку примери на функциски потпис.

23. На кои начини може да се врши повикување на функциите со повратна вредност?
24. Што ќе се случи ако бројот на аргументите во повикот на функција со повратна вредност е различен од бројот на параметрите во дефиницијата на функцијата?
25. Што се случува ако типот на некој аргумент во повикот на функција не се совпаѓа со типот на соодветниот параметар во дефиницијата на функцијата?
26. Каде е грешката во следниов програмски сегмент:

```
int f(void) {
   cout << "Povik na funkcijata f()";
   int g(void) {
          cout << "Povik na funkcijata g()";
   }
}
```

27. Напишете функција со повратна вредност која за внесена страна го пресметува волуменот на коцка.
28. Дали може функција со повратна вредност да се повика во наредбата за печатење cout?
29. Какви параметри може да имаат функциите без повратна вредност?
30. Од кој тип се функциите во C++ кои не враќаат вредност?
31. Дали функциите без повратна вредност може да ја содржат наредбата return?
32. На кои начини може да се врши повикување на функции со повратна вредност, а на кои функции без повратна вредност?
33. Објаснете како се пренесуваат во функција аргументи по вредност, а како по референца.
34. Како се означуваат референтните параметри?
35. Дали во прототип на функција без повратна вредност мора да се означат референтните параметри?
36. Која е разликата меѓу аргумент пренесен по референца и соодветниот референтен параметар?
37. Дали функција со повратна вредност може да има референтни параметри?
38. Во кои случаи се користат функции со повратна вредност, а во кои функции без повратна вредност?
39. Каде е грешката во следнава функција без повратна вредност?

```
void P(int a) {
   int b;
   b = a++;
   return b;
}
```

40. Што ќе се случи ако се обидеме да ја промениме вредноста на константен параметар во функцијата?
41. Може ли една функција да повика друга функција?
42. Дали може во една функција да се дефинира друга функција?

43. Што се глобални, а што се локални променливи?
44. Кое е подрачјето на видливост на глобалните променливи, а кое на локалните променливи?
45. Какви променливи се имињата на функциите во една апликација, глобални или локални?
46. Кое е подрачјето на видливост на параметрите на функција?
47. Објаснете како може некоја глобална променлива да се скрие (да не е достапна) во некоја функција? Дали има начин, сепак, да се види?
48. Што е живот на променлива?
49. Која е разликата во однесувањето на глобална променлива и на статичка променлива?
50. Со кој збор се означуваат вградените функции?

Задачи

Да се пререшат сите примери и решени задачи од точката **2.3 Алгоритамски контролни структури за повторување и контролни наредби за повторување**, со користење на функции.

За следните задачи да се напишат програми со користење на функции:

1. Да се најде колку има големи букви, мали букви и интерпункциски знаци во една реченица.
2. Да се претворат Декартовите координати на точката (x, y) во поларни (ρ, φ), по формулите: $\rho = \sqrt{x^2 + y^2}$, $\varphi = arctg\frac{y}{x}$.
3. Да се најде НЗД за два броја.
4. Да се најде бинарниот еквивалент на цел негативен декаден број.
5. Да се напише функција со која ќе се внесат n броеви и ќе се најдат најмалиот и најголемиот од нив.
6. Да се најдат поголемиот и помалиот прост број од зададен природен број n.
7. Да се напише функција за собирање на две времиња, изразени преку часови (0 – 23), минути (0 – 59) и секунди (0 – 59).
8. Да се напише функција од типот void за пресметување на возраста на човек (разликата од денешниот датум и датумот на раѓање). Датумите да се внесуваат како целобројни променливи: den (1 – 31), mesec (1 – 12), godina (1 – 2345).

Термини

- **Алгоритамски контролни (управувачки) структури** служат за контрола на редоследот на извршување на алгоритамот.
- **Редоследната алгоритамска контролна структура** или **секвенца** е алгоритамска структура во која алгоритамските чекори се извршуваат по оној редослед по кој се наведени.
- Редоследната алгоритамска контролна структура се нарекува **почеток–крај.**
- **Алгоритамската контролна структура за избор од две можности** е структура за избор на една од две можни насоки на продолжување на дејството во алгоритамот, во зависност од некој услов (логички израз).
- **Алгоритамски блок** е блок од повеќе чекори означен со зборовите **почеток** и **крај**.
- **ако–тогаш–инаку** е алгоритамска контролна структура за избор од две можности.
- **ако–тогаш** е алгоритамска контролна структура за избор од две можности, при што една од можностите нема извршен чекор.
- **if** е контролна наредба во C++ за реализирање на алгоритамската контролна структура **ако–тогаш.**
- **if-else** е контролна наредба во C++ за реализирање на алгоритамската контролна структура **ако–тогаш–инаку.**
- **?:** се нарекува условен оператор.
- **Алгоритамската контролна структура за избор од повеќе можности** е структура во која се врши избор на еден од повеќе можни случаи на продолжување на дејството во алгоритамот, во зависност од вредноста на некој податок или на некој аритметички израз.
- **случај** е алгоритамска контролна структура за избор од повеќе можности.
- **switch** е контролна наредба во C++ за реализација на алгоритамската контролна структура за избор од повеќе можности **случај**.
- **Циклус** е едно извршување на група алгоритамски чекори.
- **Циклично повторување** е извршувањето на иста група алгоритамски чекори повеќепати.
- **за–до–чекор** е алгоритамска контролна структура во која циклусите се бројат од pocetna до krajna вредност со некој iznos на чекорот.
- Ако во алгоритамската контролна структура **за–до–чекор** iznosot е +1, структурата се нарекува **за–зголемувај–до.**
- Ако во алгоритамската контролна структура **за–до–чекор** iznosot е –1, структурата се нарекува **за–намалувај–до.**
- **for** е контролна наредба во C++ за реализација на контролните структури **за–зголемувај–до, за–намалувај–до** и **за–до–чекор**, со инкрементирање на

бројачот по 1, декрементирање на бројачот по 1 и зголемување/намалување на бројачот за одреден iznos.

- ++ е оператор за инкрементирање.
- – – е оператор за декрементирање.
- **додека–извршувај** е алгоритамска контролна структура во која повторувањето на циклусите се врши додека е исполнет некој услов, кој се испитува пред почетокот на секој циклус.
- **while** е контролна наредба во C++ за реализација на алгоритамската контролна структура **додека–извршувај.**
- **извршувај–додека** е алгоритамска контролна структура во која повторувањето на циклусите се врши додека е исполнет некој услов, кој се испитува на крајот од секој циклус.
- **do-while** е контролна наредба во C++ за реализација на алгоритамската контролна структура **извршувај–додека.**
- **продолжи** е алгоритамска контролна структура за скок на крајот на циклусот и продолжување со следниот циклус.
- **continue** е контролна наредба во C++ за реализација на алгоритамската контролна структура **продолжи.**
- **прекин** е алгоритамска контролна структура за скок на крајот на контролната структура и прекинување на повторувањето.
- **break** е контролна наредба во C++ за реализација на алгоритамската контролна структура **прекин.**
- **излез** е алгоритамска контролна структура за скок на крајот на алгоритамот.
- **exit()** е контролна наредба (функција) во C++ за реализација на алгоритамската контролна структура **излез.**
- **скок** е алгоритамска контролна структура за скок на произволен чекор во алгоритамот.
- **goto** е контролна наредба во C++ за реализација на алгоритамската контролна структура **скок.**
- **Стандардната библиотека на C++** е библиотека која содржи математички функции, функции за влез и излез, за операции со знаци, со стрингови итн.
- **Функции со повратна вредност** се оние функции кои по извршувањето враќаат вредност на местото од каде што се повикани.
- **Функции без повратна вредност** се оние функции кои по извршувањето не враќаат вредност на местото од каде што се повикани.
- **Програмирање одгоре надолу** е техника за расчленувања на задачата на поедноставни задачи, наречени **подзадачи**.
- **Модуларно програмирање** е техника за изработка програми (модули) за подзадачите и нивно секвенцијално извршување.
- **Подалгоритам** е алгоритам кој не може да се извршува (по кодирање) самостојно.

- **Функциските подалгоритми** наречени и **функции** имаат само ***еден излезен резултат***.
- **Процедуралните подалгоритми** наречени и **процедури** имаат ***повеќе излезни резултати***.
- **Параметри (формални аргументи)** се користат во дефиницијата на подалгоритмите.
- **Аргументи (вистински аргументи)** се променливи кои се користат при повик на подалгоритам.
- **Влезни параметри** (**влезни формални аргументи**) се оние преку кои се пренесуваат вредности од повикувачот (алгоритам или подалгоритам) во подалгоритамот. Тие се означуваат со знакот ↓.
- **Излезни параметри** (**излезни формални аргументи**) се оние преку кои се пренесуваат вредности од подалгоритамот до повикувачот (алгоритам или подалгоритам). Тие се означуваат со знакот ↑.
- **Влезно-излезни параметри** (**влезно-излезни формални аргументи**) се оние преку кои се пренесуваат вредности од повикувачот (алгоритам или подалгоритам) во подалгоритамот и спротивно. Се означуваат со знакот ↕.
- **врати** е чекор со кој се враќа резултатот од функциски подалгоритам.
- **Листата на параметри** е листа во насловот на дефиниција подалгоритамот.
- **Листата на аргументи** е листа во повикот на подалгоритамот.
- **Функциска потпрограма** наречена **функција** е програма во програмски јазик напишана за функциски подалгоритам.
- **Кориснички дефинирани функции** или само **кориснички функции** се функции напишани од програмерите.
- **Дефиниција на функција** во C++ се состои од наслов (кој се состои од тип на повратната вредност, име на функцијата и листа на параметри) и тело (во големи загради).
- **Декларација на функција** со повратна вредност во C++ се состои од тип (на повратната вредност), име (на функцијата) и листа на параметри.
- **Функциски прототип** се нарекува декларацијата на функција, а се состои од тип, име и листа на параметри.
- **Функциски потпис** се состои од име на функцијата и листа на параметрите.
- **Константни параметри** се оние чија вредност не може да се менува во функцијата.
- **void** е тип на функции во C++ кои не враќаат вредност.
- **Аргументи пренесени по вредност** се оние кои се пренесуваат во функција со повик, но нивната вредност во повикувачот (функцијата main() или друга функција) не се менува. Механизмот се нарекува **пренесување на аргументи по вредност**.
- **Референцирање** е доделување друго име на променлива.
- **Референца** или **референтна променлива** е друго име на некоја променлива.

- **Аргументи пренесени по референца** се оние кои се пренесуваат во функција со повик, при што тие се друго име на соодветните референтни параметри. Механизмот се нарекува **пренесување на аргументи по референца**.
- **Подрачје на видливост** наречено и **подрачје на пристап** е она подрачје во кое има пристап до променливата, т. е. може да се вршат операции со неа.
- **Глобални променливи** се оние до кои има пристап од целата апликација, т. е. од главната функција и од сите кориснички дефинирани функции во истата датотека во која е главната функција.
- **Локални променливи** се оние кои имаат подрачје на пристап само телото на функцијата во која се декларирани или само насловот на функцијата или само блокот во кој се декларирани.
- **Скриена променлива** (прекриена со друга променлива) во некој блок или во цела функција е онаа која не е видлива (не може да се користи) во тој блок или во таа функција.
- **::** е **оператор за разрешување на подрачјето на видливост.**
- **Живот на променлива** е времето од креирањето во меморијата (веднаш по извршување на наредбата за декларација) до нејзиното изчезнување (крај на блокот, крај на функцијата или крај на апликацијата).
- **Статичка променлива** е онаа која од моментот на декларација се однесува како глобалан променлива.
- **Вградена функција** е онаа која се преведува заедно со повикувачот (main() или друга функција) и се вградува во неговиот код.
- **inline** е квалификатор за вградени функции.

Резиме

- За опис на текот на алгоритмите при структурираното програмирање, се користат посебни т.н. алгоритамски контролни (управувачки) структури со кои се управува со редоследот на извршување на алгоритамските чекори.
- Текот на секој алгоритам може да се контролира со користење на следниве алгоритамски контролни структури: **редоследна** или **секвенца**, **избор** или **селекција** и **повторување** или **итерација**.
- Во редоследната алгоритамска контролна структура чекорите се извршуваат по редослед како што се наведени. Таа се нарекува **почеток–крај**.
- **ако–тогаш–инаку** е алгоритамска контролна структура за избор од две можности, а во C++ се реализира со контролната наредба if-else.
- **ако–тогаш** е алгоритамска контролна структура за избор од две можности кога една од можностите не содржи извршен чекор, а во C++ се реализира со контролната наредба if.

- Контролните наредби if и if-else може да се вгнездуваат и во изборот if и во изборот else. Вгнездената контролна наредна во изборот else се нарекува if-else-if.
- Условниот оператор ?: се користи за претставување на едноставни наредби if-else.
- Кога има повеќе можности за продолжување на дејството во алгоритамот, се користи алгоритамската контролна структура за избор од повеќе можности **случај**. Изборот на една од можностите се врши, во зависност од вредноста на некој податок или на некој аритметички израз.
- Алгоритамската контролна структура за избор од повеќе можности **случај**, во C++ се реализира со контролната наредба switch**.**
- Алгоритамските контролни структури за повторување се користат тогаш кога е потребно една група алгоритамски чекори да се извршат повеќепати, што се нарекува циклично повторување. Едно извршување на чекорите се нарекува еден циклус.
- Разликуваме три алгоритамски контролни структури за повторување, и тоа: повторување со броење на циклусите (**за–до–чекор**), повторување со излез на почетокот од циклусот (**додека–извршувај**) и повторување со излез на крајот од циклусот (**извршувај–додека**).
- Во алгоритамската контролна структура **за–до–чекор**, циклусите се бројат со посебен brojac, со некој iznos на чекорот од pocetna до krajna вредност. Ако iznos на чекорот е +1, структурата се нарекува **за–зголемувај–до,** ако iznos е –1, структурата се нарекува **за–намалувај–до**.
- За реализација на алгоритамската контролна структура **за–до–чекор** (**за–зголемувај–до** и **за–намалувај–до**), во C++ се користи контролната наредба for.
- Контролната наредба for има дел за иницијализација, дел за услов и дел за ажурирање.
- Контролната наредба for во C++ дозволува иницијализација и ажурирање на повеќе променливи.
- Во контролната наредба for делот за иницијализација и ажурирање може да биде и празен.
- Алгоритамската контролна структура **додека–извршувај**, во C++ се реализира со контролната наредба while. Додека е исполнет некој логички услов во while, циклусите се извршуваат. Условот се испитува пред почетокот на секој циклус.
- Со контролната наредба while може да не се изврши ниту еден циклус ако условот не е исполнет уште пред започнување на првиот циклус.
- Контролирање на повторувањето со некоја однапред позната вредност се нарекува *повторување контролирано со стражар.*

- Алгоритамската контролна структура **извршувај–додека**, во C++ се реализира со контролната наредба do-while. Циклусите во do-while се извршуваат додека е исполнет некој логички услов. Условот се испитува по завршување на секој циклус.
- Со наредбата do-while, мора да се изврши барем еден циклус.
- При вгнездување на контролните наредби за повторување (for, while, do-while), сите наредби на едната контролна наредба мора да се наоѓаат во другата, односно не смее да има сечење (преклопување) на контролните наредби.
- Во програмските јазици постојат четири вида алгоритамски контролни структури за скок, и тоа: скок на крајот на циклусот – **продолжи**, скок на крајот на контролната структура – **прекин**, скок на крајот на алгоритамот – **излез** и скок на произволно место – **скок**.
- Алгоритамските контролни структури за скок, во C++ се реализираат со контролните наредби: continue, break, exit() и goto.
- Се избегнува користењето на контролната наредба goto во структурираното програмирање.
- Во структурираното програмирање се користат две техники за програмирање, и тоа: програмирање одгоре надолу и модуларно програмирање. Програмирањето одгоре надолу се врши со разделување (расчленување) на задачата на подзадачи. За секоја подзадача може да се напише посебен алгоритам, кој се нарекува подалгоритам.
- Суштината на подалгоритмите е во тоа што тие може да се користат во различни алгоритми или подалгоритми, а и на повеќе места во ист алгоритам и/или подалгоритам.
- Користењето на подалгоритмите е овозможено преку механизмот на параметри (формални аргументи) и аргументи (вистински аргументи).
- Параметрите може да бидат влезни, излезни и влезно-излезни.
- Според бројот на излезни резултати, постојат два вида подалгоритми, и тоа: функциски подалгоритми (кои враќаат само еден резултат) и процедурални подалгоритми (кои може да вратат повеќе резултати).
- Подалгоритмите во програмските јазици се реализираат со потпрограми. Потпрограмите може да бидат функциски (кога враќаат една вредност) и процедурални (кога враќаат повеќе вредности).
- Потпрограмите во C++ се реализираат со функции. Функциските потпрограми се реализираат со функции со повратна вредност, а процедуралните потпрограми се реализираат со функции без повратна вредност.
- Функциите во C++ може да бидат: библиотечни функции и кориснички дефинирани функции.
- При дефинирање на функција со повратна вредност во C++, треба да се знае дека:

  - насловот на функцијата не завршува со **;** (точка и запирка),
  - листата на параметри мора да ги содржи и имињата и типовите на параметрите,
  - повратната вредност е резултатот од функцијата кој може да биде израз (од променливи и/или константи и/или функции) или една променлива и кој/а мора да е од ист тип со типот на функцијата.
- За декларација на функција во C++, треба да се знае:
  - Ако не се наведе типот на функцијата, се подразбира типот int.
  - Имињата на функциите (по конвенција) се пишуваат со мала почетна буква. Ако името се состои од повеќе зборови, секој следен збор се пишува со голема буква.
  - Листата на параметри мора да ги содржи типовите на параметрите, а имињата не мора.
  - Декларацијата на функцијата завршува со знакот ; (точка и запирка).
- При повик на функција, листата со аргументи и листата со параметри на функцијата мора да имаат:
  - ист број на аргументи и параметри,
  - ист редослед на аргументите и параметрите,
  - ист или конвертибилен тип на соодветните аргументи и параметри.
- Функција со повратна вредност во C++ се повикува со името и листата на аргументи или со наредба за доделување или во друга наредба или израз.
- Функција без повратна вредност во C++ се повикува само со нејзиното име и листата на аргументи.
- Функциите кои се дефинирани по главната програма, се декларираат (со наведување на нивниот прототип) пред главната функција main().
- Една функција може само еднаш да се дефинира во истата програма.
- За да се заштитат параметрите во функцијата од измени во неа, тие се означуваат како константни со квалификаторот const.
- Функциите без повратна вредност во C++ имаат тип **void**.
- Од функција со повратна вредност се излегува со наредбата return со параметар, додека од функција без повратна вредност се излегува автоматски по нејзиното извршување или од кое било место во телото на функцијата со наредбата return без параметар.
- При повик на функција, се користи механизам за пренесување на аргументите по вредност или по референца.
- При пренос на аргументите по вредност, тие не се менуваат во функцијата, т. е. остануваат исти и по враќање во повикувачот (главната функција main() или друга функција).
- При пренос на аргументите по референца, тие може да се менуваат во функцијата и по враќање во повикувачот (главната функција main() или друга функција) може да бидат променети.

