1. Osnove programiranja

1.1. Programiranje i programski jezici

1.1.1. Uvod

-pojmovi **programa** i **programiranja** prisutni su danas na svakom koraku, pogotovo u masovnim medijima (internet, TV, tisak,…)
-pritom se značenje tih pojmova smatra samo po sebi razumljivim, mada sam **postupak programiranja** to najčešće nije
-budući sa smo u svakodnevnom životu okruženi sve **složenijim elektroničkim uređajima** (pogotovo računalima, ali i drugim napravama, npr. perilicama, mikrovalnim pećnicama,…), njihova upotreba i upute za rad sve više u sebi uključuju neki od postupaka **programiranja**, od vrlo jednostavnih do složenijih
*-na primjer, nove perilice rublja imaju ugrađeno sve više digitalnih **senzora** (npr. temperature vode, razine vode i sl.) pa omogućuju sve **učinkovitije**, a time i **štedljivije** pranje, ali potrebno ih je pravilno **programirati***
*-u tu svrhu služi im **LCD pokazivač** i **tipke**, a u radu se naprednije od njih koriste i tzv. **neizrazitom logikom** (engl. **fuzzy logic**) koja nema samo dva **logička stanja** (**0** i **1**, **nisko** i **visoko**), već cijeli skup **međustanja***
-sve to omogućava im bolje rezultate u odnosu na do sada uobičajene perilice, ali korisnik takve perilice mora sve više prilagođavati za rad, tj. programirati
*-na sljedećoj slici prikazana je jedna **moderna perilica** sa čak 7 ugrađenih senzora (senzori temperature, razine i tvrdoće vode, gustoće deterdženta, mase rublja, neravnomjerne raspodjele rublja u bubnju i otvorenih vrata perilice), a zatim i prikaz na njezinom displeju*

