1. **Шта је програмски језик?**

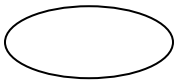
   И рачунари се споразумјевају језицима. Језици које рачунари разумију зову се **програмски језици**. Програмски језик је скуп кључних ријечи и правила за њихово коришћење које "разумије" рачунар.

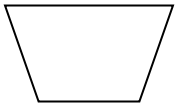
2. **Шта је алгоритам?**

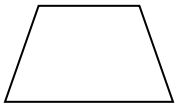
   Рачунар ће извршити сваки коначан низ операција који је написан на "њему разумљив начин". Сваки проблем који желимо ријешити на рачуналу треба знати рашчланити на дијелове или операције које рачунар разумије. Ако смо проблем добро рашчланили, тада је готово свеједно у којем програмском језику ћемо програм написати. Поступак којим рачунар рјешава неки проблем зове се **алгоритам**. Алгоритам је стара ријеч која потјече из арапског језика, а значи *поступак, правило, упутство.* Примјену алгоритама сусрећемо и у свакодневном животу.

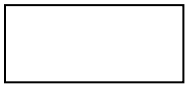
3. **Шта је дијаграм тока?**

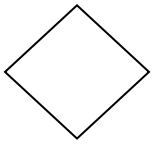
   Алгоритме за рјешавање проблема најчешће приказујемо графички помоћу дијаграма тока. Дијаграм тока графички је приказ алгоритма. Тако приказан алгоритам врло је прегледан и потпуно одређен. Посебно је погодан за анализе програма, тражење сличних рјешења или потребне измјене. При цртању дијаграма тока служимо се посебним знаковима.


почетак


улаз података


излаз података


наредба

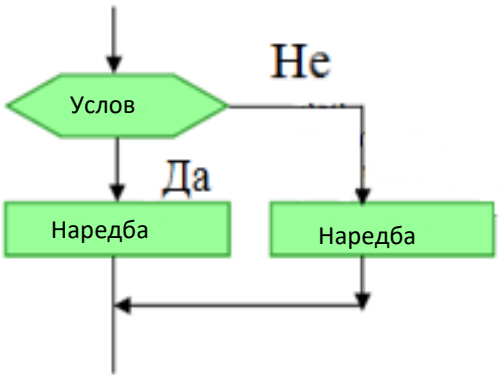

одлука

4. **Шта је карактеристично за линијску програмску структуру?**

   Структура код које се при једном извршавању алгоритма свака наредба, извршава тачно једанпут назива се линијска структура. На слици 1, наредба 1 и наредба 2 су извршене задатим редоследом и извршиле су се тачно једанпут.

5. Шта је карактеристично за разгранату програмску структуру?

Ово је структура у којој се редослед обраде мијења у зависности од услова. На слици 2, наредба 1 се извршава ако је услов тачан (Да), а наредба 2 се извршава ако је услов нетачан (Не). Послије извршавања наредбе 1 или наредбе 2 наставља се са извршавањем следећег



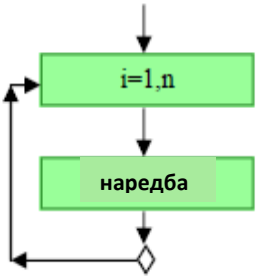
Ако је услов испуњен извршава се наредба 1, а ако услов није испуњен извршава се наредба 2

6. Шта је карактеристично за цикличну порграмску структуру?

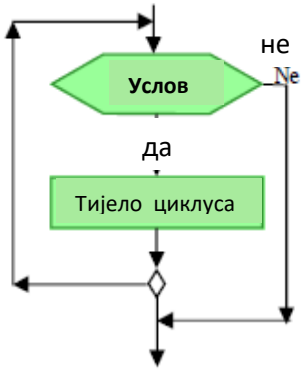
У свакодневном животу често се појеве и догађаји понављају. Није тешко претпоставити да би и у алгоритмима било погодно да се извјестан број алгоритамских корака понавља. Овакав дио алгоритма, који се може више пута понављати, образује цикличну структуру. У зависности од излазног критеријума циклуси се дијеле на бројачке циклусе и циклусе са условима.

Циклус у коме је број понављања циклуса критеријум за излазак из циклуса зовемо **бројачки циклус**.

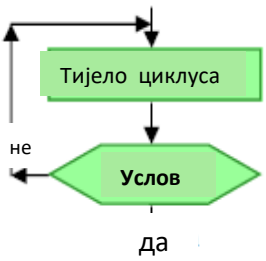
Коришћење бројачких циклуса претпоставља да ми знамо колико пута желимо да се понови дио тијела циклуса. Међутим, некада није јасно колико ће пута да се изврши циклус. Ову могућност пружа условни циклус.



Бројачки циклус



Условни циклус While



Условни циклус Repeat

7. Основне карактеристике програмског језика C++.

Програмски језик C++ је виши програмси језик који је развијен за објектно оријентирано програмирање и био је првобитно развијен у Bell Labs (лабораторij телекомуникацијске фирме Bell) под руководством Бјарне Строустропа током 1980-тих као проширење те му је оригинално име било „C with class“. Због велике потражње за објектно оријентираним језицима те изразитим способностима, стандард за програмски језик C++ ратификиран је 1998. године у стандарду ISO/IEC 14882.