- При пренос на аргументи по референца, не се креираат нови променливи за аргументите, туку аргументите од повикот претставуваат други имиња на референтните параметри на функцијата. Сите измени во референтните параметри, всушност, се вршат врз соодветните аргументи бидејќи тие се различни имиња на иста мемориска локација.
- Ако не сакаме да се менува некој аргумент во функцијата, независно дали е пренесен по вредност или по референца, треба да се означи како константен со квалификаторот const.
- Функциите со повратна вредност може да бидат аргументи на други функции.
- Глобални променливи се оние кои се достапни во целата апликација и во функциите кои се повикуваат во неа.
- Имињата на кориснички дефинираните функции во една апликација се третираат како глобални променливи.
- Локални променливи се оние кои се достапни (видливи) и може да се користат само во функцијата или блокот во кој се декларирани.
- Видливоста на локална променлива започнува со нејзината декларација.
- Параметрите на функција се третираат како локални променливи во целата функција.
- Променливите декларирани во еден блок (функција, тело на наредба – if, switch, for, while, do-while и др.) може да се користат само во него бидејќи се локални променливи.
- Променлива декларирана во една функција не може да се користи во друга функција.
- Две функции со исто име може да имаат исто подрачје на видливост само ако имаат различни листи на параметри.
- При извршување на една апликација, за сите функции се прави по една копија во меморијата.

МОДУЛ

СЛОЖЕНИ ТИПОВИ НА ПОДАТОЦИ ВО C++

3

Во оваа глава ќе се запознаете со следното:

- Низи и нивното значење во програмирањето.
- Декларација, иницијализација и доделување вредности на елементите на еднодимензионалната низа.
- Наредба for базирана на опсег и нејзиното користење.
- Декларација, иницијализација и доделување вредности на дводимензионалната низа.
- Пресметување димензии на дводимензионалната низа.
- Покажувачи и нивната важност во програмирањето.
- Користење на адресниот оператор & и операторот за дереференцирање *.
- Врска помеѓу низите и покажувачите.
- Користење на наредбите new и delete.
- Функции за работа со стрингови.
- Функции за конверзија на стринг во број и за конверзија на број во стринг.

Клучни зборови

#define.	Именувана константа
<string>	Индиректен оператор (оператор за дереференцирање)*
delete	Матрица
new	Иницијализирачка листа
stoi, stol, stoll, stoul, stoull, stof, stod, stold	Наредбата for базирана на опсег
to_string()	Низа
Адресен оператор &	Покажувач
Дводимензионална низа	Покажувачка променлива
Динамичка променлива	Симболичка константа
Еднодимензионална низа	

3.1 Еднодимензионални низи

Постојат проблеми за чие решавање е потребно да се воведат голем број променливи. На пример, за да се пресмета просечниот успех на еден ученик, потребно е да се внесат оценките по сите предмети и за секоја оценка да се воведе по една целобројна променлива, како

ocenka_1 ocenka_2 . . . ocenka_n

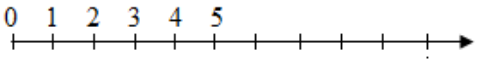
Ако претпоставиме дека има 12 предмети, потребни се 12 променливи за оценките на еден ученик. Ако во класот има 30 ученици, тогаш се потребни 12 x 30 = 360 променливи за оценките на сите ученици. Програма со толкав број променливи не само што е непрегледна туку ќе биде и многу долга. При нејзиното пишување, повеќе време ќе се потроши за пишување на имињата на променливите отколку за самата апликација.

Во ваквите случаи, кога се работи со голем број податоци од ист тип, тие се организираат во посебни структури наречени **низи** (англ. arrays). Елементите на низата може да бидат од кој било тип, но сите елементи мора да се од ист тип. На пример, сите оценки на учениците во низата именувана со ocenka, во наведениот пример, мора да се од целоброен тип.

Секој елемент во низата има свој реден број. На пример, во низата ocenka, имињата на елементите се: ocenka_1, ocenka_2… ocenka_n. Тие се разликуваат само по редните броеви 1, 2, 3... n. Ваквите величини во математиката се запишуваат на тој начин што редните броеви се стават како индекси: $ocenka_1$, $ocenka_2$... $ocenka_n$. Индексот го означува редниот број на елементот во низата: 1 – првиот елемент, 2 – вториот елемент итн. n го означува n-тиот елемент.

Да се потсетиме за аритметички и геометриски низи. Низата a_1, a_2... a_n се нарекува аритметичка ако разликата на кои било два соседни елементи е иста, а кај геометриската низа количникот на кои било два соседни елементи е ист. Исто така, гледаме дека сите елементи имаат исто име, а индексите им се 1, 2, 3... n.

Индексите може да се означат како точки на бројна оска, т. е. во еден правец, една димензија. Должината е една димензија. Површината (правоаголник) има должина и ширина, значи две димензии. Просторот (зграда) има три димензии: должина, ширина и висина.



Затоа, низите со еден индекс (една димензија) се нарекуваат **еднодимензионални низи** (англ. one dimensional arrays).

Ако знаеме дека низата има n елементи, тогаш се запишува со: a_1, a_2... a_n. Или, пократко со $a[_i]_n$ или само $a[]_n$, што значи дека индексот i има вредности од 1 до n. Се користи и следната ознака $[a]_n$, при што со n се означува дека низата е еднодимензионална и дека индексите се од 1 до n. Во литературата се користи и ознаката a[1..n]. Последнава ознака ќе ја користиме во алгоритмите.

Во програмските јазици, променливите што добиваат вредности на податоци од еднодимензионална низа се запишуваат на тој начин што индексот се става во средни загради: a[1], a[2]... a[n] или ocenka[1], ocenka[2]... ocenka[n]. Иако името на сите елементи им е исто, тие се различни променливи кои се разликуваат по индексот.

Во многу програмски јазици индексите почнуваат од 0 до n – 1.

Низите во C++ се означуваат на истиот начин, при што индексите се ненегативни цели броеви, а првиот индекс секогаш е 0. За низа од n елементи, индексите се 0, 1, 2, 3... n – 1. На пример, за низа од n елементи, ознаката е $a[]_n$, а елементите се: a[0], a[1]... a[n – 1]. Ако сакаме да ја означиме само низата, без бројот на елементи, тогаш ќе користиме ознака a[] за да се разликува од променлива со исто име.

Декларација на еднодимензионални низи

Еднодимензионалните низи како променливи во C++ се декларираат на следниов начин

тип име[*број*];

тип – е типот на елементите на низата (short, int, long, long long, float, double, char, boolean, string итн.).
име – е името на низата.
број – е бројот на елементи на низата и тој мора да биде литерал или именувана константа поголема од 0.

Примери

Пример 3.1.1

```
int a[5];
```

Низата a[] има 5 елементи и тоа: a[0], a[1], a[2], a[3] и a[4]. Типот на елементите на низата е int и на нив може да им се доделуваат само целобројни вредности.

Пример 3.1.2

```
float br[25];
```

Низата br[] е од типот float и има 25 елементи. Општиот k-ти елемент се означува со br[k], а индексот k може да има вредности 0, 1... 24. Значи, елементите се: br[0], br[1],..., br[24].

Пример 3.1.3

```
string ime[15], prezime[30];
```

Се декларираат две низи ime[] и prezime[] чии елементи може да имаат вредности само стрингови.

Пример 3.1.4

```
const int BROJELEMENTI = 100;
char c[BROJELEMENTI];
```

Во примеров, бројот на елементи на знаковната низа c[] е претходно зададен како **именувана константа** (англ. named constant), наречена и **константна променлива** (англ. constant variable), `BROJELEMENTI`.

Овој начин на задавање на бројот на елементи на низа се користи најчесто бидејќи именуваната константа се иницијализира еднаш и не може да се промени понатаму во апликацијата.

Пример 3.1.5

```
int k = 10;
float u[k];
```

Оваа декларација е погрешна бидејќи бројот на елементи мора да е константен.

За да означиме специфичен елемент на низата, ги задаваме името на низата и позицијата на тој елемент во низата. Во ***табела .1*** е дадена целобројна низа со име d. Оваа низа се состои од 10 елементи. До секој од овие елементи може да се пристапи преку името на низата и индексот на елементот во низата поставен во средни загради [].

Така, првиот елемент е d[0], а i-тиот елемент е d[i – 1].

0 е индекс на првиот елемент на низата.

9 е индекс на последниот елемент на низата.

Елемент	Вредност на елементот
d[0]	–567
d[1]	8
d[2]	9
d[3]	72
d[4]	345
d[5]	–78
d[6]	78
d[7]	9
d[8]	10
d[9]	987

Табела .1.1

Индексот на елементите на низа може да биде константа, променлива или израз чија вредност е ненегативна и во опсег на индексите од 0 до n – 1 (ако должината на низата е n). Така, за a = 7 и b = 2 елементот

```
d[a + b - 1]
```

е, всушност, елементот d[8].

Иницијализација на низа и доделување вредности на елементите

На елементите на низа може да им се доделуваат вредности со наредбата за доделување или при иницијализација.

Во наредните примери се илустрирани различни начини на доделување вредности на елементите на низа.

Пример 3.1.6

Доделување вредности може да се врши посебно на секој елемент:

```
int a[10];
a[0] = 1; a[9] = 10; a[3] = -4; a[5] = 5;
```

Пример 3.1.7

Доделувањето вредности може да се врши и преку изрази во кои се пресметува вредноста на индексите на низата. На пример, ако елементот a[5] = 24, со наредбите:

```
int c = 3;
int d = 2;
a[c + d] += 6;
```

на елементот a[5] му се зголемува вредноста за 6, т. е. неговата вредност сега ќе биде 30.

Пример 3.1.8

Доделување вредности на елементите на низа може да се врши и при декларирањето, т. е. со ***иницијализирачка листа*** (англ. initializer list):

```
 int a[] = {5, -2, 7, -3, 6};
```

Вредностите на елементите ќе бидат: a[0] = 5, a[1] = –2, a[2] = 7, a[3] = –3, a[4] = 6. Должината на низата (бројот на елементи) при ваква иницијализација ја одредува самиот преведувач.

По декларацијата и иницијализацијата на низата

```
char bukvi[] = {'a', 'b', 'c'};
```

вредностите на елементите ќе бидат: a[0] = 'a', a[1] = 'b', a[2] = 'c'.

Пример 3.1.9

Ако има помалку вредности во иницијализирачката листа отколку што има елементи во низата, останатите елементи се иницијализираат на нула.

Со следнава декларација и иницијализација

```
int nekoi[10] = {3, -1, 4};
```

елементите на низата ќе се иницијализираат со следните вредности: 3, –1, 4, 0, 0, 0, 0, 0, 0, 0. Нулите ги пополнува преведувачот.

Слично, со

```
int nuli[100] = {};
```

се декларира низата од целобројни елементи под име nuli, при што сите елементи се иницијализираат на 0.

Со декларацијата и иницијализација

```
char samoglaski[5] = {'a'};
```

експлицитно се доделува вредноста `'a'` на нултиот елемент од низата, а сите останати елементи автоматски се иницијализираат на празни места.

Забелешка: Во C++ нема автоматска иницијализација на низи. Мора барем еден елемент да е иницијализиран за потоа сите останати елементи да бидат иницијализирани.

Пример 3.1.10

Димензијата на низата може да се пресмета автоматски ако елементите се зададени при иницијализација. При следната декларација и иницијализација

```
double ovie[] = {2.34, -5.67, 3.45, 7.89};
```

не е зададена должината на низата, а ја одредува преведувачот, според зададениот број на елементи, т. е. низата ovie ќе има должина 4.

Слично, со декларацијата и иницијализација

```
string denovi[] = {"Pon", "Vto", "Sre", "Cet", "Pet", "Sab", "Ned"};
```

должината на низата denovi е 7.

Со следнава декларација и иницијализација

```
int cifri[] = {1, 2, 3, 4, 5};
```

се декларира низа со должина 5 и се иницијализираат сите елементи.

Од друга страна, со следнава декларација и иницијализација на низата

```
int v[5] = {1, 2, 4, 5, 6, 9};
```

се јавува синтаксичка грешка бидејќи има 6 иницијализирачки вредности, а низата може да има најмногу 5 елементи.

Пример 3.1.11

Низите не може:

- да се доделуваат една на друга:

```
int a[2] = {1,2};
int b[2];
b = a;                          // neispravno
```

- да се иницијализираат една со друга:

```
int c[2] = a;                   // neispravno
```

- да се печатат само преку името[1]:

```
cout << a;                      // neispravno
```

- да се споредуваат[2]:

```
if(a == b)                      // neispravno
```

- да бидат повратна вредност од функција[3].

```
return a;                       // neispravno
```

[1] Се печати адресата на низата.
[2] Се споредуваат адресите на низите.
[3] Се враќа покажувач на низата. За покажувачи ќе зборуваме подоцна.

Специфицирање на должината на низата може да се врши и со **симболичка константа**. Симболичката константа се задава со претпроцесорската директива **#define.**

На пример,

```
#define PI 3.1415;
```

Пример 3.1.12

Во овој пример се користи претпроцесорската директива **#define**, со која се дефинира симболичката константа DOLZINA чија вредност е 10, ***слика .1.1***.

Програмата ги иницијализира првите два елементи на низата со вредност 1, а останатите со 0. Со наредбата for, на елементите од третиот до десеттиот им се доделуваат вредности по формула. (Која?)

```
#include <iostream>
#include <iomanip>
using namespace std;

#define DOLZINA 10

int main() {    //Koristenje simbolicka konstanta

    int z[ DOLZINA ] = { 1, 1 };//Inicijalizacija na prvite
                                //dva elementi so 1, a ostanatite so 0

    // Dodeluvanje vrednosti na elementite od tretiot do desettiot
    int j;
    for( j = 2; j < DOLZINA; j++ )
        z[ j ] = z[ j - 2 ] + z[ j - 1 ];

    // Pecatenje na nizata z[] vo tabelaren format
    cout << setw( 10 ) << "Element" << setw( 10 ) << "Vrednost" << endl;
    for( j = 2; j < DOLZINA; j++ )
        cout << setw( 6 ) << "z[" << j << "]" << setw( 8 ) << z[ j ] << endl;

    cout << endl;
    system( "Color 17" );
    system( "pause" );
    return 0;
}
```

Слика .1.1

Излезот по извршување на програмата е:

```
   Element  Vrednost
    z[0]        1
    z[1]        1
    z[2]        2
    z[3]        3
    z[4]        5
    z[5]        8
    z[6]       13
    z[7]       21
    z[8]       34
    z[9]       55

Press any key to continue . . .
```

Пример 3.1.13

Програмскиот сегмент за читање елементи на целобројна низа од n елементи $a[]_n$ е:

```
for(int i = 0; i < n; i++)
      cin >> a[i];
```

Пример 3.1.14

Програмскиот сегмент за печатење на елементите на низата $b[]_n$ е:

```
for(int i = 0; i < n; i++)
      cout << a[i] << endl;
```

Пример 3.1.15

Програмскиот сегмент за наоѓање на збирот на елементите на низата од n елементи $c[]_n$ е:

```
zbir = 0;
for(int i = 0; i < n; i++)
      zbir += c[i];
```

Истиот програмски сегмент може да се користи и за пресметување на производот на елементите на низа, само треба да се сменат наредбите:

```
zbir = 0;           co     proizvod = 1;
zbir += c[i];       co     proizvod *= c[i];
```

Овие сегменти може да се користат за која било операција со елементите на низа.