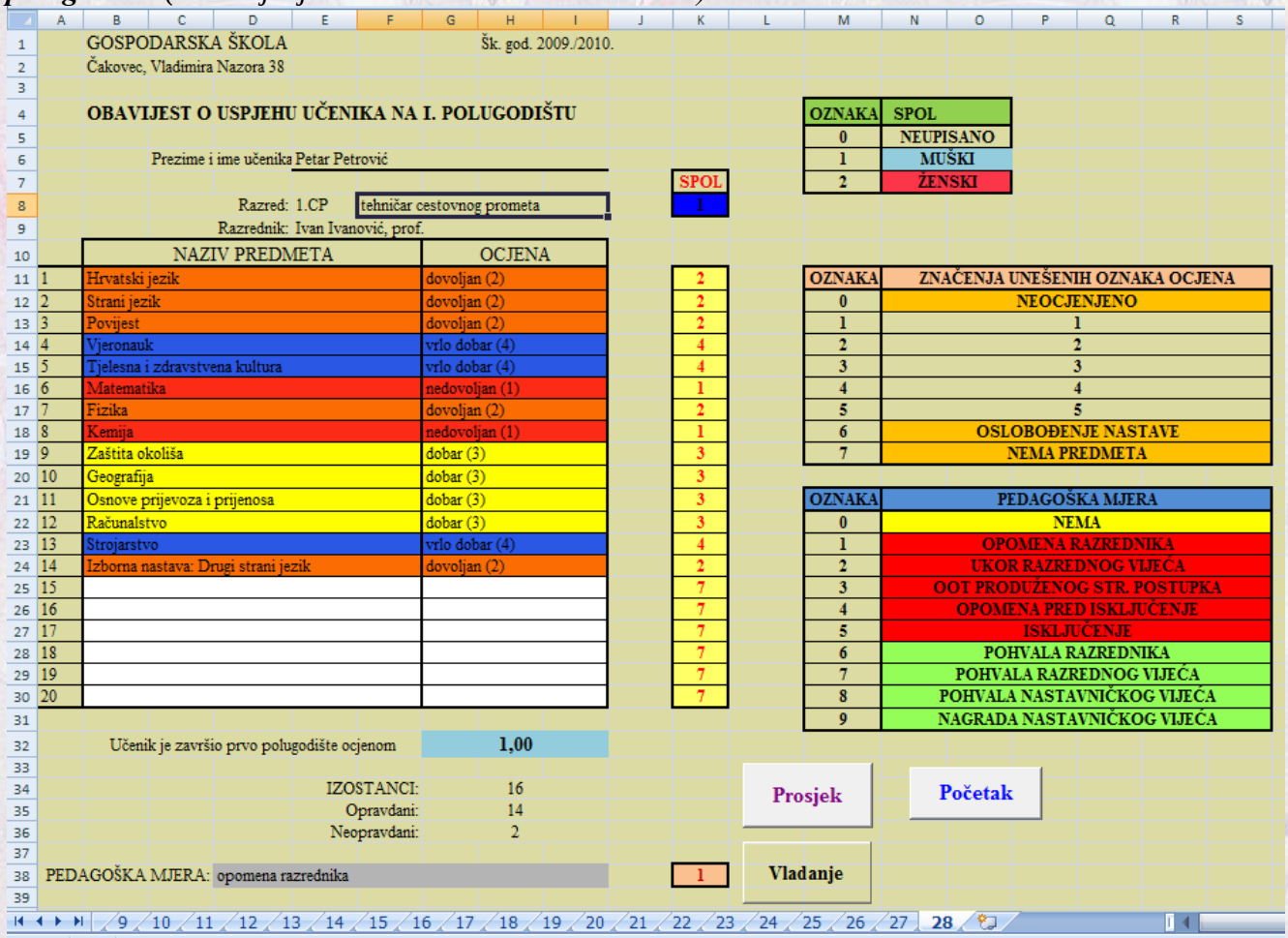
-jednostavan primjer programiranja obrađivali smo prošle godine u **Excelu** kada smo učili korištenje nekih od jednostavnijih **funkcija** (npr. min, max, average, if, sum,...)
-tada to nismo zvali programiranjem, jer pravo programiranje u Excelu može se postići radom u jeziku **Visual Basic for Applications** (**VBA**)
-ipak, u suštini i to je **programiranje**, mada vrlo jednostavno
-evo jedne ilustracije: želimo u Excelu izračunati prosjek učenika iz svih 15 predmeta na kraju školske godine
-neka su ocjene ove: 4, 2, 3, 3, 4, 5, 3, 1, 2, 3, 4, 5, 2, 3, 4
-najjednostavnije bi bilo to riješiti upotrebom funkcije AVERAGE na sve ove ocjene
-prosjek bi bio 3,20 i to bi nam funkcija AVERAGE i ispisala
-za učenika bi to bilo zgodno, jer bi unatoč zaključenoj negativnoj ocjeni iz jednog predmeta prošao sa solidnom trojkom
-u stvarnosti bi morao ići na popravni rok ispraviti tu nepoželjnu ocjenu
-dakle, funkcija AVERAGE sama za sebe ne rješava dobro problem određivanja prosjeka godine

*-međutim, pomoću funkcije **IF** možemo ovisno o **rezultatu neke provjere** (istina ili laž, točno ili netočno) napraviti dvije radnje*
-da se riješi problem traženja prosjeka bilo bi dovoljno provjeriti da li je neka ocjena negativna
-ukoliko postoji bilo koja negativna ocjena ispiše se kao prosjek 1,00; u suprotnom, ispiše se rezultat funkcije AVERAGE
*-dakle, dovoljno je pomoću funkcije **MIN** provjeriti da li je neka ocjena manja od 2, a zatim pomoću funkcije **IF** izračunati prosjek funkcijom **AVERAGE** (kada **nema negativne ocjene**) ili ispisati samo vrijednost **1,00** (u slučaju da postoji neka **negativna ocjena**)*
-formula za navedeni slučaj bila bi (uz ocjene u ćelijama A1 do A15):