C++ је програмски језик различитих дијалекта, као сто језик има различите дијалекте. У C++ дијалекти не постоје због тога што нетко живи у Крајини или Посавини, већ зато што постоје низ различити комплаилера.

Сваки од тих компаилера је мало другачији. Сваки би требао поштовати ANSI/ISO стандарде. Компаилер ће имати неке нестандардне функције (те функције су сличне различитом сленгу у различитим дијеловима државе). Понекад кориштење нестандардних функција ће створити проблем када покушате комплаирати source код са различитим компаилером.

За разлику од C-а, програмски језик C++ је објектно оријентирани програмски језик.

Замисао објектног програмирања је да се сложени задатак раздијели на мање дијелове који се онда могу међусобно неовисно рјешавати.

Језик је настао осамдесетих година 20. -тог вијека као напреднија верзија C-а. Постоји више верзија преводиоца (компајлера) за C++, а најпознатији су: Dev C++; Microsoft Visual C++ и Borland C++ Builder.

8. Напиши основну структуру програма у C++-у.

Сваки програм написан у C-у или C++ има свој изглед (структуру).

У наставку ћемо показати који су елементи једног програма написаног у C++ програмском језику:

```
#include <cstdlib>
#include <iostream>
using namespace std;
int main()
{

 deklaracija varijabli;
 naredbe programa;

system ("PAUSE");
return 0;
}
```

Линије програма које почињу с знаком # нису програмске наредбе, већ су то претпроцесорске наредбе C програма преводиоцу (компајлеру) које се извршавају прије компајлирања. Преводилац или компајлер је засебан програм који је "задужен" за превођење наредби програмског језика у машински језик.

Наредба **#include <iostream>** позива датотеку iostream која садржи скуп наредби задужених за комуникацију с програмом, а **<cstdlib>** cstdlib која садржи библиотеку стандардних наредби.

Унутар витичастих заграда налазе се декларације варијабли (које ће касније бити описане) те слиједе наредбе програма које чине наредбе за улаз, рачунање и испис резултата.

Наредбом **using namespace std;** обавјештавамо преводитеља да ћемо користити стандардне називе наредби. Та се наредба може замијенити и наредбом #include <stdio> на почетку програма.

Прва линија програма, **main**, дефинира функцију или групу зависних програмских наредби. Функције су саставни дијелови програма у C++у те се та главна или основна зове main. Већина програма садржи још неколико додатних функција које имају своје име.

Свака наредба у програму завршава са тачком-зарезом (;), осим условне наредбе (if-else).

Наредбом **system ("PAUSE");** заустављамо програм док не притиснемо неку типку. То нам је важно приликом извођења програма.

Наредба **return 0;** најчешће стоји на крају програма, а уствари говор да функција main() не враћа вриједност. Када будете учили о потпрограмима тада ћете више сазнати што ради ова наредба.

9. Коментари у програмском језики C++.

Уз програм пожељно је писати и коментар тако да онај ко чита програм може схватити о чему се ради. Коментари се пишу унутар слиједећих знакова /* */ или се било гдје у ретку упишу знакови за двије косе црте //

/* */ - ознака за вишелинијски коментар

// - ознака за једнолинијски коментар

10. Генерисање извршне датотеке

Након што се сачува овај код, треба га компајлирати, тј. произвести извршну машинску датотеку. Ово се изводи коришћењем функцијског тастера F11 на тастатури, или избором опције Cmpile And Runy менију Execute, или притиском на икону у низу алата. Након стартовања процеса компајлирања, појављује се прозор са порукама које прате процес компајлирања. Dev-C++ даје поруку у случају да нађе било какву грешку у програму. У случају да нема грешака, ствара се извршна датотека која се назива име_програма.exe.

11. Процес компајлирања.

Компајлирање C++ програма обухвата неколико корака, који су већином невидљиви за корисника:

- прво, C++ предпроцецсор иде кроз програм и изводи инструкције које су специфициране предпроцесорским директивама (нпр. #include). Резултат овога је модифицирани текст програма који више не садржи никакве директиве.

- затим, C++ компајлер преводи програмски код. Компајлер може бити прави C++ компајер који прави основни (асемблерски или машински) код, или само преводилац, који код преводи у C језик. У другом случају, резултујући C код је затим проведен кроз C компајлер како би се направио основни код. У оба случаја, резултат може бити непотпун због тога што програм позива подпрограмске библиотеке које нису дефинисане у самом програму.

- На крају, линкер завршава објектни код његовим повезивањем са објектним кодом било којег модула библиотека који програм може позвати. Коначан резултат је извршна датотека.

12. **Типови података**

Рачунару је важно с каквим подацима барата. Овисно о врсти разликујемо:

- Цјелобројне типови података (int)
- Реалне типови података (обичне (float) и двоструке прецизности(double))
- Знаковне типови података (char i string)
- Корисничке дефиниране типове (strukture)

Сваки тип податак заузима одређену величину (меморију) у рачунару и има одређени досег или опсег. У сљедећој табели дат је приказ врста података, величине у битовима коју тај податак заузима у меморији рачунара и опсег у којем се може налазити у програмском језику C.