Пример 3.1.16

Во примерот на ***слика .1.2*** се илустрирани декларација, иницијализација и доделување вредности на елементите на низи од типот: int, float, char и string.

```cpp
#include <iostream>
#include <string>
#include <iomanip>
using namespace std;

int main() {    //  Deklaracija, inicijalizacija i dodeluvanje vrednosti
                //  na elementite na niza

    int dolzina;
    int a[ 4 ] = {};
    a[ 0 ] = 123; a[ 2 ] = 12345;
    dolzina = sizeof a / sizeof( a[ 0 ] );
    cout << "Celobrojna niza so dolzina " << dolzina << endl;
    cout << "index" << setw( 10 ) << "vrednost" << endl;
    for( int index = 0; index < dolzina; index++ )
        cout << setw( 3 ) << index << setw( 10 ) << a[ index ] << endl;

    float b [] = { -3.5f, 2.7f, 7.3f };
    b[ 1 ] = 123.45f;
    dolzina = sizeof b / sizeof( b[ 0 ] );
    cout << "\nRealna niza so dolzina " << dolzina << endl;
    cout << "index" << setw( 10 ) << "vrednost" << endl;
    for( int index = 0; index < dolzina; index++ )
        cout << setw( 3 ) << index << setw( 10 ) << b[ index ] << endl;

    char znaci [] = { '@', '#', '$', '%', '^' };
    znaci[ 1 ] = '['; znaci[ 2 ] = ']';
    dolzina = sizeof znaci / sizeof( znaci[ 0 ] );
    cout << "\nZnakovna niza so dolzina " << dolzina << endl;
    cout << "index" << setw( 10 ) << "vrednost" << endl;
    for( int index = 0; index < dolzina; index++ )
        cout << setw( 3 ) << index << setw( 10 ) << znaci[ index ] << endl;

    string iminja [] = { "Antonia", "Atanasia", "Mihaela", "Jana", "Jovana", "Teo" };
    dolzina = sizeof iminja / sizeof( iminja[ 0 ] );
    cout << "\nNiza od stringovi so dolzina " << dolzina << endl;
    cout << "index" << setw( 10 ) << "vrednost" << endl;
    for( int index = 0; index < dolzina; index++ ) {
        cout << setw( 3 ) << index << "\t";
        cout << setw( 10 ) << left << iminja[ index ] << right << endl;
    }

    cout << endl;
    system( "Color 17" );
    system( "pause" );
    return 0;
}
```

Слика .1.2

Излезот од програмата е:

```
Celobrojna niza so dolzina 4
index  vrednost
  0        123
  1          0
  2      12345
  3          0

Realna niza so dolzina 3
index  vrednost
  0       -3.5
  1     123.45
  2        7.3

Znakovna niza so dolzina 5
index  vrednost
  0          @
  1          [
  2          ]
  3          %
  4          ^

Niza od stringovi so dolzina 6
index  vrednost
  0      Antonia
  1      Atanasia
  2      Mihaela
  3      Jana
  4      Jovana
  5      Teo

Press any key to continue . . .
```

Во поднасловот **Библиотечни функции**, се запознавме со операторот sizeof, со кој може да се добие големината на меморијата зафатена од некој тип или од израз (променлива).

Операторот sizeof може да се примени и на низа, при што ќе се добие големината на низата во бајти, која зависи од бројот на елементи. За да го добиеме бројот на елементи на низата, треба големината на низата (во бајти) да ја поделиме со големината на типот на елементите на низата (во бајти).

На пример:

```
char znaci[] = {'@','#','$','%','^'};
dolzina = sizeof znaci / sizeof(char);
или
dolzina = sizeof znaci / sizeof(znaci[0]);   // znaci[0] e prviot element
```

Решени задачи

Задача 3.1.1

Да се напишат алгоритам и програма со која секој елемент од бројната низа $a[]_n$ ќе го промени знакот, + во – и – во +.

Алгоритамот и програмата се дадени во продолжение.

```
алгоритам ПроменаНаЗнацитеНаНиза
почеток
        a[1..n];
        печати „Внесете го бројот на елементи на низата, n = “;
        читај n;
        за i ← 1 зголемувај до n
                читај ai;
        крај_за {i}
        за i ← 1 зголемувај до n
                ai ← –ai;
        крај_за {i}
        печати „Низата со променети знаци на елементите е: “;
        за i ← 1 зголемувај до n
                печати ai;
        крај_за {i}
крај {ПроменаНаЗнацитеНаНиза}
```

Еден излез од извршување на програмата е:

```
Vnesete broj na elementi na nizata, n = 3
Vnesete gi elementite na nizata:
a[0] = 1
a[1] = -2
a[2] = 3

Vnesenata niza e:
a[0]=1
a[1]=-2
a[2]=3

Nizata so promeneti znaci e:
a[0]=-1
a[1]=2
a[2]=-3

Press any key to continue . . .
```

```
1   #include <iostream>
2   using namespace std;
3
4   int main() {
5       //  Da se promenat znacite na elementite na niza
6
7       const int m = 100;
8       double a[ m ] = {};
9       cout << "Vnesete broj na elementi na nizata, n = ";
10      int n; cin >> n;
11      cout << "Vnesete gi elementite na nizata: \n";
12      for( int i = 0; i < n; i++ ) {
13          cout << "a[" << i << "] = ";
14          cin >> a[ i ];
15      }
16
17      cout << "\nVnesenata niza e: " << endl;
18      for( int i = 0; i < n; i++ )
19          cout << "a[" << i << "]=" << a[ i ] << endl;
20      for( int i = 0; i < n; i++ )
21          a[ i ] = -a[ i ];
22
23      cout << "\nNizata so promeneti znaci e: " << endl;
24      cout << "\nNizata so promeneti znaci e: " << endl;
25      for (int i = 0; i<n; i++)
26          cout << "a[" << i << "]=" << a[i] << endl;
27
28      cout << endl;
29      system("Color 17");
30      system("pause");
31      return 0;
32  }
```

Слика .1.

Задача 3.1.2

Да се напише програма за пресметување на збирот
$+ a_1 - a_2 + a_3 - a_4 + \ldots (+ / -) a_n$.

Еден пример од извршување на програмата е:

```
Vnesete broj na elementi na nizata, n = 5
Vnesete gi elementite na nizata:
a[0] = 11
a[1] = -22
a[2] = -333
a[3] = 4444
a[4] = -55555

Vnesenata niza e: 11, -22, -333, 4444, -55555,
Zbirot na nizata: +(11)-(-22)+(-333)-(4444)+(-55555) e -60299

Press any key to continue . . .
```

```
#include <iostream>
using namespace std;

int main() {
    //  Da se najde zbirot a1 - a2 + a3 - . . . (+/-)an

    const int m = 100;
    double zbir = 0, a[ m ] = {};
    int i, k, n;
    cout << "Vnesete broj na elementi na nizata, n = ";
    cin >> n;
    cout << "Vnesete gi elementite na nizata: \n";
    for( i = 0; i < n; i++ ) {
        cout << "a[" << i << "] = ";
        cin >> a[ i ];
    }
    cout << "\nVnesenata niza e: ";
    for( i = 0; i < n; i++ )
        cout << a[ i ] << ", ";
    k = 1;
    for( i = 0; i < n; i++ ) {
        zbir += k * a[ i ];
        k = -k;
    }
    cout << "\nZbirot na nizata: ";
    for( i = 0; i < n; i++ ) {
        if( i % 2 == 0 )
            cout << '+' << "(" << a[ i ] << ")";
        else
            cout << '-' << "(" << a[ i ] << ")";
    }
    cout << " e " << zbir << endl;

    cout << endl;
    system( "Color 17" );
    system( "pause" );
    return 0;
}
```

Слика .1.4

Задача 3.1.3

Да се напише програма за наоѓање на најголемиот и на најмалиот елемент во бројната низа $a[]_n$, со посебни функции.

Објаснува е: На почетокот се зема дека најголем елемент е првиот, a_1. Потоа, се споредува вториот елемент a_2 со најголемиот (а тоа е првиот) и поголемиот од нив се зема за најголем. Понатаму, сите останати елементи a_3, a_4... a_n се

споредуваат со дотогаш најдениот најголем елемент и ако се најде поголем од него, се зема тој да биде најголем. Притоа, се памти и неговата позиција во низата.

Ќе наведеме пример. Нека низата е a = {2, –1, 7, 9, 3}.

i	a_i	a_i > max	max	index
1	2		2	1
2	–1	(–1 > 2) не		
3	7	(7 > 2) да	7	3
4	9	(9 > 7) да	9	4
5	3	(3 > 9) не		

Значи, најголем е 4-тиот елемент со вредност 9.

Постапката за наоѓање на најмалиот елемент во низата е слична. Ќе ја илустрираме на истиот пример.

i	a_i	a_i < min	min	index
1	2		2	1
2	–1	(–1 < 2) да	–1	2
3	7	(7 < –1) не		
4	9	(9 < –1) не		
5	3	(3 < –1) не		

Значи, најмал е 2-риот елемент со вредност –1.

Во дефиницијата на функција која има аргументи низи, низите се пишуваат само со типот и името, по кое се стават средни загради за да се знае дека е низа, а не променлива.

На пример:

```
void NajgolemElement(double a[], double brojnaelementi, double &pozicija,
double &najgolem)
```

Притоа, низата се третира како референтен аргумент иако не е ставен знакот &. Тоа значи дека сите измени на елементите на низата кои ќе се случат при извршување на функцијата, остануваат и по враќање во главната функција.

При повикување на функција со аргумент низа, се пишува само името на низата и тоа без загради:

```
NajgolemElement(a, n, redenbroj, najgolem);
```

Притоа, големината на низата се пренесува како аргумент по вредност.

(Всушност, постои посебен механизам на **покажувачи**[4] (англ. pointers) кој се користи во случајот на низи. Со тој механизам, името на низата се третира како покажувач, кој ја содржи адресата на првиот елемент на низата во меморијата. (Слично како што се пренесува адресата на аргументот во соодветниот референтен параметар)).

[4] Видете во потточката **3.4 Покажувачи**.

Во некои случаи, кога не сакаме да се променат вредностите на елементите во функцијата во која е пренесена низата, бидејќи се третира како референца, треба да се означи како константна со квалификаторот const.

Функциите за наоѓање на најголемиот и на најмалиот елемент во низа, како и главната функција во која се повикуваат тие функции, се дадени на с***лика .1.5.***

```
#include <iostream>
#include <iomanip>
using namespace std;

void citanjeNiza( int&, double [] );
void pecatenjeNiza( const int, const double [] );
int najgolemElement( const int, const double [] );
int najmalElement( const int, const double [] );

int main() {    //  Najgolem i najmal element vo niza

   double a[ 100 ];
   int n, index;

   citanjeNiza( n, a );
   system( "cls" );
   cout << setw( 14 ) << "" << "Ednodimenzionalna niza \n" << endl;
   pecatenjeNiza( n, a );

   index = najgolemElement( n, a );
   cout << "\nNajgolem e elementot so indeks " << index
       << " i so vrednost " << a[ index ] << endl;

   index = najmalElement( n, a );
   cout << "Najmal e elementot so indeks " << index
       << " i so vrednost " << a[ index ] << endl;

   cout << endl;
   system( "Color 17" );
   system( "pause" );
   return 0;
}

void citanjeNiza( int& dimenzija, double x [] ) {
   cout << "Vnesete go brojot na elementi: ";
   cin >> dimenzija;
   cout << "Vnesete gi elementite na nizata: " << endl;
   for( int i = 0; i < dimenzija; i++ ) {
       cout << "x[" << i << "] = ";
       cin >> x[ i ];
   }
```

Слика .1.5

```
42      }
43
44      void pecatenjeNiza( const int dimenzija, const double x [] ) {
45          cout << setw( 10 ) << left << "Index:" << right;
46          for( int i = 0; i < dimenzija; i++ )
47              cout << setw( 5 ) << i;
48          cout << endl;
49          cout << setw( 10 ) << left << "Vrednost:" << right;
50          for( int i = 0; i < dimenzija; i++ )
51              cout << setw( 5 ) << x[ i ];
52          cout << endl;
53      }
54
55      int najgolemElement( const int brojnaelementi, const double x [] ) {
56          int pozicija = 0;
57          double najgolem = x[ 0 ];
58          for( int i = 1; i < brojnaelementi; i++ )
59              if( x[ i ] > najgolem ) {
60                  najgolem = x[ i ];
61                  pozicija = i;
62              }
63          return pozicija;
64      }
65
66      int najmalElement( const int brojnaelementi, const double x [] ) {
67          int pozicija = 0;
68          double najmal = x[ 0 ];
69          for( int i = 1; i < brojnaelementi; i++ )
70              if( x[ i ] < najmal ) {
71                  najmal = x[ i ];
72                  pozicija = i;
73              }
74          return pozicija;
75      }
```

Слика .1.5 продол ение

Пример за излез од програмата е:

```
              Ednodimenzionalna niza

Index:        0    1    2    3    4    5    6    7    8    9
Vrednost:     3    7   12    0   -3  -23    0    5   25    4

Najgolem e elementot so indeks 8 i so vrednost 25
Najmal e elementot so indeks 5 i so vrednost -23

Press any key to continue . . .
```

Наредба for базирана на опсег

Во **Пример 3.1.15** се пресметува збирот на елементите на низата од n елементи $c[]_n$:

```
zbir = 0;
for(int i = 0; i < n; i++)
	zbir += c[i];
```

Забележуваме дека елементите се собираат од првиот до последниот, како што се наредени по индексот i. Меѓутоа, ќе се добие истиот збир ако ги собереме и по кој било редослед.

Има многу проблеми во кои се извршува некоја операција врз елементите на низа, независно од нивниот редослед. На пример: во задачата за наоѓање на најмалиот (или најголемиот) елемент во низа, не мора првиот да се земе за најмал, туку може кој било. Потоа, тој се споредува со останатите, но не мора по некој редослед. Кога ќе се најде некој помал од дотогаш најмалиот, тој ќе биде најмал.

За решавање вакви проблеми, во C++ се користи т.н. **наредба for базирана на опсег** (англ. range-based for), ***слика .1.12.***

```
for(променлива : низа){
	наредба А;
	наредба Б;
	...
	наредба К;
}
```

Слика .1.12

- *променлива* е променливата од типот на низата,
- *низа* е името на низата.

Во телото на наредбата for базирана на опсег не може да има и наредби за ознаки или операции со индексите.

Ќе наведеме неколку примери.

Примери

Пример 3.1.17

Програмскиот сегмент од **Пример 3.1.15** за збирот на елементите на низата c[], може да се запише на следниов начин:

```
for(int broj: c)
	zbir += broj;
```

Пример 3.1.18

Во **Задача 3.1.2**, наредбата for за збирот на елементите:

```
for(i = 0; i < n; i++) {
	zbir += k * a[i];
	k = -k;
}
```

може да се напише и со наредба for базирана на опсег.

```
for(double broj: a) {
	zbir += k * broj;
	k = -k;
}
```

Пример 3.1.19

За наоѓање на најмалиот или на најголемиот елемент во низа, не е потребно да се споредуваат елементите според индексот, туку треба да се споредат сите елементи. Затоа, може да се користи наредбата for базирана на опсег.

```
double najmal = a[0];
double najgolem = a[0];
for(auto broj: a) {
	if(broj < najmal)
		najmal = broj;
	if(broj > najgolem)
		najgolem = broj;
}
```

Задачи за вежбање

Да се напишат програми за следниве задачи со користење на еднодимензионални низи:

1. Да се најде производот на елементите на целобројната низа $a[]_n$.
2. Да се пресмета аритметичката (A) и хармониската (H) средина на бројната низа $a[]_n$.

$$A = \frac{a_1 + a_2 + \ldots + a_n}{n} \qquad H = \frac{n}{\frac{1}{a_1} + \frac{1}{a_2} + \ldots + \frac{1}{a_n}}$$

3. Од низата броеви $a[]_n$ да се пресмета посебно збирот на парните и збирот на непарните броеви.
4. Да се издвојат во посебни низи елементите на низата $a[]_n$ со парни индекси и со непарни индекси.
5. Да се изврши циклично поместување на елементите на низата од знаци $a[]_n$ за к места надесно или налево.
6. Да се провери дали во бројната низа $a[]_n$ се наоѓа елемент со вредност v.
7. Да се најде колку елементи од бројната низа $a[]_n$ имаат помала вредност од v, а колку поголема.
8. Да се формира нова низа $c[]_n$, чии елементи се збир на соодветните елементи на низите $a[]_n$ и $b[]_n$, т. е. $c_i = a_i + b_i$, за i = 0, 1... n – 1.

9. Да се пресмета производот на елементите на бројните низи $a[]_n$ и $b[]_n$, т. е.
   $a_0b_0 + a_1b_1 + \ldots + a_{n-1}b_{n-1}$.
10. Да се пресмета вредноста на полиномот
    $P_n(x) = a_{n-1}x^{n-1} + a_{n-2}x^{n-2} + \ldots + a_1 + a_0$
    во кој коефициентите и аргументот се реални броеви, а n е природен број.