=IF(MIN(A1:A15)<2;1,00;AVERAGE(A1:A15))

-zbog preglednosti u formuli su pojedine funkcije obojene različitim bojama
*-sada bismo mogli još više zakomplicirati računanje prosjeka, jer se može provjeravati i neke druge situacije koje daju **krivi prosjek**, npr. da li su unešene sve ocjene, da li su sve ocjene brojevi (npr. moglo bi se unijeti veliko slovo S umjesto 5), da li su ocjene u opsegu od 1 do 5, te jesu li sve ocjene cjelobrojne*
*-formula se u tom slučaju **komplicira**, ali zato je **upotrebljivija u stvarnoj situaciji** od gore navedene*

*-idućom slikom prikazan je prozor Excel datoteke programirane u jeziku **VBA** za **ispis svjedodžbe na polugodištu** (uz izmijenjena imena učenika i razrednika)*



GOSPODARSKA ŠKOLA — Šk. god. 2009./2010.
Čakovec, Vladimira Nazora 38

OBAVIJEST O USPJEHU UČENIKA NA I. POLUGODIŠTU

Prezime i ime učenika: Petar Petrović
Razred: 1.CP — tehničar cestovnog prometa
Razrednik: Ivan Ivanović, prof.

	NAZIV PREDMETA	OCJENA	
1	Hrvatski jezik	dovoljan (2)	2
2	Strani jezik	dovoljan (2)	2
3	Povijest	dovoljan (2)	2
4	Vjeronauk	vrlo dobar (4)	4
5	Tjelesna i zdravstvena kultura	vrlo dobar (4)	4
6	Matematika	nedovoljan (1)	1
7	Fizika	dovoljan (2)	2
8	Kemija	nedovoljan (1)	1
9	Zaštita okoliša	dobar (3)	3
10	Geografija	dobar (3)	3
11	Osnove prijevoza i prijenosa	dobar (3)	3
12	Računalstvo	dobar (3)	3
13	Strojarstvo	vrlo dobar (4)	4
14	Izborna nastava: Drugi strani jezik	dovoljan (2)	2
15			7
16			7
17			7
18			7
19			7
20			7

Učenik je završio prvo polugodište ocjenom: 1,00

IZOSTANCI: 16
Opravdani: 14
Neopravdani: 2

PEDAGOŠKA MJERA: opomena razrednika — 1

SPOL: 1

OZNAKA	SPOL
0	NEUPISANO
1	MUŠKI
2	ŽENSKI

OZNAKA	ZNAČENJA UNEŠENIH OZNAKA OCJENA
0	NEOCJENJENO
1	1
2	2
3	3
4	4
5	5
6	OSLOBOĐENJE NASTAVE
7	NEMA PREDMETA

OZNAKA	PEDAGOŠKA MJERA
0	NEMA
1	OPOMENA RAZREDNIKA
2	UKOR RAZREDNOG VIJEĆA
3	OOT PRODUŽENOG STR. POSTUPKA
4	OPOMENA PRED ISKLJUČENJE
5	ISKLJUČENJE
6	POHVALA RAZREDNIKA
7	POHVALA RAZREDNOG VIJEĆA
8	POHVALA NASTAVNIČKOG VIJEĆA
9	NAGRADA NASTAVNIČKOG VIJEĆA

Prosjek | Početak | Vladanje

*-na slici se vide neki elementi koji se **ne mogu** u Excelu koristiti **bez programiranja** (**gumbi** Prosjek, Početak i Vladanje)*
-budući da je **Excel** aplikacija koja se u nekim zanimanjima koristi vrlo intenzivno (**špediteri**, ekonomisti, **prometni tehničari**,...), znati **osnove programiranja** može biti vrlo korisno
-osnove programiranja korisno je znati i za primjenu u tzv. **skriptnim jezicima** (engl. **script language**) koji omogućuju **automatiziranu primjenu** nekih **programskih alata**
*-npr. u **Photoshopu** se obrada albuma fotografija može automatizirati tako da se svakoj fotografiji podesi automatski kontrast i balans boja upotrebom skriptne datoteke*