Tip podatka	*Veličina u bitovima*	*Obim*
char	8	-128 do 127
signed char	8	-128 do127
unsigned char	8	0 do 255
short int	16	-32768 do 32767
unsigned int	16	0 do 65535
int	16	-32768 do 32767
long	32	-2147483648 do 2147483647
unsigned long	32	0 do 4294967295
float	32	$3,4x10^{-38}$ do $3,4x10^{38}$
double	64	$1,7x10^{-308}$ do $1,7x10^{308}$
long double	80	$3,4x10^{-4932}$ do $1,1\ x10^{4932}$

Табела1. Типови података

Основни тип цијелог броја је int. Он може имати негативну или позитивну вриједност унутар приказаних граница. Тип података long користи се када нам је за варијаблу потребно веће подручје. Типови података могу бити предзначни signed и то значи да је у једном биту смјештена информација о томе да ли је број позитиван или негативан. Неке варијабле не могу никада попримити негативну вриједност па их декларишемо као непредзначне или unsigned.

Реални бројеви имају децимални дио. Зовемо их још и floating-point (пливајући зарез) бројевима. Име су добили по начину представљања таквих бројева у меморији рачунара. Тај тип података назива се float. Типови података double и long double су попут типа flato, једино омогућавају да се смјети већи број знаменака са већом прецизношћу.

За знаковни тип података користимо се char (што је исто као и signed char). Уколико желимо користити комплетан сет знакова треба употријебити unsigned char. Када баратамо с ријечима и реченицама можемо користити и тип - string (низ знакова).

Често се јавља потреба да код извођења програма желимо задржати садржај екрана да прочитамо резултате извођење програма. За то ће нам послужити наредба getch() која учита један знак и наставља програм, без притиска на типку Enter. Често се користи као посљедња наредба у програму, а налази се у датотеци функција conio.h.

13. Варијабле

Варијабла је симболичко име за меморијску локацију у коју се могу похранити подаци и накнадно их позвати. Варијабле се користе за чување вриједности података тако да се исте могу користити у разним прорачунима у програму. Све варијабле имају двије важне особине:

- Тип, који се поставља када се варијабла дефинише (нпр. цијели број, реални број, карактер, ...) Када се једном дефинише, тип варијабле у C++ се не може промијенити.
- Вриједност, која се може промијенити давањем нове вриједности варијабли. Врста вриједности која се може придружити некој варијабли зависи од њеног типа.

На примјер, int варијабла може да узима само вриједности цијалих бројева (нпр. -5, 13, ..) Када се варијабла дефинише, њена вриједност је недефинисана све док јој се не придружи нека. Придруживање вриједности некој варијабли по први пут назива се иницијализација. Неопходно је обезбиједити да се свака варијабла иницијализира прије него се користи. Такођер је могуће да се варијабла дефинише и инициализира у исто вријеме, што је врло практично.

14. Имена варијабли

Сва имена варијабли требају започињати словом енглеског алфабета или подвлаком _. за остатак имена користе се слова или бројеви. Знакови интерпункције, односно контролни знакови се не смију користити. За препознавање имена варијабли битна су прва 32 знака. Број знакова изнад 32 се занемарује. Постоје разлике између малих и великих слова тј. програм их препознаје као различите знакове у именима варијабли.

Примјери декларација варијабли:

```
int broj;        /*цјелобројна варијабла*/
char znak;       /*знаковна варијабла*/
float temp;      /*децимална варијабла*/
```

Варијабла број декларисана је као цијели број. Варијабла знак декларисана је као знак тј. она може бити било које слово или број, али бројеви који су декларисани као знак не могу се одузимати, сабирати, множити и дијелити нити над њима могу вршити било које математичке операције.

Варијабла темп декларисана је као децимални број.

15. Константе

Константе имају током програма одређену вриједност која се не може промијенити. Кључна ријеч у декларисању и иницијализацији константе је const иза које слиједи тип константе и њезина вриједност.

Примјери константи:

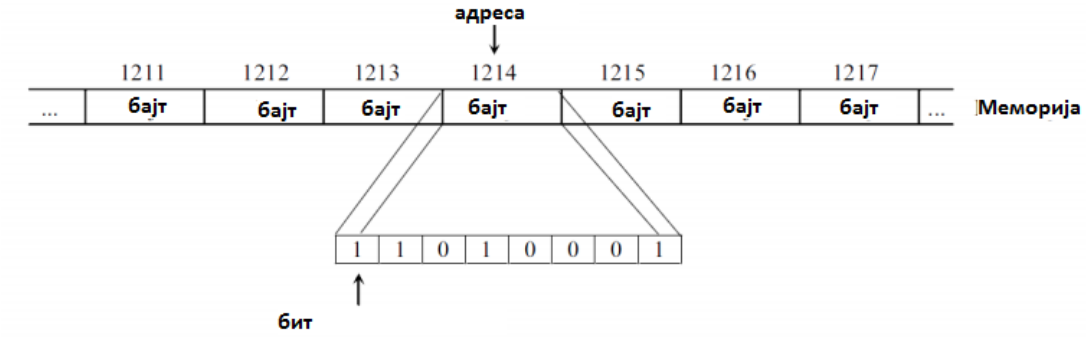
```
const float pi=3.14159;
const int a=8;
```

Правила која вриједе за одређивање имена варијабли вриједе и за имена константи.