Прашања за проверка на знаењето

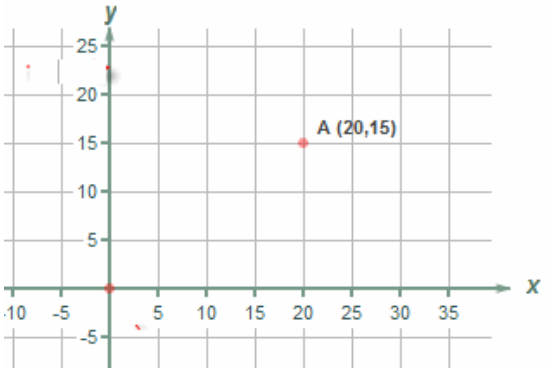
1. Како се декларира еднодимензионална низа?
2. Од каков тип на податок треба да биде индексот на елементите на еднодимензионална низа?
3. Ако е декларирана еднодимензионална низа
```
int a[ 10 ];
```
   кои вредности може да ги прими индексот на елементите на низата?
4. На кои начини може да се врши доделување вредности на елементите на еднодимензионална низа?
5. На кои начини може да се врши иницијализирање на елементите на еднодимензионална низа?
6. Како се дефинира симболичка константа?
7. Како се одредува димензијата на еднодимензионална низа?
8. Кога се користи наредбата for базирана на опсег? Напишете ја нејзината синтакса.

3.2 Дводимензионални низи – матрици

Во потточката **3.1 Еднодимензионални низи** рековме дека податоците кои имаат една димензија може (математички) да се претстават со низи со еден индекс: a_1, a_2, ... , a_n, или скратено $a[]_n$. Овие низи во програмските јазици, па и во C++, се запишуваат со индекси во средни загради: a[0], a[1]... a[n – 1]. Притоа, индексите почнуваат од 0.

Има примери во математиката и во други области, каде што за запишување на некои податоци мора да се користат 2, 3... и повеќе индекси.

На пример, знаеме дека секоја точка во рамнина е одредена со две димензии – координати: x-координата и y-координата, т. е. A(x, y), ***слика .2.1***.



Слика .2.1

Слично, оценките на учениците од еден клас по сите предмети

(да претпоставиме 12) може да се прикажат со табела со две димензии, со редици (хоризонтално, прва димензија) и колони (вертикално – втора димензија).

		1	2	3	4	5	6	7	8	9	10	11	12	
	Предмет →	p_1	p_2	p_3	p_4	p_5	p_6	p_7	p_8	p_9	p_{10}	p_{11}	p_{12}	Prosek
	Ученик ↓													
1	u_1	3	4	5	2	3	4	3	4	5	3	3	2	
2	u_2	4	5	4	5	3	4	5	4	5	5	5	4	
3	u_3	5	5	5	5	2	4	4	4	4	2	4	5	
…	. . .													
	Prosek													

Слика .2.2

Овие податоци може да се запишат како **дводимензионална низа** (англ. two-dimensional array), при што едната димензија е ученици (u), а другата димензија е предмети (p). Значи, секоја оценка во табелата има два индекси. Ако низата ја именуваме со o (скратено од ocenki), тогаш вредностите на елементите на дводимензионалната низа се оценките на учениците по предметите. На пример, $o(u_2, p_7) = 5$, $o(u_3, p_{10}) = 2$ итн. Или, пократко, само со индексите, $o_{2,7} = 5, o_{3,10} = 2$ итн.

Ако знаеме дека има m ученици и n предмети, тогаш низата (математички) се запишува со: $o_{1,1}, o_{1,2},..., o_{1,n}, o_{2,1}, o_{2,2},..., o_{2,n}, o_{3,1},..., o_{m,1}, o_{m,2},..., o_{m,n}$.

Или, попрегледно како дводимензионална табела:

$$\begin{bmatrix} o_{1,1} & o_{1,2} & \cdots & o_{1,j} & \cdots & o_{1,n} \\ o_{2,1} & o_{2,2} & \cdots & o_{2,j} & \cdots & o_{2,n} \\ \cdots & \cdots & \cdots & \cdots & \cdots & \cdots \\ o_{i,1} & o_{i,1} & \cdots & o_{i,j} & \cdots & o_{i,n} \\ \cdots & \cdots & \cdots & \cdots & \cdots & \cdots \\ o_{m,1} & o_{m,2} & \cdots & o_{m,j} & \cdots & o_{m,n} \end{bmatrix}$$

Дводимензионална низа се означува со $o[_{i,j}]_{m,n}$, што значи дека индексот i има вредности од 1 до m, а индексот j има вредности од 1 до n. Пократко, низата се означува со $o[]_{m,n}$, при што m и n покажуваат дека низата е дводимензионална и се знае дека првиот индекс е од 1 до m, а вториот од 1 до n.

Во литературата се користи и записот o[1..m][1..n].

Општиот елемент се означува со $o_{i,j}$.

(Дводимензионалните низи во математиката се нарекуваат **матрици**).

Ваквите дводимензионални низи во програмские јазици, па и во C++, се запишуваат со o[m][n], а општиот елемент е со два индекси во средни загради, o[i][j]. На пример, o[2][7] = 5, o[3][10] = 2 итн. (Погрешно е претставувањето o[2,7]).

Декларација, иницијализација и доделување вредности на елементите на дводимензионални низи

Декларацијата на дводимензионална низа е слична со декларацијата на еднодимензионална низа

тип име [*број1*] [*број2*];

тип – е типот на елементите на низата (short, int, long, float, double, char, boolean, string итн.).
име – е името на низата,
број1 и *број2* – се димензиите на низата и тие мора да бидат литерали или именувани константи поголеми од 0.

За да биде веднаш појасно, во горната табела со оценки за ученици, *број1* ќе биде бројот на ученици во класот (бројот на редици), а *број2* ќе биде бројот на предмети (бројот на колони).

Двата индекси на дводимензионална низа започнуваат од 0.
Првите индекси на дводимензионална низа се: 0, 1, 2... *број1* – 1.
Вторите индекси на дводимензионална низа се: 0, 1, 2... *број2* – 1.

Примери

Пример 3.2.1

Со декларацијата

```
int a[3][4];
```

се декларира низата $a[]_{3,4}$ од 12 елементи, на кои првиот индекс им е 0, 1 или 2, а вториот индекс им е 0, 1, 2 или 3. Оваа низа графички може да се прикаже со табела со 3 редици и 4 колони:

редици (прв индекс) \ колони (втор индекс)	0	1	2	3
0	$a_{0,0}$	$a_{0,1}$	$a_{0,2}$	$a_{0,3}$
1	$a_{1,0}$	$a_{1,1}$	$a_{1,2}$	$a_{1,3}$
2	$a_{2,0}$	$a_{2,1}$	$a_{2,2}$	$a_{2,3}$

Елементите на низата се запишуваат со индексите во средни загради.

	0	1	2	3
0	a[0][0]	a[0][1]	a[0][2]	a[0][3]
1	a[1][0]	a[1][1]	a[1][2]	a[1][3]
2	a[2][0]	a[2][1]	a[2][2]	a[2][3]

Вредности на елементите на дводимензионална низа се доделуваат со наредбата за доделување.

Пример 3.2.2

Со следниве наредби:

```
float c[10][15];
c[0][4] = -5.23f; c[9][2] = 7.543f;
```

се декларира низата c[][] од 10 x 15, т. е. 150 елементи и им се доделуваат вредности на елементите c[0][4] и c[9][2]. (Но не се познати вредностите на другите елементи).

Дводимензионална низа може да се иницијализира при декларирање и со *иницијализирачка листа*.

Пример 3.2.3

Со наредбата

```
int d[3][2] = {};
```

се иницијализираат сите елементи на низата d на вредност 0.

Со наредбата

```
char bukvi[3][2] = {'a', 'b', 'c'};
```

се декларира и иницијализира дводимензионалната низа bukvi.

	0	1
0	a	b
1	c	
2		

Ако нема доволно вредности во иницијализирачката листа за иницијализирање на сите елементи на низата, остатокот се иницијализира со празно место.

Со наредбата

```
int d[3][2] = {-3,5,4};
```

се иницијализираат три елементи со зададените вредности, а останатите со 0.

	0	1
0	–3	5
1	4	0
2	0	0

Пример 3.2.4

Со наредбата

```
int d[3][2] = {{-3, 5},{4, 1},{7, -2}};
```

се декларира и иницијализира низа од 3 редици (секоја редица се става во големи загради) и 2 колони (секоја редица има по 2 елементи), според податоците од иницијализирачката листа. Истата декларација може да се запише и попрегледно:

```
int d[3][2] = {
                    {-3, 5},
                    {4, 1},
                    {7, -2}
              };
```

	0	1
0	–3	5
1	4	1
2	7	–2

Исто така, ако нема доволно елементи за иницијализирање на некоја редица, останатите елементи во неа се иницијализираат со 0.

```
int d[3][2] = {{-3}, {4, 1}, {7}};
```

	0	1
0	–3	0
1	4	1
2	7	0

Истите декларации и иницијализации може да се направат и со следниве наредби:

```
int d[][2] = {{-3, 5},{4, 1},{7, -2}};
```

и

```
int d[][2] = {{-3}, {4, 1}, {7}};
```

при што големината на првата димензија не мора да се наведе. Преведувачот сам ја пресметува.

Меѓутоа, не може да се декларира со наредбата

```
int d[][] = {{-3, 5}, {4, 1}, {7, -2}};
```

бидејќи преведувачот не знае по колку елементи да земе од секоја редица (внатрешната заграда). Може да земе по 1 или по 2. Затоа, може да се изостави само големината на првата димензија.

Пример 3.2.5

Димензиите на низата може да се зададат и преку константи.

Со наредбите:

```
const int m = 10;
const int n = 5;
float b[m][n];
```

се декларира дводимензионална низа со 10 редици и 5 колони, т. е. првиот елемент е b[0][0], а последниот е b[9][4].

Но не мора да се користи целата низа b[m][n] елементи, туку може само дел од неа. На пример, можеме да зададеме помали димензии на низата:

```
int redici = 4;
int koloni = 3;
```

и да се користи само делот b[redici][koloni].

Пресметување димензии на дводимензионална низа

Ако низата a[]$_{m,n}$ е иницијализирана со иницијализирачка листа, тогаш за одредување на должината (бројот на редици), во C++ се користи функцијата sizeof().

```
bajtiElement = sizeof(a[0][0]);     – број бајти на еден елемент.
bajtiRedica = sizeof(a[0]);         – број бајти на една редица.
bajtiNiza = sizeof(a)               – број бајти на цела низа.
```

Потоа може да се пресмета:

```
dolzina = bajtiNiza / bajtiElement;
koloni = bajtiRedica / bajtiElement;
redici = dolzina / koloni;
```

Пример 3.2.6

Програмскиот сегмент за читање елементи на дводимензионалната низа a[]$_{m,n}$ е:

```
for(int i = 0; i < m; i++)
	for(int j = 0; j < n; j++)
		cin >> a[i][j];
```

Пример 3.2.7

Програмскиот сегмент за печатење на дводимензионалната низа b[]$_{m,n}$ е:

```
for(int i = 0; i < m; i++) {
	for(int j = 0; j < n; j++)
		cout << "\tb[" << i << "," << j << "] = " << b[i][j];
	cout << endl;
}
```

Пример 3.2.8

Програмскиот сегмент за наоѓање на збирот на елементите на целобројната дводимензионална низа c[]$_{m,n}$ е:

```
int zbir = 0;
for(int i = 0; i < m; i++) {
	for(int j = 0; j < n; j++)
		zbir += c[i][j];
}
```

Пример 3.2.9

Со програмата од ***слика .2.*** се илустрирани декларација, иницијализација, читање и печатење на дводимензионална низа.

```cpp
#include <iostream>
#include <iomanip>
using namespace std;

const int m = 2;            // Dimenzii na prvata niza
const int n = 3;

int main() {    // Deklaracija, inicijalizacija, citanje i
                // pecatenje na dvodimenzionalna niza

    int i, j;   // Brojaci na redici i na koloni, soodvetno
                // Inicijalizacija na niza so inicijaliziracka lista
    int nizaA[ m ][ n ] = { { 1, 2, 3 }, { 4, 5, 6 } };
    cout << "Prvata niza e inicijalizirana" << endl;

    cout << "\nPecatenje na prvata niza" << endl;
    for( i = 0; i < m; i++ ) {
        for( j = 0; j < n; j++ )
            cout << setw( 5 ) << nizaA[ i ][ j ];
        cout << endl;       //Nov red po sekoja redica
    }

    const int m = 3;        // Dimenzii na vtorata niza
    const int n = 4;
    int nizaB[ m ][ n ] = {};   // Deklaracija i inicijalizacija na 0
                                // Citanje elementi na niza
    cout << "\nVnesete gi elementite na vtorata niza:" << endl;
    for( i = 0; i < m; i++ ) {
        cout << "\nVnesete elementi vo " << i + 1 << "-ta redica: " << endl;
        for( j = 0; j < n; j++ ) {
            cout << "Vnesete go " << j + 1 << "-ot element ";
            cin >> nizaB[ i ][ j ];
        }
    }

    cout << "\nPecatenje na vtorata niza" << endl;
    for( i = 0; i < m; i++ ) {
        for( j = 0; j < n; j++ )
            cout << setw( 5 ) << nizaB[ i ][ j ];
        cout << endl;       //Nov red po sekoja redica
    }

    cout << endl;
    system( "Color 17" );
    system( "pause" );
    return 0;
}
```

Слика .2.

Пример 3.2.10

Со овој пример се илустрирани декларација, иницијализација и доделување вредности на елементите на дводимензионална низа од различен тип, како и пресметување на бројот на редици и на колони.

```
#include <iostream>
#include <string>
using namespace std;

int main()
{   // Deklaracija, inicijalizacija i dodeluvanje vrednosti
    // na elementite na dvodimenzionalna niza

    int i, j, dolzina, redici, koloni, bajtiElement, bajtiRedica, bajtiNiza;
    int a[2][3] = {};
    cout << "N                vi inicijalizirana na 0 "
        << "\n                2 vrednosti:" << endl;
    a[1][2] =                 45;
    redici = 2;
    koloni = 3;
    for (i = 0; i < redici; i++)
        for (j = 0; j < koloni; j++)
            cout << "\ncel[" << i << "][" << j << "] = " << a[i][j];

    float b[][2] = { { -3.5f, 2.7f },{ 7.3f },{ 0.3f, -2.f } };
    b[0][1] = 133.f; b[2][1] = -57.f;
    bajtiElement = sizeof(b[0][0]);
    bajtiRedica = sizeof(b[0]);
    bajtiNiza = sizeof(b);
    dolzina = bajtiNiza / bajtiElement;
    koloni = bajtiRedica / bajtiElement;
    redici = dolzina / koloni;
    cout << "\n\nNiza od realni broevi inicijalizirana "
        << "\ni potoa dodeleni 2 vrednosti:" << endl;
    for (i = 0; i < redici; i++)
        for (j = 0; j < koloni; j++)
            cout << "\nrealen[" << i << "][" << j << "] = " << b[i][j];

    char znaci[][3] = { { '@','#','$' },{ '%','^' },{ '*' },{ '-','+','=' } };
    koloni = 3;
    redici = sizeof(znaci) / sizeof(znaci[0][0]) / koloni;
    cout << "\n\nNiza od znaci:" << endl;
    for (i = 0; i < redici; i++)
        for (j = 0; j < koloni; j++)
            cout << "\nznak[" << i << "][" << j << "] = " << znaci[i][j];

    string iminja[][1] = { { "Antonia" },{ "Atanasia" },{ "Mihaela" },{ "Jovana" },
    {},{ "Jana" } };
    koloni = 1;
    redici = sizeof(iminja) / sizeof(iminja[0][0]) / koloni;
    cout << "\n\nNiza od stringovi:" << endl;
    for (i = 0; i < redici; i++)
        for (j = 0; j < koloni; j++)
            cout << "\nstring[" << i << "][" << j << "] = " << iminja[i][j];
```

(int)3
Search Online

Слика .2.4

```
50
51      cout << endl;
52      cout << endl;
53      system("Color 17");
54      system("pause");
55      return 0;
56  }
```

Слика .2.4 продол ение

Резултатот од извршување на програмата е:

```
Niza od celi broevi inicijalizirana na 0
i potoa dodeleni 2 vrednosti:

cel[0][0] = 0
cel[0][1] = 0
cel[0][2] = 0
cel[1][0] = -45
cel[1][1] = 0
cel[1][2] = 123

Niza od realni broevi inicijalizirana
i potoa dodeleni 2 vrednosti:

realen[0][0] = -3.5
realen[0][1] = 133
realen[1][0] = 7.3
realen[1][1] = 0
realen[2][0] = 0.3
realen[2][1] = -57

Niza od znaci:

znak[0][0] = @
znak[0][1] = #
znak[0][2] = $
znak[1][0] = %
znak[1][1] = ^
znak[1][2] =
znak[2][0] = *
znak[2][1] =
znak[2][2] =
znak[3][0] = -
znak[3][1] = +
znak[3][2] = =

Niza od stringovi:

string[0][0] = Antonia
string[1][0] = Atanasia
string[2][0] = Mihaela
string[3][0] = Jovana
string[4][0] = Jana
string[5][0] = Teo

Press any key to continue . . .
```

Решени задачи

Задача 3.2.1
Да се отпечати таблицата за множење до n.