-problem kod bilo kojeg programiranja je da **naše iskustvo i način razmišljanja** često **nisu usklađeni s načinom programiranja uređaja i naprava**
-zbog toga se moramo kod programiranja **prilagoditi** traženom objektu (računalo, perilica, mobitel,…)
-prije samog programiranja može se postaviti nekoliko **definicija općih pojmova** vezanih uz programiranje (koje smo učili u 1. razredu)
-**programska oprema** (engl. **software**) – **skup svih programa instaliranih** na nekom računalu
-npr. na računalu imamo instalirane ove programe: Windows 7, Office 2007, AVG, Foxit Reader, Winamp, Irfanview i Corel X4
-svi ti programi čine programsku opremu promatranog računala
-**program** (engl. **program**) – niz **naredbi** (uputa) koje se izvode **točno određenim redoslijedom** da bi se **izvršio neki zadani cilj** (npr. sviranje pjesme, promjena veličine fotografije, ispis na printer,…)
-primjer jednog jednostavnog programa u programskom jeziku Bascom:

```
  Dim Tipka As Byte, Broj As Byte
  Tipka = 0
Broj = 1
  Do
      PortD = 0
      Wait 1
      PortD = 255
      Wait 1
      If Tipka = 1 Then
         Exitdo
      Endif
  Broj = Broj + 1
  Loop Until Broj > 200
  End
```

-u ovom primjeru se najviše 200 puta istovremeno pali i gasi 8 LED-ova (trajanje svjetla i ugašenog stanja je 1 s), osim ako korisnik pritiskom na neku tipku to paljenje i gašenje ne prekine ranije
-**naredba** (engl. **instruction**) – **nalog računalu** za izvršenje neke **jednostavne radnje** (npr. zbrajanje dva broja, pamćenje nekog broja, registriranje pomaka miša,…)
-primjer jedne naredbe:

$$A=A+1$$

(vrijednost A se poveća za 1; ako je A prije ove naredbe bilo 15, nakon nje je 16)
-da bi se **program** mogao **koristiti** na računalu potrebno ga je prethodno **instalirati**
-**instalacija programa** (engl. **program installation**, **program setup**) – postupak kojim se napisani program **priprema za rad**
-pritom se program upisuje (najčešće) na glavnu memoriju računala (najčešće hard disk)
-u tijeku instalacije obično se formira više novih datoteka, mapa i linkova (veza, poveznica), a instalirani program zauzima više prostora u memoriji nego neinstalirani
-program pišu **programeri** u nekom od brojnih **programskih jezika** (npr. C, C++, C##, Visual Basic, QBasic, Bascom,…)
-**programski jezik** (engl. **programming language**) - to je program koji prepoznaje neke **unaprijed zadane nazive** (tzv. **ključne riječi** (engl. **keyword**)) čijim **kombiniranjem po unaprijed zadanim pravilima** pišemo nove programe
-postupak **pisanja programa** zovemo **programiranjem** (engl. **programming**)
-programiranje je **složeni umni postupak** koji zahtijeva prilično **znanja i uvježbavanja** te **upornost**, a može biti **vremenski zahtjevan**
-smatra se da je programiranje (profesionalno) jedan od **najstresnijih** poslova
-veliki problem kod programiranja je postojanje **ogromnog broja programskih jezika** od kojih svaki traži dosta **dugo učenje i vježbanje**