16. Коришћење меморије приликом програмирања.

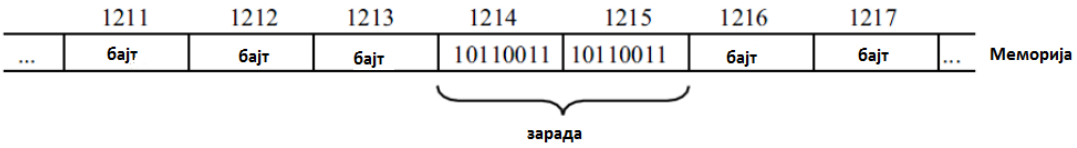
За похрањивање извршног кода као и података са којима прогам манипулише, компјутер има на располагању RAM меморију (Read Access Memory). Меморија се може замислити као

непрекидан низ бита, од којих сваки може да похрани бинарни број (0 или 1). Обично је меморија подијељена на групе од 8 узастопних бита (ово представља бајт, byte). Бајтови су узастопно адресирани, тако да је сваки бајт јединствено представљен својом адресом (Слика 2.).



Слика 2.

C++ компајлер генерише извршни код, који мапира улазне величине на меморијске локације. На примјер, дефиниција варијабле int зарада = 500; наводи компајлер да алоцира неколико бајта како би представио варијаблу зарада. Тачан број бајта који је алоциран и метод који се користи за бинарну репрезентацију цијелог броја зависи од специфичности C++ имплементације, али узмимо да се ради о 2 бајта. Компајлер користи адресу првог бајта на коју се алоцира зарада како би означио варијаблу. Претходна једнакост узрокује да се вриједност 500 похрани у ова два бајта која су алоцирана (Слика 3.)



Слика 3.

Треба напоменути да је организација меморије и коришћење адреса који се односе на податке веома важно за програмера, док тачна бинарна репрезентација података које он користи то није.

17. Улазно/излазне наредбе

Најчешћи начин на који програм комуницира са вањским свијетом је преко једноставних улазно/излазних (IO) операција. C++ омогућује два корисна оператора за ову сврху: >> за улаз, и << за излаз. У ранијем тексту показана је употреба оператора <<. Наредни примјер показује употребу оператора >>.

```
#include <iostream>
using namespace std;
int main () {
int radniDani = 22;
```

```
float radniSati = 7.5;
float satnica, plata;
cout << "Kolika je satnica? ";
cin >> satnica;
plata = radniDani * radniSati * satnica;
cout << "Plata = ";
cout << plata;
cout << endl;
system(“Pause“);
return 0; }
```

Линија 8 чита улазну вриједност, коју уноси корисник и копира је у сатница. Улазни оператор >> узима улазни stream као лијеви операнд (cin је стандардни C++ улазни stream који одговара подацима унесеним помоћу тастатуре), а варијаблу (на коју се копира улазни податак) као десни операнд.

18. Наредба cin

Наредбом cin уносимо податке с тастатуре и придружујемо их варијаблама у програму. Варијабле којима желимо придружити вриједности записујемо иза наредбе cin. На примјер, наредбом

```
cin>>a;
```

уз помоћ тастатуре, придружујемо варијабли а број или слово, а наредбом

```
cin >>a>>b;
```

придружујемо вриједности варијаблама а и б.

Важна напомена: Варијабле претходно требамо декларисати.

19. Наредба cout

Наредбом cout исписујемо текст, вриједности математичких израза и вриједности варијабли у програму. Текст који желимо исписати наводимо унутар наводника, а варијабле чије вриједности желимо исписати пишемо иза наредбе. На примјер, наредбом

```
cout<<"Pozdrav";
```

исписује се поздравна порука Поздрав на монитору рачунара, а наредбом

```
cout <<2+2;
```

исписује се број 4 јер је то вриједност математичког израза 2+2.

Вриједности варијабли исписујемо тако да их наводимо иза наредбе cout. На примјер

```
cout<< a;
```

исписује на монитору рачунала вриједност варијабле а.

20. Аритметички оператори

Ово поглавље обрађује уграђене C++ операторе који се користе за стварање израза, при чему израз представља било какав прорачун који даје неку вриједност. C++ нуди операторе за извршавање аритметичких, ралацијских, логичких, bitwise и условних израза. Такође нуди веома корисне "попратне ефекте" (с side-effect) као што су придруживање, инкремент и декремент. Аритметички оператори C++ нуди пет основних оператора, који су сумирани у Табели 2.