Програмата е дадена на с*лика .2.5*.

```
#include <iostream>
#include <iomanip>
using namespace std;

int main() {    // Tablica mnozenje

    const int m = 20;
    const int n = 20;
    int t[ m ][ n ];
    int i, j, k;
    cout << "Vnesete do koj broj sakate da se ispecati tablicata: ";
    cin >> k;
    for( i = 1; i <= k; i++ )
        for( j = 1; j <= k; j++ )
            t[ i ][ j ] = i * j;
    system( "cls" );
    cout << setw( 20 ) << "" << "TABLICA MNOZENJE \n" << endl;
    cout << setw( 3 ) << "";
    for( j = 1; j <= k; j++ )
        cout << setw( 5 ) << j;
    cout << endl << endl;
    for( i = 1; i <= k; i++ ) {
        cout << setw( 3 ) << i;
        for( j = 1; j <= k; j++ )
            cout << setw( 5 ) << t[ i ][ j ];
        cout << endl;
    }

    cout << endl;
    system( "Color 17" );
    system( "pause" );
    return 0;
}
```

Слика .2.5

Еден излез од програмата е:

```
                    TABLICA MNOZENJE

      1    2    3    4    5    6    7    8    9   10   11   12   13   14   15

  1   1    2    3    4    5    6    7    8    9   10   11   12   13   14   15
  2   2    4    6    8   10   12   14   16   18   20   22   24   26   28   30
  3   3    6    9   12   15   18   21   24   27   30   33   36   39   42   45
  4   4    8   12   16   20   24   28   32   36   40   44   48   52   56   60
  5   5   10   15   20   25   30   35   40   45   50   55   60   65   70   75
  6   6   12   18   24   30   36   42   48   54   60   66   72   78   84   90
  7   7   14   21   28   35   42   49   56   63   70   77   84   91   98  105
  8   8   16   24   32   40   48   56   64   72   80   88   96  104  112  120
  9   9   18   27   36   45   54   63   72   81   90   99  108  117  126  135
 10  10   20   30   40   50   60   70   80   90  100  110  120  130  140  150
 11  11   22   33   44   55   66   77   88   99  110  121  132  143  154  165
 12  12   24   36   48   60   72   84   96  108  120  132  144  156  168  180
 13  13   26   39   52   65   78   91  104  117  130  143  156  169  182  195
 14  14   28   42   56   70   84   98  112  126  140  154  168  182  196  210
 15  15   30   45   60   75   90  105  120  135  150  165  180  195  210  225

Press any key to continue . . .
```

Задача 3.2.2

Да се најдат најмалиот и најголемиот елемент во дводимензионална низа. За читање, за печатење и за наоѓање на најмалиот и на најголемиот елемент да се напишат посебни функции.

Објаснува е: Во апликацијата се користат функциите citanje2DNiza(), pecatenje2DNIza(), max2DElement() и min2DElement(). При барање на најмалиот и на најголемиот елемент, излезни вредности од функциите се индексите на елементот.

```
1   #include <iostream>
2   #include <iomanip>
3   using namespace std;
4
5   void citanje2DNiza( int&, int&, double [][ 10 ] );
6   void pecatenje2DNiza( const int, const int, const double [][ 10 ] );
7   void max2DElement( const int, const int, const double [][ 10 ], int&, int& );
8   void min2DElement( const int, const int, const double [][ 10 ], int&, int& );
9
10  int main() {    // Najmal i najgolem element vo dvodimenzionalna niza
11      const int m = 10;
12      const int n = 10;
13      double a[ m ][ n ] = {};
14      int i, j, redici, koloni, imax, jmax, imin, jmin;
15      double min, max;
16
17      citanje2DNiza( redici, koloni, a );
18
19      system( "cls" );
20      cout << "Dvodimenzionalna niza" << endl;
21      pecatenje2DNiza( redici, koloni, a );
22
```

Слика .2.

```
23      // Baranje na najmaliot i na najgolemiot element
24      min2DElement( redici, koloni, a, imin, jmin );
25      max2DElement( redici, koloni, a, imax, jmax );
26      cout << "\nNajgolemiot element vo nizata e a["
27          << imax << "][" << jmax << "] = " << a[ imax ][ jmax ] << endl;
28      cout << "\nNajmaliot element vo nizata e a["
29          << imin << "][" << jmin << "] = " << a[ imin ][ jmin ] << endl;
30
31      cout << endl << endl;
32      system( "Color 17" );
33      system( "pause" );
34      return 0;
35  }
36
37  void citanje2DNiza( int& red, int& kol, double a [][ 10 ] ) {
38      cout << "Vnesete go brojot na redici (<=10): "; cin >> red;
39      cout << "Vnesete go brojot na koloni (<=10): "; cin >> kol;
40      cout << "Vnesete gi elementite na nizata po redici: " << endl;
41      for( int i = 0; i < red; i++ ) {
42          for( int j = 0; j < kol; j++ ) {
43              cout << "a[" << i << "][" << j << "] = ";
44              cin >> a[ i ][ j ];
45          }
46      }
47  }
48
49  void pecatenje2DNiza( const int red, const int kol, const double a [][ 10 ] ) {
50      cout << setw( 2 ) << "";
51      for( int j = 0; j < kol; j++ )
52          cout << setw( 5 ) << j;
53      cout << endl << endl;
54      for( int i = 0; i < red; i++ ) {
55          cout << setw( 2 ) << i;
56          for( int j = 0; j < kol; j++ )
57              cout << setw( 5 ) << a[ i ][ j ];
58          cout << endl;
59      }
60  }
61
62  void min2DElement( const int red, const int kol, const double b [][ 10 ],
63      int& iNajmal, int& jNajmal ) {
64      double najmal = b[ 0 ][ 0 ];
65      iNajmal = 0; jNajmal = 0;
66      for( int i = 0; i < red; i++ )
67          for( int j = 0; j < kol; j++ )
68              if( b[ i ][ j ] < najmal ) {
69                  najmal = b[ i ][ j ];
70                  iNajmal = i; jNajmal = j;
```

Слика .2. продол ение 1

```
71              }
72      }
73
74      void max2DElement( const int red, const int kol, const double b [][ 10 ],
75          int& iNajgolem, int& jNajgolem ) {
76          double najgolem = b[ 0 ][ 0 ];
77          iNajgolem = 0; jNajgolem = 0;
78          for( int i = 0; i < red; i++ )
79              for( int j = 0; j < kol; j++ )
80                  if( b[ i ][ j ] > najgolem ) {
81                      najgolem = b[ i ][ j ];
82                      iNajgolem = i; jNajgolem = j;
83                  }
84      }
```

Слика .2. продол ение 2

Излезот од извршување на програмата е:

```
Dvodimenzionalna niza
      0    1    2    3

 0    3   -2   -7    9
 1    4   -6    5   34
 2   23  -12  -56  -33
 3    0    3    4   -2
 4   -5   -7    0    4

Najgolemiot element vo nizata e a[1][3] = 34

Najmaliot element vo nizata e a[2][2] = -56


Press any key to continue . . .
```

Задачи за вежбање

1. Да се најде аритметичката средина на елементите на дводимензионалната низа $a[]_{m,n}$.
2. Да се пресмета збирот на двете дводимензионални низи со исти димензии, $c[]_{m,n} = a[]_{m,n} + b[]_{m,n}$. (Збирот на i,j-тиот елемент е $c_{i,j} = a_{i,j} + b_{i,j}$).
3. Да се формира дводимензионалната низа $a[][]_{n,n}$ што ќе ги содржи броевите од 1 до n^2 во облик на змија. На пример, за n = 3 низата го има следниот изглед:

```
1 2 3
6 5 4
7 8 9
```

4. Да се најдат максималните елементи по редици и минималните елементи по колони во дводимензионалната низа $a[]_{m,n}$.
5. Да се најде посебно збирот на елементите на главната дијагонала и збирот на елементите на споредната дијагонала на дводимензионалната низа $q[][]_{n,n}$.

6. Да се најде збирот на елементите над главната дијагонала на дводимензионалната низа a[]$_{n,n}$.
7. Да се избришат i-тата редица и j-тата колона од дводимензионалната низа a[]$_{m,n}$.
8. Да се пресмета збирот на елементите кои лежат на дијагоналите кои минуваат низ елементот $a_{i,j}$ на дводимензионалната низа a[]$_{m,n}$.
9. Да се заротираат колоните на дводимензионалната низа a[]$_{m,n}$ за к места налево (или надесно).
10. Да се генерира следнава дводимензионална низа a[]$_{m,n}$.

1	4	9	16	25	…
2	3	8	15	24	…
5	6	7	14	23	…
10	11	12	13	22	…
17	18	19	20	21	…
26	27	…	…	…	…

Прашања за проверка на знаењето

1. Како се декларира и иницијализира дводимензионална низа?
2. Како можеме да ја декларираме дводимензионалната низа b[][] со 4 редици и 3 колони?
3. Дали во дефинираната низа b[][] во претходната задача постојат елементите: b[3][4], b[4][3], b[0][4], b[0][3], b[3][0] и b[4][0]?
4. Нека е дадено
```
int a[5][5] = {1, 2, 3, 4, 5, 6, 7, 8};
```
Која е вредноста на елементот a[1][3]?
5. Ако е декларирано
```
double d[10][10];
```
каде е грешката во наредбава
```
d[5][10] = 4.55;
```
6. Што ќе отпечати следниов програмски сегмент:
```
int i, j, niza[5][5];
for(i = 0, j = 0; i < 5; i++, j++)
   niza[i][j] = i + j;
for(i = 0, j = 0; i < 5; i++, j++)
   cout << niza[i][j] << endl;
```
7. Да се напише програмски сегмент за читање на елементите на дводимензионалната низа a[]$_{m,n}$.
8. Да се напише програмски сегмент за печатење на дводимензионалната низа a[]$_{m,n}$.

3.3 ПОКАЖУВАЧИ

Големината на меморијата која им се доделува на променливите при нивното сместување во меморијата зависи од нивниот тип. (Тоа го видовме кога зборувавме за типот на податоци). На пример, ако се декларирани променливите:

```
int celBroj;
float realenBroj;
```

тогаш при сместување на променливата celBroj во меморијата, ѝ се доделува меморија во која ќе може да се запише најголемиот цел број од дефинираниот опсег на целите броеви од типот int. Исто така, при сместување на променливата realenBroj во меморијата, нејзе ѝ се доделува толку меморија за да може во неа да се запише најголемиот реален број од дефинираниот опсег на реални броеви од типот float.

Променливите може да се сместуваат во меморијата на два начини:

- Статички.
- Динамички.

При извршување на програмата, **статичките променливи** (англ. static variables) се сместуваат на фиксна адреса во меморијата. Тие остануваат на таа адреса до завршување на програмата иако нивната вредност може да се менува. Бидејќи адресата им е фиксна и секогаш позната, пристапот до нив при читање или при запишување нова вредност е многу брз.

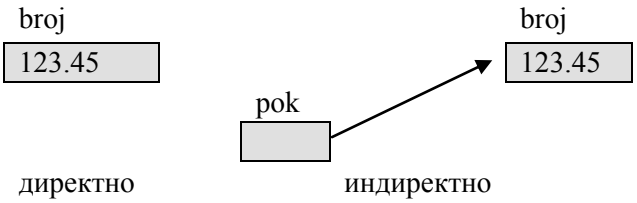
Динамичките променливи (англ. dynamic variables), пак, се сместуваат во меморијата во моментот на нивното креирање (по декларирањето). Притоа, адресата на сместување се одредува во тој момент.

Во современите програмски јазици, за пристап до динамичките променливи кои се сместуваат на која било слободна адреса во моментот на креирање, се користи механизам на **покажувачи** (англ. pointers).

Декларација на покажувачи

Покажувачите се променливи чии вредности се мемориски адреси. Вакви променливи се наречени **покажувачки променливи** (англ. pointer variables) или променливи од типот покажувач. Познато е дека променливите содржат податоци од соодветниот тип: цели броеви, реални броеви, знаци, стрингови итн. Покажувачките променливи не може да содржат други податоци, освен адреси на променливи. Во таа смисла, можеме да кажеме дека променливите од кој било тип, освен од типот покажувач, директно референцираат вредност (број, знак, стринг итн.), а променливите од типот покажувач индиректно референцираат вредност.

На пример, ако broj е променлива, а pok покажувач на променливата broj, тогаш директниот и индиректниот пристап до променливата broj е претставен на следнава слика.



Покажувачите се декларираат на следниов начин:

```
тип *име_на_покажувач;
```

тип – е типот на променливи на кои ќе покажува покажувачот,
* – е знак за декларација на покажувачот, т. е. на покажувачката променлива.

На пример, со

```
int *pok;
```

се декларира покажувачот pok, кој може да се користи само за да покажува на променливи од типот int.

Со декларацијата

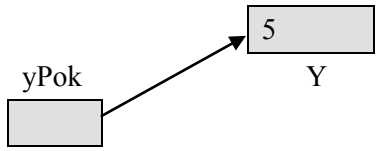
```
float *p;
```

се декларира покажувачот p, кој може да се користи само за да покажува на променливи од типот float.

Адресен оператор &

Операторот & се нарекува **адресен оператор** (англ. address operator) Тој е унарен оператор, кој ја враќа адресата на операндот наведен по него. На пример, ако ги имаме следните декларации и иницијализации:

```
int y = 5;
int *yPok;
```

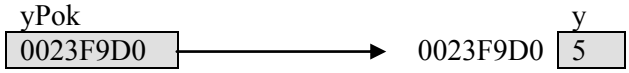


со наредбата

```
yPok = &y;
```

на покажувачот yPok му се доделува адресата на променливата y. Затоа, за покажувачот yPok се вели дека „покажува на y“.

Доколку променливата y се наоѓа на адреса 0023F9D0, по извршувањето на претходната наредба, се добива следнава ситуација:



Доделувањето адреса на покажувачот може да се направи и при неговото декларирање на следниов начин:

```
int *yPok = &y;
```

Не може да се напише

```
yPok = y;
```

бидејќи на покажувач не може да му се додели друга вредност, освен адреса.

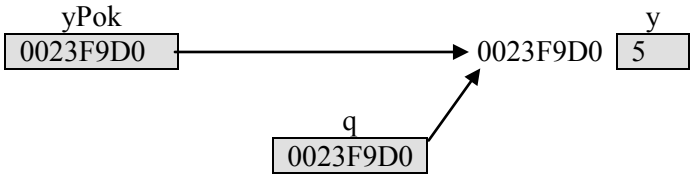
Ако со

```
int *q;
```

декларираме покажувач q кој може да се користи за покажување на целобројни променливи, тогаш со наредбата

```
q = yPok;
```

се насочува и овој покажувач да покажува на променливата y.



Ако декларираме покажувач на реални променливи

```
float *r;
```

не може да се напише

```
r = yPok;
```

или

```
r = &y;
```

бидејќи покажувачот r не може да се користи за покажување на целобројни променливи, туку само на реални променливи.

Операнд на адресниот оператор & мора да е променлива.

Оператор за дереференцирање*

Променливата y може да се означи и преку покажувачот yPok со *yPok.

Операторот * се нарекува **индиректен оператор** (англ. indirection operator) или **оператор за дереференцирање** (англ. dereferencing operator). Тој ја враќа вредноста која ја има променливата на која покажува неговиот операнд. Во случајов, *yPok ја враќа вредноста на променливата на која покажува покажувачот yPok.

На пример, со наредбата

```
cout << *yPok;
```

ќе се отпечати вредноста на променливата y, т. е. 5. Истата вредност ќе се отпечати и со наредбата

```
cout << y;
```

Ако декларираме

```
int *p = &x;
```

тогаш со наредбата

```
*p = *yPok;
```

вредноста на променливата y (= 5) ќе ѝ се додели на променливата x.

Ако следна наредба е

```
*p = *p + 3;
```

тогаш вредноста на променливата y ќе се зголеми за 3 и ќе биде 8.

Да наведеме уште еден пример. Нека ја имаме следнава секвенца од наредби:

```
int* ip;      // ip e pokazuvac kon int promenliva
int b = 47;
int *ipb = &b;
```

На овој начин b и ipb се иницијализирани, а покажувачот кон целиот број ipb ја содржи адресата на b. За да се пристапи кон променливата b преку покажувачот, тој се дереференцира на следниов начин:

```
*ipb = 100;
```

Сега b содржи вредност 100 наместо 47.

Иницијализација на покажувачи

Покажувачите треба да се иницијализираат или при декларација или во некоја наредба за доделување. Покажувачот може да биде иницијализиран на 0, NULL или на адресата на некоја променлива. Покажувач со вредност NULL не покажува на ниту една променлива. Иницијализирањето на покажувачот на 0 е еквивалентно со иницијализирањето на NULL, но меѓу програмерите повеќе се користи NULL. Вредноста 0 е единствената целобројна вредност која може да му се додели на некој покажувач.

Треба да се нагласи дека при декларирање на покажувач без истовремена иницијализација, тој не покажува никаде. Затоа, пред да се употреби, тој мора да се постави да покажува на конкретна адреса.