-ipak, većina programskih jezika ima **zajedničke osnovne elemente** koji se mogu u različitim jezicima pomalo **razlikovati načinom upotrebe i pisanja**, ali su **slični**
-cilj nastave ovog predmeta iz dijela o programiranju je svladavanje upravo tog **zajedničkog dijela programskih jezika**, a kao programski jezik izabran je **C**, odnosno **C++**
-mi ćemo na vježbama koristiti besplatni programski alat **wxDev-C++** koji možete skinuti s internetske adrese **http://wxdsgn.sourceforge.net**, a knjigu o programiranju tim programskim alatom na adresi **http://wxdevcpp-book.sourceforge.net**

1.1.2. Programski jezici

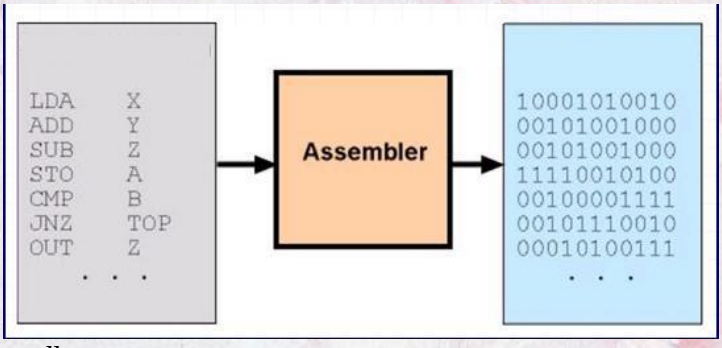
-razvoj programskih jezika pratio je **razvoj računala**
-na početku razvoja računala samo je mali broj ljudi imao mogućnost programirati računalo, a svako računalo tražilo je **drukčije programiranje**
-tek naglim razvojem računalne tehnike računala postaju **dostupnija**, krug programera se širi, a **programiranje se standardizira** uvođenjem **programskih jezika**
-tako počinje programiranje računala u obliku koji nam je danas poznat
-**programske jezike** možemo podijeliti u ovih 5 skupina:

a) **strojni jezici**

-strojni jezik je **najniža razina** programa, a piše se u **binarnom obliku** (cijeli program sastoji se samo od nizova 0 i 1)
-svaki strojni jezik pisan je **samo za određeni procesor** pa je pisanje programa u strojnom jeziku vrlo **složeno** i zahtijeva dobro **poznavanje građe računala**
-zbog toga su te programe u početku razvoja računala pisali sami **konstruktori** računala
-tako pisani programi **nisu se mogli prenositi** na drukčije građena računala
-danas se pisanjem programa strojnim jezikom bave samo **usko specijalizirani stručnjaci**
-u početku razvoja računala njime su se pisali svi programi, ali zbog svoje **nerazumljivosti i teškog pamćenja** takvo programiranje je vrlo brzo potisnuto boljim rješenjima
-primjer naredbe u strojnom jeziku 8-bitnog procesora: 1110101001011010 (naredba povećava sadržaj registra zvanog akumulator za 90; nepotcrtani dio broja (8 bitova) označava naredbu koja povećava sadržaj akumulatora za neki broj, dok je potcrtani dio iznos povećanja sadržaja akumulatora u binarnom obliku (dekadski je to 90))
-za nas kao korisnike računala **strojni jezik nema nikakvo značenje**