Оператор	Име	Примјер
+	Сабирање	12 + 4.9 // даје 16.9
-	Одузимање	3.98 – 4 // даје 0.02
*	Множење	2 * 3.4 // даје 4.5
/	Дијељење	9 / 2.0 // даје 4.5
%	Остатак при дјељењу	13 % 3 //даје 1

Табела 2. Аритметички оператори

Осим остатка при дјељењу (%) сви аритметички оператори прихватају мијешање цијелих и реалних бројева. Ако су оба операнда цијели бројеви, и резултат је цијели број. Међутим, ако је један од операнада реалан, онда је и резулат реалан (типа double). Када су оба операнда при дијељењу цијели бројеви, резултат је такође цијели број (тзв. цјелобројно дијељење). У том случају резултат се заокружује на доњу вриједност, тј.

9 / 2 // даје 4, а не 4.5!

-9 / 2 // даје -5,а не -4.5!

С обзиром да нежељено цјелобројно дијељење представља једну од најчешћих грешки у програмирању, неопходно је да промијенимо један од операнада да буде реалан број, као нпр.

```
int cijena = 100;
int voulumen = 80;
double JedinicnaCijena / (double) volumen; // даје 1.25
```

Оператор % даје остатак при дијељењу два цијела броја (оба операнда морају бити цијели бројеви),

нпр. 13%3 даје 1

21. Релацијски оператори

Релацијски оператори C++ нуди 6 релацијских оператора за рачунање бројних величина (Табела 3.)

Оператор	Име	Примјер
= =	Једнакост	5 = = 5 // даје 1
! =	Неједнакост	5 ! = 5 // даје 0
<	Мање од	5 < 5.5 // даје 1
<=	Мање или једнако	5 <= 5 // даје 1
>	Веће од	5 > 5.5 // даје 0
>=	Веће или једнако	6.3 >= 5 // даје 1

Табела 3. Релацијски оператори

Треба запамтити да се оператори <= и >= могу корисити само у том облику, а да =< и => не значе ништа у овом контексту. Операнди неког релацијског оператора морају бити бројеви. Но, и карактери су исправни операнди пошто представљају бројну вриједност (сјетимо се ASCII табеле). Релацијски операотри се не смију корисити зе поређење стрингова, пошто се у том

случају пореде њихове адресе, а не садржај. У том случају, резлутат је неодређен. Ипак, постоје функције које могу поредити и лексикографску разлику два стринга.

22. Логички оператори

За комбиновање логичких израза C++ нуди три логичка оператора (Табела 4). Слично релацијским операторима, резултат при кориштењу логичких оператора је 0 (false) или 1 (true).

Оператор	Име	Примјер
!	Логичка негација	! (5 = = 5) // даје 0
&&	Логичко и	5 < 6 && 6 < 6 // даје 1
| |	Логичко или	5 < 6 | | 6 < 5 // даје 1

Табела 4. Логички оператори

Логичка негација је унарни опеаратор, тј. има само један операнд којем даје негативну вриједност.

Инкрементални и декрементални оператори Такозвани ауто инкреметални (++) и ауто декрементални (--) оператори обезбијеђују пригодан начин за повећавање, односно смањивање бројне варијабле за 1. Употреба ових оператора је сумирана у Табели 5, при чему се предпоставља да је int к = 5;

23. Инкремент и декремент оператори

Оператор	Име	Примјер
++	Инкремент (префикс)	++k + 10 //даје 16
++	Инкремент (постфикс)	k++ + 10 //даје 15
--	Декремент (префикс)	--k + 10 //даје 14
--	Декремент (постфикс)	k -- + 10 //даје 15

Табела 5. Инкремент и декремент оператори

Као што се види, оба оператора се могу корисити у префиксном или постфиксном облику. Разлика је велика, јер када се оператор користи у префиксном облику прво се примијењује оператор, а онда се у изразу користи резултат. Када се користи у постфиксном облику, прво се рачуна израз, а онда се примијењује оператор. Оба оператора се могу примијенити како на цјелобројне, тако и на реалне бројеве, иако се ова карактеристика веома ријетко користи на реалним бројевима.

24. Оператори продруживање

Оператор придруживања се користи за похрањивање вриједности на неку меморијску локацију (која је обично придружена некој варијабли). Лијеви операнд оператора треба бити нека лијева_вриједност, док десни операнд може бити произвољни израз. Десни операнд се израчуна и придружи лијевој страни. При томе лијева_вриједност представља било шта што заузима неку меморијску локацију на коју се може похранити нека величина, може бити варијабла, те заснована на поинтерима и референцама. Оператор придруживања може имати много варијанти, које се добијају његовим комбиновањем са аритметичким и bitwise операторима. Ове варијанте су дате у сљедећој табели.

Оператор	Примјер	Еквивалентно са
=	n=25	

+=	n+=25	n=n + 25
-=	n-=25	n-=n - 25
*=	n*=25	n*=n * 25
/=	n/=25	n/=n / 25
%=	n%=25	n%=n % 25

Табела 6. Оператори придруживања

Како оператор придруживања сам по себи представља израз чија се вриједност похрањује у лијеви операнд, он се може користити као десни операнд за наредну операцију придруживања, односно може се написати:

```
int m, n, p;
m = n = p = 100; // znači: n = (m = (p = 100));
m = (n = p = 100) + 2; // znači: m = (n = (p = 100)) + 2;
```

или

```
m = 100;
m += n = p = 10; // значи: m = m + (n = p = 10);
```

25. Условни оператори

Условни оператор треба три операнда (одатле име тернарни). Он има општу формулу:

операнд1 ? операнд2 : операнд3

операнд1 се израчунава, и третира се као логички услов. Ако је резултат различит од нуле, тада се израчунава операнд2. У супротном, израчунава се операнд3. На примјер:

```
int m = 1, n = 2;

int min = (m < n ? m : n); // мин добија вриједност 1
```

Провјерити шта је резултат сљедеће употребе условног оператора:

```
int min = (m < n ? m++ : n++);
```

Постоји још неколико врста оператора (нпр. зарез оператор, sizeof оператор), али о њема неће бити ријечи овдје.