Следниов пример:

```
int *ip;
*ip=100;
```

ќе јави грешка од типот „Null pointer assignment“, бидејќи претставува обид да се додели вредност 100 на непостојна мемориска локација затоа што ip не покажува на ниту една мемориска локација.

Исправна иницијализација е следнава:

```
int *ip, x;
ip = &x;
*ip = 100;
```

Примери

Пример 3.3.1
Користење на операторите & и *.

Во следнава програма (с*лика . .1*) е илустрирано користењето на операторите * и &. Мемориските локации се печатат со помош на операторот << како хексадецимални броеви.

```
1   /* Koristenje na  & i * operatorite */
2   #include <iostream>
3   using namespace std;
4
5   int main() {
6       int b;        /* b e cel broj */
7       int* bPok;    /* bPok e pokazhuvach kon cel broj */
8
9       b = 7;
10      bPok = &b;    /* bPok pokazhuva na adresata na b */
11
12      cout << "Adresata na b e " << &b
13          << "\nVrednosta na bPok e " << bPok;
14
15      cout << "\n\nVrednosta na b e " << b
16          << "\nVrednosta na *bPok e " << *bPok;
17
18      cout << "\n\nPokazhuvame deka * i & se komplementarni "
19          << "eden na drug\n&*bPok = " << &*bPok
20          << "\n*&bPok = " << *&bPok << endl;
21
22      cout << endl;
23      system( "Color 17" );
24      system( "pause" );
25      return 0;
26  }
```

Слика . .1

Треба да се забележи дека:

а) адресата на променливата a и вредноста на покажувачот aPok се исти,

б) операторите * и & се комплементарни еден на друг.

Еден можен излез од оваа програма е:

```
Adresata na b e 002CF71C
Vrednosta na bPok e 002CF71C

Vrednosta na b e 7
Vrednosta na *bPok e 7

Pokazhuvame deka * i & se komplementarni eden na drug
&*bPok = 002CF71C
*&bPok = 002CF71C

Press any key to continue . . .
```

Пример 3.3.2

Користење на покажувач како аргумент на функција.

Во програмата на ***слика . .2***, функцијата f() има аргумент покажувач. Со наредбата *p = 5; го дереференцира, со што се менува содржината на променливата x.

```
1   // Pokazuvac kako argument na funkcija
2   #include <iostream>
3   using namespace std;
4
5   void f( int* p );
6
7   int main() {
8       int x = 47;
9       cout << "x = " << x << endl;
10      cout << "&x = " << &x << endl;
11      f( &x );
12      cout << "x = " << x << endl;
13
14      cout << endl;
15      system( "Color 17" );
16      system( "pause" );
17      return 0;
18  }
19
20  void f( int* p ) {
21      cout << "p = " << p << endl;
22      cout << "*p = " << *p << endl;
23      *p = 5;
24      cout << "p = " << p << endl;
25  }
```

Слика . .2

Еден можен излез е:

```
x = 47
&x = 0029F940
p = 0029F940
*p = 47
p = 0029F940
x = 5
```

Покажувачи и низи

Постои тесна врска меѓу низите и покажувачите.

На пример, нека е декларирана и иницијализирана низата

```
int z[] = {1, 2, 4, 8, 16};
```

Нејзиниот идентификатор z има вредност на адресата на првиот елемент од низата и е од *тип покажувач на променливи од типот на елементите на низата.* Во овој случај, z е покажувач на целобројни променливи, т. е. од типот int.

Затоа:

```
z;
```

и

```
&z[0];
```

ја содржат адресата на првиот елемент на низата.

Покажувачите често се користат за пристап до елементите на низа. Тоа е можно бидејќи *елементите на низите во меморијата се сместуваат во последователни адреси*.

Нека ја имаме следнава декларација

```
int z[10], *p;
```

Со наредбата

```
p = z;        //imeto na nizata e pokazuvac kon prviot element
              //ova e isto kako i p = &z[0]
```

на покажувачот p му се доделува вредноста на адресата на првиот елемент на низата z[].

Вредноста на првиот елемент од низата може да се добие со:

```
*z;
```

или

```
z[0];
```

Примери

Пример 3.3.3

Излезот од следнава програма е двапати иста хексадецимална адреса.

```
1   #include <iostream>
2   using namespace std;
3
4   int main() {
5       int a[ 10 ];
6       cout << "a = " << a << endl;
7       cout << "&a[0] = " << &a[ 0 ] << endl;
8
9       cout << endl;
10      system( "Color 17" );
11      system( "pause" );
12      return 0;
13
14  }
```

Слика . .

```
a = 0038F938
&a[0] = 0038F938
```

Пример 3.3.4

Со следнава програма (***слика . .4***) се покажува како се доделуваат вредности на низата a[], преку покажувач на првиот елемент во неа.

```
1   #include <iostream>
2   using namespace std;
3
4   int main() {
5       int a[ 10 ];
6       int i;
7       int* ip = a;
8       for( i = 0; i < 5; i++ ) {
9           *( ip + i ) = 10 * i;
10          cout << *( ip + i ) << endl;
11          cout << a[ i ] << endl;
12      }
13
14      cout << endl;
15      system( "Color 17" );
16      system( "pause" );
17      return 0;
18
19  }
```

Слика . .4

Излезот кој се добива е:

```
0
0
10
10
20
20
30
30
40
40

Press any key to continue . . .
```

Адресна аритметика

Над покажувачите може да се извршуваат аритметички операции, и тоа:

- Инкрементирање на покажувач (++).
- Декрементирање на покажувач (– –).
- Додавање целобројна вредност (+ или +=).
- Одземање целобројна вредност (– или –=).
- Додавање (одземање) еден покажувач од друг.

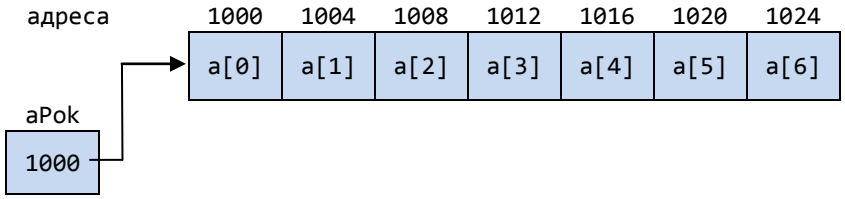
Нека е декларирана целобројната низа a[] од 7 елементи и покажувач aPok кој покажува на првиот елемент на низата:

```
int a[7];
int* aPok = a;
```

Ќе претпоставиме дека првиот елемент на низата a[] се наоѓа на мемориската адреса 1000. Тоа значи дека покажувачот aPok има вредност 1000.

Исто така, ќе претпоставиме дека целобројните променливи зафаќаат по 4 бајти, како што е претставено на следнава слика.

Да ги разгледаме наведените операции со покажувачи.



а Инкрементира е и декрементира е на пока увач

Со наредбата

```
++aPok;
```

покажувачот aPok ќе се инкрементира за 1, т. е. ќе покажува на следниот елемент a[1]. Значи, неговата содржина нема да биде 1000, туку ќе биде 1004.

Во општ случај, ако покажувачот се инкрементира (декрементира) за 1, неговата адреса се зголемува (намалува) за онолку бајти колку што зафаќа променливата на која покажува. Во нашиов случај, бидејќи елементите на низата се целобројни вредности и зафаќаат по 4 бајти, содржината на aPok ќе биде 1004. Ако низата a[] беше от типот double и ако претпоставиме дека променливите од типот double зафаќаат меморија од 8 бајти, тогаш содржината на aPok ќе беше 1008.

б Додава е и одзема е целобројна вредност на/од пока увач

Со наредбата

```
aPok = aPok + 3;
```

или

```
aPok += 3;
```

содржината на покажувачот ќе се зголеми за 3 (1000 + 3 * 4 = 1012) и тој ќе покажува (во нашиот пример) на елементот a[3] .

Ако следна наредбата е

```
aPok = aPok – 2;
```

покажувачот ќе покажува на елементот a[1].

в Додава е на еден пока увач на друг и одзема е на еден пока увач од друг

Ако се декларирани уште два покажувачи

```
int *aP1,*aP2;
```

и поставени да покажуваат на 2-риот и 5-тиот елемент на низата:

```
aP1 = a+2;
aP2 = a+5;
```

тогаш со наредбата

```
aPok = a + aP2 – aP1;
```

покажувачот aPok ќе покажува на елементот a[3].

Покажувачката аритметика нема смисла да се изведува за други типови податоци освен за низи.

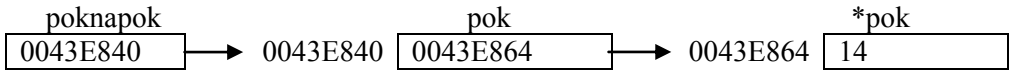
Операторите за еднаквост или релациските оператори може да се користат и за покажувачите, при што се споредуваат адресите запишани во покажувачите. На пример, при испитување дали некој покажувач покажува некаде, треба да се спореди со NULL.

Покажувачот може да покажува и на друг покажувач.

На пример, со

```
int **poknapok = &pok;
```

се декларира и иницијализира покажувачот poknapok да покажува на покажувачот pok.



Забелешка

Видовме дека, ако е декларирано

```
int a[10], *ip;
```

тогаш следниве наредби се исправни:

```
ip = a;
ip++;
```

Меѓутоа, следниве наредби не се исправни:

```
a = ip;
```

и

```
a++;
```

Примери

Пример 3.3.5

Со програмата на ***слика . .5*** се прикажува содржината на покажувачи од целоброен и од реален тип.

Излезот кој се добива може да биде:

```
ip = 1243916
ip = 1243920
dp = 1243828
dp = 1243836

Press any key to continue . . .
```

```
1   #include <iostream>
2   using namespace std;
3
4   int main() {
5       int i[ 10 ];
6       double d[ 10 ];
7       int* ip = i;
8       double* dp = d;
9       // Pokazuvaci na celobrojni vrednosti
10      cout << "ip = " << ( long ) ip << endl;
11      ip++;
12      cout << "ip = " << ( long ) ip << endl;
13      // Pokazuvaci na realni vrednosti
14      cout << "dp = " << ( long ) dp << endl;
15      dp++;
16      cout << "dp = " << ( long ) dp << endl;
17
18      cout << endl;
19      system( "Color 17" );
20      system( "pause" );
21      return 0;
22
23  }
```

Слика . .5

Се забележува дека целобројните променливи зафаќаат (1243920 – 1243916 =) 4 бајти, додека реалните променливи зафаќаат (1243836 – 1243828 =) 8 бајти.

Пример 3.3.6

Со овој пример се покажуваат различните начини на насочување на покажувачите и доделување вредности на променливите на кои покажуваат.

```
1   // manipulacija so pokazhuvachi
2   #include <iostream>
3   using namespace std;
4
5   int main() {
6       int broevi[ 5 ];
7       int* p;
8       // Dodeluvawe vrednosti na nizata so pokazuvaci
9       p = broevi;             *p = 10;            // p[ 0 ]
10      p++;                    *p = 20;            // p[ 1 ]
11      p = &broevi[ 2 ];       *p = 30;            // p[ 2 ]
12      p = broevi + 3;         *p = 40;            // p[ 3 ]
13      p = broevi;             *( p + 4 ) = 50;    // p[ 4 ]
```

Слика . .6

```
14      for( int n = 0; n < 5; n++ )
15          cout << broevi[ n ] << ", ";
16      cout << endl;
17      system( "Color 17" );
18      system( "pause" );
19      return 0;
20  }
```

Слика . .6 продол ение

Излезот кој се добива е:

```
10, 20, 30, 40, 50,
```

Наредбите new и delete

Наредбата **new** има форма

покажувач = new *тип;*

Со неа во меморијата се креира променлива од соодветен *тип* и на *покажувач* му се доделува адресата на креираната променлива. (Во спротивно, покажувачот добива вредност NULL).

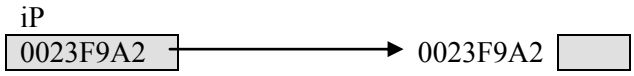
На пример, ако е декларирана покажувачот

```
int *iP;
```

тогаш со наредбата

```
iP = new int;
```

во меморијата се креира неименувана променлива и нејзината адреса му се доделува на покажувачот iP.



Вредноста на неименуваната променлива може да ѝ се додели при креирањето со

```
int *iP;
ip = new int(25);
```

или истовремено

```
int *iP = new int(25);
```

Со декларацијата

```
int *aPok = new a[4];
```

се резервира меморија за низа од 4 елементи од типот int, а на покажувачот aPok му се доделува адресата на првиот елемент од низата.

Наредба **delete** се користи за бришење (ослободување) на мемориската локација.

На пример, со

```
delete iP;
```

се ослободува (брише) меморијата на променливата на која покажувал покажувачот iP, при што тој останува недефиниран.

При бришење на низи, се користи синтаксата

```
delete [] покажувач;
```

за да се нагласи дека покажувачот покажувал на низа од елементи.

На пример,

```
delete [] aPok;
```

Задачи за вежбање

1. Напишете програма во која ќе се дефинираат три покажувачи од типот int, float и char и три променливи a, b и c од соодветните типови. Променливите да се иницијализираат, а потоа да се постават покажувачите да покажуваат на соодветните променливи. Да се отпечатат вредностите и адресите на променливите a, b и c, користејќи ги покажувачите.
2. Напишете програма која ќе генерира грешка „Null pointer assignment“, а потоа покажете како може да се поправи грешката.
3. Креирајте програма во која се внесува низата $a[]_n$ од целобројни елементи, а потоа отпечатете ја вредноста на секој елемент заедно со неговата адреса.
4. Креирајте програма во која се внесува низата $a[]_n$ од double елементи, а потоа отпечатете ја вредноста на секој елемент заедно со неговата адреса. Која разлика ја забележувате во излезот, во однос на претходната задача?
5. Напишете програма во која се дефинираат знаковната низа $x[]_n$ и знакот c. Програмата треба да одговори на прашањето дали знакот c се содржи во низата $x[]_n$. Притоа, за пристап до елементите на низата да се користи покажувач.

Прашања за проверка на знаењето

1. Какви начини постојат за доделување меморија на променливите? Објаснете ги.
2. Како се врши декларација на покажувач?
3. Како се нарекува операторот * и што претставува тој?
4. Како се нарекува операторот & и што претставува тој?
5. Што е содржината на покажувач, а што на покажувачка променлива?
6. Дали кој било покажувач може да покажува на променлива од кој било тип? Образложете го одговорот.
7. Што е погрешно во следниов програмски сегмент:
```
int *pok;
char a;
pok = &a;
```
8. Што е погрешно во следниов програмски сегмент:
```
int *pok;
*pok = 5;
```
9. По низата наредби:
```
float *fp, f=100f;
```

```
fp = &f;
f = 200f;
*f = 300;
```

која ќе биде вредноста на променливата f?

10. Што е погрешно во следнава низа наредби?

```
char *c, ch = 'A';
cout << *c;
```

11. Ако покажувачот pok покажува на променливата a, тогаш што претставува изразот &*pok?
12. Кога има смисла да се користи адресна аритметика кај покажувачи?
13. Како се декларира (и иницијализира) покажувач со наредбата new?
14. Како се декларира покажувач на низа со наредбата new?
15. Како се брише покажувач на променлива, а како на низа, со наредбата delete?

3.4 Стрингови

Во поднасловите **Наредба за печатење** и **Типови податоци** од потточката **1.6 Вовед во C++**, како и во поднасловот **Читање и печатење знаковни и стринг-податоци** од потточката **1.8 Читање и печатење податоци**, се запознавме со поимот **стринг** (англ. string). Исто така, во многу примери користевме стринг променливи или низи од стрингови.

Рековме дека стринговите се податоци составени од знаковни податоци како секвенци од знаци[5]. Таквите податоци се третираат како посебни структури составени од простиот тип char, а се наречени тип string.

Со стринговите може да се вршат разни операции, како: спојување, споредување, одредување должина на стринг и многу други. Најчесто, операциите со стрингови се извршуваат со функции кои се наоѓаат во библиотеката **<string>** на ***стандардната библиотека на C*** . За да ги користиме овие функции во програмите, мора на почетокот од програмата да се вклучи библиотека <string>, со директивата #include.

```
#include <string>
```

Функции за работа со стрингови

Во C++ многу се работи со стрингови. Постојат многу функции во библиотеката <string>. Овде ќе ги објасниме оние кои најчесто се користат.

Во примерите ќе ги користиме следните променливи:

```
string s, s1, s2;
s = "C++ e najdobriot programski jazik";
```

[5] Стринговите се разликуваат од текстуалните низи по тоа што тие немаат на крајот нулти знак.

```
s1 = "Dobar";
s2 = "den";
```
s3 – стринг променлива
rezultat – стринг променлива
potstring – стринг променлива
znak – знаковна променлива
n – целобројна променлива
tocno – логичка променлива

Да се потсетиме дека секој знак во стрингот има позиција од 0 (првиот знак), 1 (вториот знак) итн. Последниот знак има позиција колку што е бројот на знаци (должина на стрингот) минус 1. На пример, стрингот s има должина 33. Затоа, позициите на знаците се 0, 1, 2... 32.

Во досегашниот текст користевме две операции со стрингови, и тоа:

– Доделување еден стринг на друг со операторот =.

Пример: `s3 = s1;`

– Спојување стрингови со операторот +.