b) **simbolički jezici niske razine (asembleri)**

-**simbolički jezici niske razine** (engl. **low level language**) nastali su kako bi ljudima **olakšali programiranje**, jer ljudi lakše **pamte simbole** nego binarne brojeve
-pritom se pojam niska razina odnosi na to da su takvi jezici još uvijek **prilagođeniji računalu**, nego programeru koji ih koristi
-**asembler** (engl. **assembler**) je simbolički jezik u kome je svaka **binarna naredba strojnog jezika predočena odgovarajućim simbolom**
-naredbe strojnog jezika predočuju se simbolima koji se najčešće sastoje od **kombinacije nekoliko slova**
-primjer:
*ADD - zbroji (engl. **add**)*
*SUB - oduzmi (engl. **subtract**)*
*CMP - usporedi (engl. **compare**)*
-takve **kombinacije slova koje se lako pamte**, jer podsjećaju na **značenje** naredbe, a **predočuju strojne naredbe**, zovemo **mnemonicima** (engl. **mnemonic**)
-program napisan u asembleru mora biti **preveden** u binarni oblik da bi ga procesor mogao izvršiti
-**simbole u binarni oblik** prevodi program koji zovemo **jezični prevoditelj** (engl. **language translator**), a često se sam prevoditelj zove asemblerom, mada je asembler u stvari programski jezik
-iduća slika prikazuje odnos naredbi strojnog i asemblerskog jezika i ulogu prevoditelja



-primjer asemblerske naredbe:
ADD A, #100
-ova naredba sadržaj registra procesora zvanog akumulator povećava za 100
-programi pisani u asembleru su nešto **čitljiviji i lakši za razumijevanje** od binarnog zapisa, ali ih je još uvijek **vrlo teško pisati i ispravljati**
-i oni **ovise o vrsti i unutarnjoj građi računala** (procesoru) pa se u načelu mogu izvršavati samo na procesoru za koji su pisani.
-asemblerski programi imaju **veliku brzinu izvršavanja** (rade brže od istih programa pisanih u drugim jezicima više razine)
-danas se uglavnom koriste na **specijaliziranim procesorima** (mikrokontroleri, npr. porodica PIC 32, 8051,...) koji se ugrađuju u različite **elektroničke naprave**, dok se **rijetko koriste na računalima**

c) **viši programski jezici**
-da bi se još više **olakšalo programiranje** i da bi se isti program mogao izvršavati na **različitim računalima** (procesorima) stvoren je niz **simboličkih jezika visoke razine** (engl. **high level language**)
-pojam **visoka razina** odnosi se na to da su naredbe viših programskih jezika mnogo više **nalik govornom jeziku**, **lakše su za pamćenje i upotrebu** od naredbi simboličkih jezika niže razine
-kod simboličkih jezika visoke razine se **više naredbi strojnog ili asemblerskog jezika** predočuje **jednom simboličkom naredbom**
-programi napisani u nekom od viših programskih jezika u načelu su **neovisni o računalu** (točnije o procesoru) na kome se izvršavaju
-primjer programa u jeziku više razine (jezik C):
int a=5;
int b=7;
int c;
c=b-a;
-u tom primjeru pod imenima a i b pamte se dva cijela broja (5 i 7), a njihova razlika pamti se pod imenom c
-**simbolički jezici visoke razine** mogu biti **po namjeni**:
a) **jezici opće namjene** (engl. **general-purpose language**)
-mogu se koristiti **za bilo koje zadatke**
b) **jezici prilagođeni određenoj vrsti problema** (engl. **application-specific language**)
-to su jezici **posebno prilagođeni za određeno područje primjene**
-u drugoj polovini 20.-og stoljeća nastaju programski jezici Fortran, Cobol, Basic, Pascal, C, C++, Visual Basic, Visual C, Java, C## i mnogi drugi
-**C jezik** (autor **Denis M. Ritchie**, 1972. godine) je jezik **opće namjene**, **velikih mogućnosti**, u načelu **neovisan o računalu** na kojem se izvodi
-postigao je vrlo velik uspjeh jer su njime razvijani različiti operacijski sustavi i namjenski programi (programi namijenjeni rješavanju određenih zadataka izravno zanimljivih korisniku)
-programski jezik C **nema mnogo ključnih** (rezerviranih) **riječi**, prema ANSI (engl. *American National Standard Institute*) C standardu samo 32
-C je **modularan jezik** jer omogućava **podjelu programskog zadatka na manje cjeline** koje se mogu **neovisno rješavati i provjeravati**, a po završetku ugraditi u glavni program