26. Наредба if

Понекад је пожељно да се изврши одређена наредба која зависи од испуњења неког услова. Управо ту могућност пружа if наредба, чији је општи облик:

```
if (izraz)
naredba;
```

Прво се извршава израз, и ако је резултат различит од нуле извршава се наредба. У супротном, ништа се не дешава.
На примјер, ако бисмо жељели провјерити да ли је при дјељењу дјелилац различит од нуле, имали бисмо:

```
if (djelilac != 0)
kolicnik=djelitelj/djelilac;
```

Да бисмо извршили више наредби које овиси о неком истом услову, користимо сложену наредбу, тј. све наредбе стављамо између заграда. Варијанта if наредбе која омогућује да се специфицирају двије алтернативне наредбе, једна која се извршава када је услов испуњен и друга када није, се назива if-else наредба и има облик:

```
if (uslov)
naredba1;
else
naredba2;
```

27. Наредба switch

Switch наредба омогућује избор између више алтернатива, које су засноване на вриједности израза. Општи облик switch наредбе је:

```
switch (izraz)
{ case konstanta_1:
naredbe; ...
case konstanta_n:
naredbe;
default:
naredbe; }
```

Прво се рачуна израз (који се назива switch таг), а затим се резултат пореди са сваком од нумеричких константи (које се називају лабеле), по реду како се јављају, док се не поклопи са једном од компоненти. Након тога се извршавају наредбе које слиједе. Извршавање се изводи све док се не наиђе на наредбу бреак или док се не изврше све накнадне наредбе. Посљедњи случај (дефаулт) може, а и не мора да се користи, и покреће се ако ниједна од претходних константи није задовољена. Класични примјер оцијењивања неког рада на основу освојених бодова дат је у даљем тексу:

```
#include <iostream>
using namespace std;
int main() {
int ocj;
cout << "Unesite ocjenu: ";
cin >> ocj;
switch (ocj) {
case 5: cout << "Imate 90 – 100 bodova" << endl; break;
case 4: cout << "Imate 80 – 89 bodova" << endl; break;
case 3: cout << "Imate 70 – 79 bodova" << endl; break;
case 2: cout << "Imate 60 – 69 bodova" << endl; break;
default: cout << "Imate ispod 60 bodova" << endl; }
system(“Pause“);
return 0;}
```

28. Наредба while

Наредба while (назива се и while петља) омогућује понављање неке наредбе све док је испуњен неки услов. Општи облик ове наредбе је:

```
while (izraz) naredba;
```

Прво се израчунава израз (назива се и услов петље). Ако је резултат различит он нуле тада се извршава наредба (назива се и тијело петље) и цијели процес се понавља. У супротном, процес се зауставља. На примјер, ако бисмо жељели израчунати збир свих бројева од 1 до н, употреба while наредбе би изгледала као:

```
i = 1;
sum = 0;
while (i <= n) sum += i++;
```

Интересантно је да није неуобичајено за while наредбу да има празно тијело петље, тј. нулл наредбу. Такав примјер је проблем налажења највећег непарног фактора неког броја.

```
while (n % 2 == 0 && n /= 2) ;
```

Овдје услов петље извршава све неопходне калкулације, тако да нема потребе за тијелом.

29. Наредба do

Наредба do (назива се и do петља) је слична наредби while, осим што се прво извршава тијело петље, а затим се провјерава услов. Општи облик наредбе је:

```
do
naredba; while (izraz);
```

Прво се извршава наредба, а затим провјерава израз. Ако је израз различит од нуле цијели процес се понавља. У супротном, петља се зауставља. до петља се мање користи него while петља. Обично се користи када се тијело петље мора извршити најмање једанпут без обзира на испуњење услова. Такав примјер је поновљено уношење неког броја и израчунавање његовог квадрата док се не унесе 0:

```
do {
        cin >> n;
       cout << n * n << '\n'; }
while (n != 0);
```

За разлику од while петље, до петља се никада не користи са празним тијелом првенствено због јасноће.