Пример: `s3 = s1 + s2;`

За овие две операции со стрингови, постојат посебни функции, кои ќе ги објасниме во продолжение:

- **Доделување еден стринг на друг** **assign()**

`s3.assign(s1)` Вредноста на стрингот s1 се доделува на стрингот s3.

Ефектот е ист со наредбата
```
s3 = s1;
```

Пример:
```
s1 = "Dobar";
```
`s3.assign(s1);` На стринг s3 му се доделува константата `"Dobar"`.

- **Спојување два стринга** **append()**

`s1.append(s2)` Ги спојува стринговите s1 и s2.

Ефектот е ист со наредбата
```
s1 = s1 + s2;
```

Примери
```
s1 = "Dobar";
s2 = "den";
```
`s1.append(s2);` Стрингот s1 ќе добие вредност `"Dobarden"`.

За да ги раздвоиме со празно место:
```
s1.append(" ");
```

```
s1.append(s2);
```
s1 ќе добие вредност "Dobar den".

или со една наредба:

```
(s1.append(" ")).append(s2);
```
s1 ќе добие вредност "Dobar den".

Ефектот е ист со наредбата

```
s1 = s1 + " " + s2;
```

- **Должина на стринг** **length()**

```
s.length()
```
Се одредува должина на стрингот s.

Пример:

```
n=s.length();
```
n ќе добие вредност 33.

- **Должина на стринг** **size()**

```
s.size()
```
Се одредува должина на стрингот s. Функцијата има ист ефект како функцијата length().

Пример:

```
n=s.size();
```
n ќе добие вредност 33.

- **Максимална можна должина на стринг** **max_size()**

```
s.max_size()
```
Се одредува максималната должина која може да ја достигне стрингот s, зависно од системот на кој се извршува апликацијата.

Пример:

```
n=s.max_size();
```
n ќе добие вредност 2 147 483 647.

- **Споредба на стрингови** **compare()**

```
s1.compare(s2)
```
Проверува дали стрингот s1 е еднаков, поголем или помал од стрингот s2.

Споредувањето се врши лексикографски, при што се споредуваат бројните вредности на знаците на стринговите, според табелата ASCII.

позитивна вредност, ако s1 > s2,
негативна вредност, ако s1 < s2.
0 ако s1 = s2,

Ако сите знаци од стринговите се еднакви, а едниот има на крајот празни места, поголем е подолгиот стринг. На пример, "dobar" < "dobar ".

Примери

```
s1 = "dobar";
s2 = "dobre";
```

Со наредбата

`n = s1.compare(s2);` целобројната променлива n ќе добие вредност –1 бидејќи се различни знаците на 3-тата позиција, `'a'`= 97 < `'r'`= 114.

Со наредбите:

`n = s1.compare("dobar");`	n ќе добие вредност 0.
`n = s1.compare(s2);`	n ќе добие вредност –1 (`'a' < 'r'`).
`n = s1.compare("Dobre");`	n ќе добие вредност 1. (`'d' = 100 > 'D' = 68`).

- **Замена на вредностите на два стринга** **swap()**

  `s1.swap(s2)` Ги заменува содржините на стринговите s1 и s2.

  Пример:

  ```
  s1 = "dobar";
  s2 = "dobre";
  s1.swap(s2);
  ```

  s1 ќе добие вредност `"dobre"`, а s2 вредност `"dobar"`.

- **Потстринг од стринг** **substr()**

  `s.substr(`*`позиција, должина`*`)`

  Од стрингот s се издвојува потстринг од позицијата *`позиција`* (= 0, 1... length(s) –1) со должина зададена со променливата *`должина`* (> 0).

  Примери

`s1 = s.substr(17, 7);`	s1 ќе добие вредност `"program"`.
`s1 = s.substr(17, 20);`	s1 ќе добие вредност `"programski jazik"`.
`s1 = s.substr(3, 1);`	s1 ќе добие вредност `""`.
`s1 = s.substr(37, 3);`	Ќе се јави грешка при извршување.

- **Издвојување знак од стринг** **at()**

  `s.at(`*`позиција`*`)` Издвојување на знакот на позиција зададена со променливата *`позиција`* (= 0, 1, 2... s.length() – 1).

  Пример:

  `znak = s.at(9);` znak ќе добие вредност `'d'`.

- **Барање потстринг или знак во стринг**[6] **find() rfind()**

  `s.find(`*`стринг`* или *`знак`*`)` Ја дава позицијата на првото појавување на *`стринг`* или *`знак`* ако се бара одлево надесно.

  `s.rfind(string или `*`знак`*`)` Ја дава позицијата на првото појавување на *`стринг`* или *`знак`* ако се бара оддесно налево.

  Примери

  ```
  znak = 'j';
  n = s.find(znak);
  n = s.rfind(znak);
  ```
  n ќе добие вредност 8.
  n ќе добие вредност 28.

  Ако знакот е празно место `' '`, n ќе добие вредности 3 и 27.

  ```
  potstring = "gram";
  n = s.find(potstring);
  ```
  n ќе добие вредност 20 .

  ```
  potstring = "grad";
  n = s.find(potstring);
  ```
  n ќе добие вредност –1 бидејќи потстрингот не е најден.

- **Бришење знаци од некоја позиција до крајот на стринг** **erase()**

  `s.erase(`*`позиција`*`)` Бришење на знаците во стрингот од позицијата зададена со променливата *`позиција`* (вклучувајќи ја и неа), до крајот на стрингот.

  Пример:

  `s.erase(16);` s ќе добие вредност `"C++ e najdobriot"`.

- **Замена на потстринг со стринг** **replace()**

  `s.replace(`*`позиција, бројзнаци, стринг`*`)`
  Во стрингот s, почнувајќи од позицијата *`позиција`*, потстрингот од следните *`бројзнаци`* знаци се заменува со стрингот *`стринг`*. (Потстрингот од *`бројзнаци`* се брише).

  `s.replace(`*`позиција, бројзнаци, стринг, одпозиција, бројзнаци1`*`)`
  Замената може да се изврши и со потстринг од стрингот *`стринг`*, од позиција `odpozicija` и со должина *`бројзнаци1`*.

[6] Постојат и други функции за барање, како: find_first_of, find_last_of, find_first_not_of.

Пример

```
s3 = " naj naj";
s.replace(9, 7, s3);
```
s ќе добие вредност
`"C++ e naj naj naj programski jazik".`

```
s.replace(9, 7, s3, 1, 3);
```
s ќе добие вредност
`"C++ e naj naj programski jazik"`.

- **Вметнување на стринг (или потстринг) во стринг** **insert()**

`s.insert(`*`позиција, стринг`*`)`
Во стрингот s, почнувајќи од позицијата *`позиција`*, се вметнува стрингот *`стринг`*.

`s.insert(`*`позиција, стринг, одпозиција, бројзнаци1`*`);`
Во стрингот s, почнувајќи од позицијата *`позиција`*, се вметнува потстринг од стрингот *`стринг`*, кој се состои од *`бројзнаци1`* знаци, почнувајќи од позиција *`одпозиција`*.

Примери

```
s3 = " i najlesen";
s.insert(16, s3);
```
s ќе добие вредност
`"C++ e najdobriot i najlesen programski jazik"`.

```
s.replace(16, s3, 0, 6);
```
s ќе добие вредност
`"C++ e najdobriot i naj programski jazik"`.

- **Проверка дали стрингот е празен** **empty()**

`s.empty()` Враќа вредност true ако стрингот s е празен.

Пример

```
string str;
do {
	cout << "Vnesete ime: "; getline(cin, str);
} while(str.empty());
```

Некои од наведените функции за работа со стрингови се илустрирани во следните примери.

Примери

Пример 3.4.1

```
// Funkcii za rabota so stringovi: assign(), append(), length() i compare()

#include <iostream>
#include <string>
#include <iomanip>

using namespace std;

int main() {

    string s, s1, s2, s3, rezultat, potstring;
    char znak;
    int n, m;
    bool tocno;
    s = "C++ e najdobriot programski jazik";
    s1 = "Dobar";
    s2 = "den";
    s3 = "";
    cout << " Dodeluvanje eden string na drug string" << endl;
    cout << "s1 = " << s1 << ", s3 = " << s3;
    s3.assign(s1);
    cout << ", s3=(s1) " << s3 << endl;
    cout << " Spojuvanje na stringovi" << endl;
    s1.append(" ");
    cout << "s1 = " << s1;
   s1.append(s2);
   cout << ", s2 = " << s2 << ", s1 s2 = " << s1 << endl;
   cout << " Dolzina na string" << endl;
   n = s.length();
   cout << "Stringot \"" << s << "\" e dolg " << n << " znaci." << endl;
   n = s1.size();
   cout << "Stringot \"" << s1 << "\" e dolg " << n << " znaci." << endl;
   cout << " Sporedba na stringovi" << endl;
   s1 = "dobar";
   s2 = "dobre";
   n = s1.compare("dobar");
   cout << "Stringot \"" << s1 << "\" sporeden so stringot \"dobar\" dava "
       << n << endl;
   n = s1.compare(s2);
   cout << "Stringot \"" << s1 << "\" sporeden so stringot \"" << s2 << "\" dava "
       << n << endl;
   n = s1.compare("Dobre");
   cout << "Stringot \"" << s1 << "\" sporeden so stringot \"Dobre\"  dava "
       << n << endl;

```

Слика .4.1

```
46      cout << endl;
47      system("Color 17");
48      system("pause");
49      return 0;
50  }
```

Слика .4.1 продол ение

```
 Dodeluvanje eden string na drug string
s1 = Dobar, s3 = , s3=(s1) Dobar
 Spojuvanje na stringovi
s1 = Dobar , s2 = den, s1 s2 = Dobar den
 Dolzina na string
Stringot "C++ e najdobriot programski jazik" e dolg 33 znaci.
Stringot "Dobar den" e dolg 9 znaci.
 Sporedba na stringovi
Stringot "dobar" sporeden so stringot "dobar" dava 0
Stringot "dobar" sporeden so stringot "dobre" dava -1
Stringot "dobar" sporeden so stringot "Dobre"  dava 1
```

Пример 3.4.2

```
1   // Funkcii za rabota so stringovi: swap(), substr() i at()
2
3   #include <iostream>
4   #include <string>
5   #include <iomanip>
6   using namespace std;
7
8   int main() {
9
10      string s, s1, s2, s3, rezultat, potstring;
11      char znak;
12      int n, m;
13      bool tocno;
14      s = "C++ e najdobriot programski jazik";
15      s1 = "Dobar";
16      s2 = "den";
17      s3 = "";
18      cout << " Zamena na vrednostite na dva stringoa" << endl;
19      cout << "Pred zamena: s1 = " << s1 << ", s2 = " << s2 << endl;
20      s1.swap(s2);
21      cout << "Po zamena: s1 = " << s1 << ", s2 = " << s2 << endl;
22      cout << " Izdvojuvanje potstring od string" << endl;
23      cout << "String: \"" << s << "\"" << endl;
24      s1 = s.substr(17, 7);
25      cout << "Potstring (17,7): " << s1 << endl;
26      s1 = s.substr(17, 20);
27      cout << "Potstring (17,20): " << s1 << endl;
28      s1 = s.substr(3, 1);
29      cout << "Potstring (3,1): " << s1 << endl;
30      cout << " Izdvojuvanje znak od string" << endl;
31      znak = s.at(9);
```

Слика .4.2

```
32      cout << "Znakot na 9-ta pozicija e: " << znak << endl;
33
34      cout << endl;
35      system("Color 17");
36      system("pause");
37      return 0;
38  }
```

Слика .4.2 продол ение

```
 Zamena na vrednostite na dva stringoa
Pred zamena: s1 = Dobar, s2 = den
Po zamena: s1 = den, s2 = Dobar
 Izdvojuvanje potstring od string
String: "C++ e najdobriot programski jazik"
Potstring (17,7): program
Potstring (17,20): programski jazik
Potstring (3,1):
 Izdvojuvanje znak od string
Znakot na 9-ta pozicija e: d
```

Решени задачи

Задача 3.4.1
Да се одреди колку пати се јавува некој знак во стринг.

Во програмата на ***слика .4.*** функцијата znakVoString() со која се одредува колку пати се јавува внесениот знак во стрингот (реченицата) е дефинирана по главната функција main(). Затоа, пред main() е наведен прототипот на функцијата znakVoString().

```
1   #include <iostream>
2   #include <string>
3   using namespace std;
4
5   int znakVoString( char, string );
6
7   int main() {    // Broj na pojavuvanja na znak vo string (so funkcija)
8
9       string recenica;
10      char znak;
11      cout << "Vnesete edna recenica: ";
12      getline( cin, recenica );
13      cout << "Vnesete koj znak go barate vo recenicata: ";
14      cin >> znak;
15      int brojNaPojavuvanja = znakVoString( znak, recenica );
16      if( brojNaPojavuvanja > 0 )
17          cout << "\nZnakot " << znak << " vo recenicata \n\"" << recenica
18          << "\"\nse pojavuva " << brojNaPojavuvanja << " pati." << endl;
```

Слика .4.

```
19          else
20              cout << "\nZnakot " << znak << " vo recenicata \n" << recenica
21              << "\nne se pojavuva." << endl;
22
23          cout << endl;
24          system( "Color 17" );
25          system( "pause" );
26          return 0;
27      }
28
29      int znakVoString( char znak, string s ) {
30          int pati = 0;
31          int dolzina = s.length();
32          for( int pozicija = 0; pozicija < dolzina; pozicija++ ) {
33              if( s.at( pozicija ) == znak )
34                  pati++;
35          }
36          return pati;
37      }
```

Слика .4. продол ение

Еден излез од извршување на програмата е:

```
Vnesete edna recenica: Jas sum od Titov Veles
Vnesete koj znak go barate vo recenicata: s

Znakot s vo recenicata
"Jas sum od Titov Veles"
se pojavuva 3 pati.

Press any key to continue . . .
```

Задача 3.4.2

Да се внесе стринг и да се изброи колку има мали букви, а колку има големи букви.

За испитување дали некој znak од стрингот е буква, се користи функцијата isalpha(). За испитување дали буквата е голема се користи функцијата isupper(), а дали е мала буква се користи функцијата islower().

Програмата за задачата е дадена на ***слика .4.4***.

Еден излез од извршување на програмата е:

```
Vnesete string: C++ e najdobar Programski jazik

 Stringot: C++ e najdobar Programski jazik
 ima 2 golemi bukvi i 23 mali bukvi

Press any key to continue . . .
```

```
#include <iostream>
#include <string>
using namespace std;

int main() {    // Broj na golemi i mali bukvi vo string (so for)

    char znak;
    int brojGolemiBukvi = 0, brojMaliBukvi = 0;
    string s;
    cout << "Vnesete string: "; getline( cin, s );
    int dolzina = s.length();
    for( int brojac = 0; brojac < dolzina; brojac++ ) {
        znak = s.at( brojac );
        if( isalpha( znak ) )
            if( isupper( znak ) )
                brojGolemiBukvi++;
            else
                brojMaliBukvi++;
    }
    cout << "\n Stringot: " << s << "\n ima " << brojGolemiBukvi
        << " golemi bukvi i " << brojMaliBukvi << " mali bukvi" << endl;

    cout << endl;
    system( "Color 17" );
    system( "pause" );
    return 0;
}
```

Слика .4.4

Функции за конверзија на стринг во број

Видовме дека при читање стрингови, со операторот >> се чита само првиот збор, т. е. до првото празно место. Затоа, ја разгледавме функцијата getline(), со која се чита цел стринг.

Исто така, видовме дека ако во една програма читаме податоци и со операторот >> и со функцијата getline(), треба да се води сметка дека по читање бројна вредност со операторот >>, останува непрочитан знакот за нова линија, кој посебно треба да се прочита со функцијата cin.get() или со функцијата cin.getline().

За да не се грижиме за грешки при читањето, можеби е наједноставно да ја користиме само функцијата getline() бидејќи и бројните вредности во влезниот поток се низи од знаци (цифри или цифри и други знаци). Ова е овозможено со посебни функции за конверзија на стрингови во броеви, во случај кога се чита бројна вредност.

Воведени се следниве функции:

stoi(s)	– конверзија на стрингот s во цел број од типот int,
stol(s)	– конверзија на стрингот s во цел број од типот long,
stoll(s)	– конверзија на стрингот s во цел број од типот long long,
stoul(s)	– конверзија на стрингот s во цел број од типот unsigned long,
stoull(s)	– конверзија на стрингот s во цел број од типот unsigned long long,
stof(s)	– конверзија на стрингот s во реален број од типот float,
stod(s)	– конверзија на стрингот s во реален број од типот double,
stold()	– конверзија на стрингот s во реален број од типот long double.

Функциите се демонстрирани во следниот пример.