-bitno poboljšanje jezika C napravio je oko 1980. godine **Bjarne Stroustrup** (Danac) koji mu je dao podršku za tzv. **objektno orijentirano programiranje** (engl. **object-oriented programming**) te ga nazvao programskim jezikom **C++**

-jezik **C++** u različitim izvedenicama (C++, Visual C++,..) danas je jedan od **najkorištenijih programskih jezika opće namjene**

-program pisan u **jeziku više razine** mora se prije pokretanja pomoću **jezičnog prevoditelja** (engl. **language translator**) **pretvoriti u oblik pogodan za određeni procesor**

-**jezične prevoditelje** dijelimo u dvije grupe:

a) **interpreteri** (engl. interpreter)

-svaku naredbu višeg programskog jezika pretvaraju u naredbe strojnog jezika **tijekom izvršavanja** (takav je npr. **QBasic**)

b) **kompajleri** (engl. compiler)

-svaku naredbu višeg programskog jezika pretvaraju u naredbe strojnog jezika **prije izvršavanja** (takav je npr. **C++**), a rezultat je **izvršna datoteka** (engl. **executable file**, sufiks **.exe**) koja se može **pokrenuti i izvršiti radnje** za koje je programirama

d) **programski jezici prilagođeni krajnjim korisnicima**

-to su jezici kojima se **ubrzava programiranje** , a njima se mogu služiti i **neprogrameri**

-takvi su npr. **upitni jezici za baze podataka** (npr. SQL)

e) **programski jezici neovisni o sklopovlju i operativnom sustavu**

-to su jezici koji se jednom napisani mogu **izvršavati na bilo kojem računalu s bilo kojim instaliranim operativnim sustavom**

-njihovu pojavu potaknuo je razvoj **Interneta** i potreba za **prenosivošću programa** s jednog računala na drugo, **neovisno o računalu i instaliranom operativnom sustavu**

-primjer takvog jezika je **Java**

-programer piše Java naredbe koje se potom prevode u opći oblik, a na ciljnom računalu se dodatno prevode u strojne naredbe tog računala upotrebom tzv. Java prividnog (virtualnog) računala (engl. **Java virtual machine**)

-**programske jezike po građi** (strukturi) dijelimo na:

a) **proceduralne** (engl. **procedural language**)

-program se **dijeli na niz manjih cjelina** od kojih **svaka radi dio ukupnog zadatka**

-npr. za računanje duljine dijagonale pravokutnog trokuta po Pitagorinom teoremu možemo napisati program koji se sastoji od tri dijela: dio za unos podataka (duljine kateta), dio za računanje duljine po formuli $c=(a^2+b^2)^{1/2}$, te dio za ispis rezultata

-primjeri proceduralnih jezika su C, Basic, Pascal,...

b) **objektno orijentirane** (engl. **object-oriented language**)

-umjesto da program dijelimo na cjeline od kojih svaka radi neki zadatak, mi u programu **definiramo objekte koji se sastoje od podataka i operacija koje se mogu provesti na njima**

-potom **vanjske radnje** (npr. pomaci miša po prozoru) **definiraju događanja među objektima**, a time i **tijek programa**

-primjeri takvih programskih jezika su Visual Basic, Visual C,..

1.2. Načela programiranja

-u rješavanju zadataka **čovjek** se služi

a) znanjem

b) iskustvom

c) logičkim rasuđivanjem

d) intuicijom

e) pamćenjem

f) osjećajima itd.