30. Наредба for

Наредба for (for петља) је слична наредби while, али има двије додатне компоненте: израз који се израчунава само једном прије свега, и израз који се израчунава једном на крају сваке итерације. Општи облик наредбе for је:

```
for (izraz1; izraz2; izraz3)
naredba;
```

Прво се израчунава израз1. Сваким пролазом проз петљу се израчунава израз2. Ако је резултат различит од нуле израчунава се израз3. У супротном петља се зауставља.

for петља се најћешће користи у ситуацијама када се нека промјенљива повећава или смањује за неку величину датој итерацији, односно када је број итерација унапријед познат. Сљедећи примјер рачуна збир свих бројева од 1 до n:

```
sum = 0;
for (i = 1; i <= n; ++i)
        sum += i;
```

Било која од компоненти у петљи може бити празна. На примјер, ако се уклоне први и трећи израз, онда до петља личи на while петљу:

```
for (; i != 0;) // je ekvivalentno sa:
while (i != 0) bilo-sta; // bilo-sta;
```

Уклањање свих израза у петљи даје бесконачну петљу:

```
for (;;) // beskonačna petlja
bilo-sta;
```

Пошто петље представљају наредбе, могу се појавити унутар других петљи (тзв. угнијеждене петље).

На примјер:

```
for (int i = 1; i <= 3; ++i)
        for (int j = 1; j <= 3; ++j)
                cout << '(' << i << ',' << j << ")\n";
```

даје парове скупа {1,2,3}

31. Функције

Ово поглавље описује функције, које дефинише корисник, као један од главних грађевинских блокова у C++ програмирању. Функције обезбјеђују прикладан начин упакивања неког нумеричког рецепта, који се може користити колико год је то пута потребно.

32. Дефиниција функције

Дефиниција функције се састоји од два главна дијела: заглавља или интерфејса, и тијела функције. Интерфејс (неки га називају и прототип) дефинише како се функција може користити. Он се састоји од три дијела:

•Имена. Ово је, у ствари, јединствени идентификатор.

•Параметара (или потписа функције). Ово је низ од нула или више идентификатора неког типа који се користе за прослијеђивање вриједности у и из функције.

•Типа функције. Ово специфицира тип вриједности који функција враћа. Функција која не враћа ниједну вриједност би требала да има тип void. Тијело функције садржи рачунске кораке (наредбе) који чине неку функцију. Коришћење функције се изводи њеним позивањем. Позив функције се састоји од имена функције, праћеним заградама за позивање (). Унутар ових заграда се појављује нула или више аргумената који се одвајају зарезом. Број аргумената би требао одговарати броју параметара функције. Сваки аргумент је израз чији тип би требао одговарати типу одговарајућег параметра у интерфејсу функције. Када се извршава позив функције, прво се рачунају аргументи и њихове резултујуће вриједности се придружују одговарајућим параметрима. Након тога се

извршава тијело функције. На крају, функција враћа вриједност (ако иста постоји) позиву. Функције се дефинише на следећи начин:

```
tip_funkcije ime_funkcije(najava liste_argumenata)
{
najava lokalnih varijabli;
naredbe;
return izlazna_vrijednost;
}
```

гдје је:

- **тип_функције** одређује тип вриједности коју позвана функција враћа у надређену функцију,

а може бити било који од основних типова података (int, char, float, double, void),

- **име_функције** је идентификатор преко којег се функција позива, док је

- **листа_аргумената** листа формалних параметара са припадајућим типовима преко којих се низ података из надређене функције преносе у позвану функцију.

- Функција у главни програм враћа једну или нити једну вриједност овисно о типу функције.
- За пријенос вриједности у надређени програм користи се наредба return;.
- Функција се позива на слиједећи начин: **име_функције** (стварна **листа_аргумената**);
- Функције се могу позивати у изразима, петљама, или као аргументи у позивима других функција. Свака кориштена функција се мора најавити прије позива.

33. <u>Поља или низови</u>

Низови су скупине података који представљају једну цјелину. Низ има своје име, тип и величину тј. Заузима одређени простор у меморији рачунара. Низ се састоји од чланова низа који имају своју вриједност и мјесто.

Низове у процесу програмирању користимо када желимо радити с више података који су организирани у редове и колоне, истог су типа и имена. Међусобно се разликују по свом мјесту и својој вриједности. Стога их је лакше премјештати, сортирати и с њима вршити различите рачунске операције.

Низови могу садржавати:

- бројеве (цијеле или децималне)
- знакове (слова и посебне знакове)

Овисно о томе разликујемо нумеричке и знаковне низове. У дефинираном низу, сви подаци морају бити истог типа. Није могуће у истом низу мијешати бројеве и знакове. Сваки члан у низу има своје мјесто. То мјесто зовемо ИНДЕКС. Осим мјеста, члан низа има и своју вриједност.

На примјер: Ако кажемо да желимо дефинирати низ имена A који ће имати 7 чланова и чији ће чланови бити цијели бројеви, онда ћемо у програмском то написати овако:

```
int A [7].
```

Графички приказано то изгледа овако :

A[0]	A[1]	A[2]	A[3]	A[4]	A[5]	A[6]
2	3	5	4	5	34	56

Видимо да сваки члан низа има своје мјесто (индекс) и вриједност. Индекси иду од 0-6 и записани су унутар угластих заграда.

Тако на примјер вриједност првог члана низа A[0] износи 2, другог, A[1], износи 3 итд. Уочимо да чланови низа иду од 0-N-1 гдје је N број који говори колико има чланова низа.

Постоје једнодимензионални, дводимензионални и вишедимензионални низови.