Пример 3.4.3

```
1   // Konverzija na string vo broj
2
3   #include <iostream>
4   #include <string>
5   using namespace std;
6
7   int main() {
8       string s;
9       int brojInt;
10      long brojLong;
11      long long brojLongLong;
12      unsigned long brojNeoznacenLong;
13      unsigned long long brpjNeoznacenLongLong;
14      float brojFloat;
15      double brojDouble;
16      long double brojLongDouble;
17      cout << "Vnesete string \"2123456789\": ";  getline(cin, s);
18      brojInt = stoi(s);
19      cout << "\t\tBrojot od tip int e: " << brojInt << endl;
20      cout << "Vnesete string \"2123456789\": ";  getline(cin, s);
21      brojLong = stol(s);
22      cout << "\t\tBrojot od tip long e: " << brojLong << endl;
23      cout << "Vnesete string \"91234567891234567890\": "; getline(cin, s);
24      brojLongLong = stoll(s);
25      cout << "\t\tBrojot od tip long long e: " << brojLongLong << endl;
26      cout << "Vnesete string \"4123456789\": ";  getline(cin, s);
27      brojNeoznacenLong = stoul(s);
28      cout << "\t\tBrojot od tip unsigned long e: " << brojNeoznacenLong << e
29      cout << "Vnesete string \"18123456789123456789\": ";    getline(cin, s)
30      brpjNeoznacenLongLong = stoull(s);
```

Слика .4.5

```
31      cout << "\t\tBrojot od tip unsigned long long e: "
32          << brpjNeoznacenLongLong << endl;
33      cout << "Vnesete string \"1.23456789\": ";  getline(cin, s);
34      brojFloat = stof(s);
35      cout << "\t\tBrojot od tip float e: " << brojFloat << endl;
36      cout << "Vnesete string \"2.123456789e-308\": ";    getline(cin, s);
37      brojDouble = stod(s);
38      cout << "\t\tBrojot od tip double e: " << brojDouble << endl;
39      cout << "Vnesete string \"1.123456789e+308\": ";    getline(cin, s);
40      brojLongDouble = stold(s);
41      cout << "\t\tBrojot od tip long double e: " << brojLongDouble << endl;
42
43      cout << endl;
44      system("Color 17");
45      system("pause");
46      return 0;
47  }
```

Слика .4.5 продол ение

Еден излез по извршување на програмата е:

```
Vnesete string "2123456789": 2123456789
               Brojot od tip int e: 2123456789
Vnesete string "2123456789": 2123456789
               Brojot od tip long e: 2123456789
Vnesete string "9123456789123456789": 9123456789123456789
               Brojot od tip long long e: 9123456789123456789
Vnesete string "4123456789": 4123456789
               Brojot od tip unsigned long e: 4123456789
Vnesete string "18123456789123456789": 18123456789123456789
               Brojot od tip unsigned long long e: 18123456789123456789
Vnesete string "1.23456789": 1.23456789
               Brojot od tip float e: 1.23457
Vnesete string "2.123456789e-308": 2.123456789e-308
               Brojot od tip double e: 2.12346e-308
Vnesete string "1.123456789e+308": 1.123456789e+308
               Brojot od tip long double e: 1.12346e+308
```

Конверзија на број во стринг

За конверзија на број во стринг се користи функцијата to_string() :

```
to_string(broj)
```

broj може да биде од типот: int, long, long long, unsigned int, unsigned long, unsigned long long, float, double и long double.

На пример, следниот програмски сегмент:

```
int broj1 = 11;
double broj2 = 12.345;
unsigned long long broj3 = 12345678901234567890;
cout << to_strin(broj1) << endl;
cout << to_strin(broj2) << endl;
cout << to_strin(broj3) << endl;
```

ќе отпечати:

```
11
12.345000
123456789012345678 90

Press any key to continue . . .
```

Задачи за вежбање

1. Да се внесе збор и да се отпечати неговиот спротивен збор. (Спротивен збор е оној кој е напишан одназад нанапред. На пример, на otelo спротивен е oleto).
2. Во следниот стринг да се замени зборот Makedonski со The Great.
   „Najgolemiot makedonski car bil Aleksandar Makedonski, sin na Filip II Makedonski.“
3. Да се внесат име (*ime*), презиме (*prezime*) и име на училиштето (*uciliste*) и да се отпечати:
   Jas sum *ime prezime*, ucenik vo ucilisteto *uciliste*.
   (Променливите *ime*, *prezime* и *uciliste* да се отпечатат со големи букви).
4. Да се одреди на која позиција во реченицата од претходната задача почнува презимето.
5. Во даден стринг да се најде растојанието помеѓу два дадени збора кои се содржат во него. (Растојанието е број на знаци помеѓу зборовите). На пример, во задачата 3 растојанието помеѓу зборовите *prezime* и *uciliste* е 23.
6. Да се провери дали внесената реченица (стринг) завршува со точка.
7. Да се внесе стрингот `"123456789"` и да се пресмета:
   ```
   s1 = 1 + 2 + 3 + 4 + 5 + 6 + 7 + 8 + 9;
   s2 = 1 - 2 + 3 - 4 + 5 - 6 + 7 - 8 + 9;
   p1 = 1 · 2 · 3 · 4 · 5 · 6 · 7 · 8 · 9;
   ```
2. Да се внесе 5-цифрен цел број како стринг и да се пресмета збирот на негови-те цифри.
15 Да се изврши циклично поместување на знаците на стринг за к места налево (или надесно). Стрингот да е аргумент на функција, а повратната вредност да биде стринг со поместени знаци.

Прашања за проверка на знаењето

1. Во која библиотека на C++ се наоѓаат функциите за работа со стрингови?
2. Со која функција се врши замена на вредностите на два стринга?
3. Со која функција се одредува должината на стринг?
4. Наведете неколку функции за конверзија на стринг во број?
5. За што се користи функцијата to_string().

Термини

- **Низа** е структуриран тип од истородни податоци.
- **Еднодимензионална низа** е низа со еден индекс.
- **Именувана константа** наречена и **константна променлива** е дефинирана константа која не може да се менува во програмата.
- **Иницијализирачка листа** е листа од вредности кои се доделуваат на елементите на низа.
- **Симболичка константа** се задава со претпроцесорската директива **#define.**
- **Наредбата for базирана на опсег** се користи кога треба да се изврши некоја операција со сите елементи на низа, без да се води сметка за нивниот редослед.
- **Дводимензионална низа** е низа со два индекса.
- **Матрица** е математички термин за дводимензионална низа.
- **Статичка променлива** е онаа која се сместува на фиксна адреса во меморијата.
- **Динамичка променлива** е онаа која се сместува на адресата во меморијата која е слободна во моментот на нејзиното креирање.
- **Покажувач** или **покажувачка променлива** е променлива чија вредност е мемориска адреса.
- **Адресен оператор &** е оператор кој ја враќа адресата на операндот кој е наведен по него.
- **Индиректен оператор *** (**оператор за дереференцирање**) е оператор кој ја враќа вредноста на променливата на која покажува покажувачот по него.
- **new** е наредба за креирање динамичка променлива.
- **delete** е наредба за бришење динамичка променлива.
- **<string>** е библиотека во C++ која содржи функции за работа со стрингови.
- **stoi, stol, stoll, stoul, stoull, stof, stod, stold** се функции за конверзија на стринг во број од типот: int, long, long long, unsigned long, unsigned long long, float, double и long double.
- **to_string()** е функција за конверзија на број во стринг.

Резиме

- Низите се структуриран тип податоци чии елементи се податоци од ист тип.
- Секој елемент во низата има свој реден број, наречен индекс.
- Низата може да биде еднодимензионална (со еден индекс), дводимензионална (со два индекси), тродимензионална (со три индекси) итн.
- На елементите на низа може да им се доделуваат вредности со наредбата за доделување или при декларацијата со иницијализирачка листа.

- Еднодимензионална низа се декларира со: *тип име* [*број*], при што *број* може да биде цел број или именувана константа.
- Димензијата на низа иницијализирана со иницијализирачка листа се одредува автоматски од преведувачот.
- Ако има помалку вредности во иницијализирачката листа отколку што има елементи во низата, останатите елементи на низата се иницијализираат на дифолт вредност, според типот на низата. А, ако има повеќе вредности во иницијализирачката листа отколку што има елементи во низата, се јавува синтаксичка грешка.
- Димензијата на низа може да се зададе и преку симболичка константа зададена со директивата #define.
- Во C++ нема автоматско иницијализирање на низи.
- Индексите на елементите на еднодимензионална низа со должина n во C++ се: 0, 1... n – 1.
- Наредбата for базирана на опсег се користи за операции со сите елементи на низа, без да се води сметка за редоследот на елементите.
- Дводимензионална низа е низа со два индекса.
- Индексите на елементите на дводимензионалната низа со димензии [m][n] во C++ се: 0, 1... m – 1 за првата димензија и 0, 1... n – 1 за втората димензија.
- Дводимензионална низа се декларира со: *тип име* [*број1*] [*број2*], при што *број1* и *број2* може да бидат цел број или именувана константа.
- Дводимензионална низа може да се иницијализира при декларирање или со иницијализирачка листа.
- Ако нема доволно вредности во иницијализирачката листа за иницијализирање на сите елементи на дводимензионалната низа, остатокот се иницијализира со дифолт вредноста на типот на низата.
- Ако дводимензионална низа е иницијализирана со иницијализирачка листа, тогаш за одредување на бројот на редици и на колони во C++ се користи операторот sizeof().
- Дводимензионална низа може да се третира како низа од низи, така што секоја редица да биде еднодимензионална низа.
- Статичките променливи се сместуваат на фиксна адреса во меморијата и остануваат на истата адреса до завршување на програмата.
- Динамичките променливи се сместуваат на адреси кои се слободни во моментот на нивното креирање.
- За пристап до динамичките променливи се користи механизам на покажувачи.
- Покажувачите може да содржат само мемориска адреса.
- Покажувачите може да покажуваат само на променливи кои имаат ист тип како и покажувачот.
- Адресниот оператор & ја враќа адресата на операндот наведен по него.

- Еден покажувач може да се додели на друг покажувач и тогаш двата покажуваат на иста променлива.
- За добивање на вредноста на променлива преку покажувачи, се користи индиректниот оператор *, кој се наведува пред покажувачот кој покажува на променливата.
- Покажувачот може да биде иницијализиран на 0, NULL или на адресата на некоја променлива.
- Покажувач со вредност NULL не покажува на ниту една променлива.
- Вредноста 0 е единствената целобројна вредност која може да му се додели на некој покажувач.
- Името на низа има вредност на адресата на првиот елемент од низата и е од тип покажувач на променливи од типот на елементите на низата.
- Со покажувачите може да се вршат следните операции: инкрементирање, декрементирање, додавање или одземање целобројна вредност и додавање на еден покажувач на друг или одземање на еден покажувач од друг.
- Може да се декларира покажувач на покажувач.
- Со наредбата new се креира динамичка променлива во меморијата.
- Со наредбата delete се брише променлива и се ослободува меморијата што ја зафаќала.
- Ако во програмата се користат стрингови, тогаш мора да се вклучи библиотеката <string> со директивата #include.
- Во C++ постојат многу функции за работа со стрингови.
- Функциите за конверзија на стринг во број во C++ се: stoi, stol, stoll, stoul, stoull, stof, stod, stold.
- За конверзија на број во стринг, се користи функцијата to_string().

ДОДАТОК

ПРОСТИ ТИПОВИ НА ПОДАТОЦИ

Тип	Бита	Опсег		
Цели броеви				
short	16	-32768	до	3276
unsigned short	16	0	до	6553
int	32	-2147483648	до	2147
long	32	-2147483648	до	2147
unsigned int	32	0	до	4294
long long	64	-9e18	до	+ 8e1
Знаци				
char	8	'A'-'Z', 'a'-'z', '0'-'9'...		
Реални броеви				
float	32	+/- 10E-37	до	+/- 1(
double	64	+/- 10E-307	до	+/- 1(
long double	32	+/- 10E-307	до	+/- 1(

ЗНАЦИ ASCII

Б

ASCII	Хексаде-цимално	Знак	ASCII	Хексаде-цимално	Знак	ASCII	Хексаде-цимално	Знак	ASCII	Хексаде-цимално	Знак
0	0	NUL	32	20	(празно)	64	40	@	96	60	`
1	1	SOH	33	21	!	65	41	A	97	61	a
2	2	STX	34	22	"	66	42	B	98	62	b
3	3	ETX	35	23	#	67	43	C	99	63	c
4	4	EOT	36	24	$	68	44	D	100	64	d
5	5	ENQ	37	25	%	69	45	E	101	65	e
6	6	ACK	38	26	&	70	46	F	102	66	f
7	7	BEL	39	27	'	71	47	G	103	67	g
8	8	BS	40	28	(	72	48	H	104	68	h
9	9	TAB	41	29	)	73	49	I	105	69	i
10	A	LF	42	2A	*	74	4A	J	106	6A	j
11	B	VT	43	2B	+	75	4B	K	107	6B	k
12	C	FF	44	2C	,	76	4C	L	108	6C	l
13	D	CR	45	2D	-	77	4D	M	109	6D	m
14	E	SO	46	2E	.	78	4E	N	110	6E	n
15	F	SI	47	2F	/	79	4F	O	111	6F	o
16	10	DLE	48	30	0	80	50	P	112	70	p
17	11	DC1	49	31	1	81	51	Q	113	71	q
18	12	DC2	50	32	2	82	52	R	114	72	r
19	13	DC3	51	33	3	83	53	S	115	73	s
20	14	DC4	52	34	4	84	54	T	116	74	t
21	15	NAK	53	35	5	85	55	U	117	75	u
22	16	SYN	54	36	6	86	56	V	118	76	v
23	17	ETB	55	37	7	87	57	W	119	77	w
24	18	CAN	56	38	8	88	58	X	120	78	x
25	19	EM	57	39	9	89	59	Y	121	79	y
26	1A	SUB	58	3A	:	90	5A	Z	122	7A	z
27	1B	ESC	59	3B	;	91	5B	[	123	7B	{
28	1C	FS	60	3C	<	92	5C	\	124	7C	|
29	1D	GS	61	3D	=	93	5D	]	125	7D	}
30	1E	RS	62	3E	>	94	5E	^	126	7E	~
31	1F	US	63	3F	?	95	5F	_	127	7F	DEL

ЛИТЕРАТУРА

Cormen, T., Leiserson. C., Rivest, R., Stein, C.: *Introduction to ALGORITHMS*, 3th Ed., MIT Press, 2009.

Deitel, P., Deitel, H.: *C++20 for Programmers, Pearson, 3rd edition, 2020.*

Deitel, P., Deitel, H.: *C++ How to Program*, Pearson, 10 edition, 2016.

Deitel, P., Deitel, H.: *C++ How to Program (Early Objects Version)*, Pearson, 9 edition, 2013.

Eckel, B.: *Thimking in C++: Introduction to Standard C++,* Vol.1, Prentice Hall, 2000.

Јованчевски, Ѓ.: *C++ Програмирање,* универзитетски учебник, Гоцмар, Скопје 2018.

Јованчевски, Ѓ., Ацковска, Н., Стојчевска, Б, Јованов, М..: *C++ Збирка алгоритми и програми,* универзитетски учебник, Гоцмар, Скопје 2017, 2007.

Јованчевски, Ѓ., Ацковска, Н., Стојчевска, Б, Јованов, М..: *C++ Програмирање за почетници*, Гоцмар, Скопје 2011.

Јованчевски, Ѓ., Стојова, Р.: *Програмски јазици*, учебник за IV година гимназиско образование, Гоцмар, Скопје 2009, 2006, 2004;

Јованчевски, Ѓ., Лазаревска, Ж.: *Програмски јазици*, учебник за III година гимназиско образование, Гоцмар, Скопје 2009, 2006;

Јованчевски, Ѓ.: *Информатика*, учебник за I година гимназиско образование, Гоцмар, Скопје 2009, 2002.

Јованчевски, Ѓ., Ацковска, Н., Стојчевска, Б.: *C++ Основи на програмирање,* универзитетски учебник, Гоцмар, Скопје 2007.

Јованчевски, Ѓ.: *Алгоритми и програми,* Гоцмар, Скопје 1993.

Knuth, D.: *The Art of Computer Programming,* Volumes 1-4A, 1st Ed., Addison-Wesley Professional, 2011.

Lippman, S., Lajoie, J., Moo, B.: *C++ Primer*, Addison Wesley Professional, 5 edition, 2012.

Horton, I.: Visual *C++ 2013,* Wiley, 2014.

Sedgewick, R.: *Algorithms in C++,* 3th Ed., Addison Wesley Professional, 1998.

Sedgewick, R., Wayne, K.: *Algorithms,* 4th Ed., Addison Wesley Professional, 2011.

Stroustrup, B: *The C++ Programming Language*, 4th Ed., Addison Wesley Professional, 2013.

Stroustrup, B: *Programming: Principles and Practice Using C++*, 2th Ed., Addison Wesley Professional, 2014.

Wilf, H.: *Algorithms and Complexity,* Taylor & Francis, 2002.

```
#include <iostream>
#include <string>
using namespace std;

bool palindrom( int n, string zbor ) {
	if( zbor.at( 0 ) != zbor.at( n - 1 ) )
		return false;
	else {
		if( n > 2 ) {
			zbor = zbor.substr( 1, n - 2 );
			return palindrom( n - 2, zbor );
		}
		else
			return true;
	}
}

int main() {
	string zbor;
	cout << "Vnesete zbor ";
	getline( cin, zbor );
	cout << "Zborot \"" << zbor << "\" ";
	int n = zbor.length();
	if( palindrom( n, zbor ) )
		cout << "E palindrom? " << endl;
	else
		cout << "NE E palindrom? " << endl;

	cout << endl;
	system( "Color 17" );
	system( "pause" );
	return 0;
}
```