34. <u>Једнодимензионални низови</u>

Једнодимензионални низови имају само један низ података. На примјер низ A приказан у таблели има 7 чланова. Сви чланови су цијели бројеви и налазе се поредани су један иза другога (у низу).

A[0]	A[1]	A[2]	A[3]	A[4]	A[5]	A[6]
34	5	3	21	5	3	76

Како би програм могао радити с низом, на почетку програма потребно је резервирати меморију у рачуналу. То се ради наредбом за декларацију која се уопштено записује:

tip_niza naziv_niza [dimenzije] ;

На примјер сљедеће декларације означавају:

int a[10] – да је декларирaн низ цијелих бројева који се зове a и има укупно 10 чланова, а индекси иду од 0 до 9

float X[8] – да је декларирaн низ децималних бројева који се зове X и има укупно 8 чланова , а индекси иду од 0 до 7

char b[30] - да је декларирaн низ од 30 знакова

Важно је увијек резервирати више простора него што ће се у програму користити јер ће иначе програм неће радити исправно. На примјер ако декларишемо да ћемо користити низ од 10 чланова, а унесемо 20 програм неће моћи прихватити преосталих 10 чланова низа те неће моћи исправно радити.

Низ се може задати унутар програма или уносом с тастатуре (наредбом cin или scanf). Ако се вриједности чланова низа задају унутар програма наводе се унутар витичастих заграда нпр: наредбом

A[6]={2,34,1,67,99,7};

је задан низ A којег чини 6 цијелих бројева.

Када се вриједности чланова низа уносе преко тастатуре (наредбом cin) користи се петља фор. Петљом фор уносе се једна по један члан низа, повећавањем контролне варијабле за један. У наставку је дио програмског кода који показује унос чланова низа помоћу петље for.

```
for (i=0;i<N;i++) {
cin>>X[i];
}
```

За испис чланова низа такођер се користи петља for:

```
for (i=0;i<N;i++) {
cout>>X[i];
}
```

35. <u>Дводимензионални низови</u>

Карактеристика дводимензионалних низова је да имају више редова и колона. Таблицом у наставку представљен је цјелобројни низ назива A.

	1. колона	2. колона	3. колона	4. колона
1. ред	2	4	5	6
2. ред	0	3	9	7

Читајући таблицу, чланови низа су сљедећи:
A[0,0] = 2
A[0,1] = 4
A[0,2] = 5
A[0,3] = 6
A[1,0] = 0
A[1,1] = 3
A[1,2] = 9
A[1,3] = 7
Први број унутар углатих заграда је ред, а други колона.
Ако чланове низа задајемо у програму онда то за, претходно таблицом приказани низ, записујемо овако:
int A[2][4]={{2,4,5,6},{0,3,9,7}};
гдје први број означава број реда, а други број колоне. Као и код једнодимензионалних низова први ред и колона имају редни број (индех) 0.
Дводимензионални низ, се примјерице у пракси користи за рјешавање проблема сортирања низа (од највећег до најмањег или од најмањег до највећег). У наставку је примјер сортирања низа коришћењем једнодимензионалног низа.

36. Низови знакова

За рад с низом знакова можемо користити тип string или char. Тип string се користи у случају ако радимо с ријечима или реченицама, док тип char ради само с једним знаком. Дефинипемо ли поље или низ знакова типа char, такође ћемо моћи радити с ријечима или реченицама.
При рад с типом string на почетку програма потребно је извршити декларацију варијабли наредбом;
string ime_varijable;
Будући да варијабла string приликом уноса унесе знакове једне ријечи, постоји посебна наредба за унос цијеле реченице:

getline(cin, ime_varijable);

Датотеке

Датотеке омогућују похрањивање велике количине података који се могу користити неовисно о програму.
Датотеке могу бити текстуалне или бинарне. У наставку ће бити описане текстуалне датотеке.
2 су основна начина уписа/исписа података у датотеку:
- форматирани
- неформатирани

Неформатирани начин уписа и исписа података из датотеке користи наредбе:
офстреам: ofstreamза уписивање података у датотеку
ifstream: за читање података из датотеке
fstream: за читање и писање података у датотеку
а обавља се комбинацијама кључних ријечи open и close.

37. Објекти и класе

У новије вријеме у концепту програмирања користе се објекти и класе.

Објект у реалном свијету, може бити: аутомобил или бицикл, а у програмској игри лопта или неки лик. Сваки објект је дефиниран стањем и понашањем; на примјер, стање лика у игри може бити: лик трчи, скаче или стоји, а понашање лика можемо дефинисати брзином трчања и висином скока. У програму је објект описан варијаблама које му одређују стање и методама које му одређују понашање.

Класа представља нацрт (предложак) објекта. Стога се увијек прво креирају класе на темељу којих се производе објекти. На примјер, када креирамо класу за лик, у игри можемо креирати више ликова који могу имати другачије стање (нпр:неки лик може летјети) и понашање (има брзину летења).

Погледајте у наставку видео у којем ученици кроз примјере у разреду објашњавају основне појмове објектног програмирања, а то су:

- класе
- објекти
- својства
- методе и
- догађаји