-u rješavanju zadataka **računalo** koristi samo:

a) **pamćenje**

b) **logičko rasuđivanje**

-dakle, da bi računalo riješilo zadatak, zadatak treba **pretvoriti** u oblik koji uključuje samo te dvije sposobnosti (**pamćenje i logičko rasuđivanje**)
-u **pretvorbi zadataka** pomažu nam **pomoćni postupci**
-glavni **pomoćni postupci** za pretvorbu zadatka **u oblik prihvatljiv računalu** su:
a) **planiranje**
b) **analiza zadatka**
c) **algoritam**
d) **pseudokôd**
e) **dijagram tijeka**
-što je **zadatak složeniji**, to je u načelu potrebno **više pomoćnih postupaka**
-što se više napreduje pri rješavanju zadatka, prikaz zadatka postaje sve razlučeniji na manje radnje
-prvi korak u rješavanju zadatka je **planiranje**
-planiranjem se **određuje tko će, kada i što raditi**
-njime se **predviđaju i raspoređuju pojedine faze izrade programa**
-preduvjet da bi se neki **zadatak uspješno riješio** je **znati kako on zapravo glasi**
-zvuči jednostavno, ali to najčešće nije tako
-**analiza zadatka** je **raščlanjivanje i potpuno razumijevanje zadatka i željenih rezultata**
-rezultat analize je tzv. **specifikacija zadatka**
-specifikacija zadatka je dokument koji sadrži **podroban opis zadatka i željenih rezultata**
-specifikacija nije prijedlog kako riješiti zadatak nego **opis onoga što je na raspolaganju i željenog rezultata**
-računalo zadatak može riješiti samo ako dobije **naputak kako to učiniti**
-takav se naputak naziva **algoritam** (engl. **algorithm**)
-**cilj algoritma** je cjelokupni **zadatak svesti na niz jednostavnih, manjih radnji**
-**algoritam** je jedan od koraka pri **pretvorbi zadatka u računalni program**
-obično ga se **prikazuje**:
a) **dijagramom tijeka** (engl. **flow chart**)
b) **pseudokôdom** (engl. **pseudocode**)
-**grafički prikaz algoritma** naziva se **dijagram tijeka**
-dijagram tijeka je koristan jer **pregledno prikazuje algoritam**, **omogućava lakšu analizu i provjeru predloženog rješenja,** te **pronalaženje boljih postupaka rješavanja zadatka**
-dijagram tijeka se sastoji od nekoliko jednostavnih **geometrijskih likova** spojenih **usmjerenim crtama**
-**usmjerene crte** pokazuju **tijek rješavanja zadatka** pa odatle i naziv dijagrama
-**pseudokôd** je tobožnji program (grč. **pseudos** – laž), jer **nalikuje na računalni program**, ali **nije napisan u programskom jeziku**
-sastoji se od **kratkih izraza na govornom jeziku** koji **opisuju i ukratko objašnjavaju pojedine zadatke** algoritma
-osoba koja piše pseudokôd ne mora znati programski jezik i ne mora razmišljati o pravilima pisanja programskog jezika
-pseudokôd bi trebao biti napisan tako da programer može na temelju njega **napisati program u bilo kojem programskom jeziku**
-svaki programski jezik ima vlastiti **ograničeni skup riječi** koje imaju **posebna značenja** i **ne smiju se koristiti u druge svrhe**
-takve se riječi nazivaju **ključnim** (**rezerviranim**) **riječima** (engl. **keyword**, **reserved word**)
-za svaki su programski jezik **propisana pravila slaganja** (**pisanja**) **ključnih riječi u naredbe**
-takva se pravila nazivaju **sintaksa** (engl. **syntax**)
-ako se ne zadovolji propisana sintaksa, program će biti **neispravan** i **neće se moći izvršiti**
-da bi program bio **uporabno koristan**, mora biti **logički ispravan**
-za otkrivanje **logičkih pogrešaka** potrebno je **provjeravati** (**testirati**) **program**
-program provjerava autor programa, više ljudi kod proizvođača progama ili neovisni ispitivači
-**održavanje programa** je **postupak mijenjanja programa** tijekom njegovog "životnog vijeka"
-**održavanje** može biti:

a) **izravno** (npr. temeljem ugovora o održavanju)
b) **neizravno** (npr. izdavanjem novih inačica i ispravaka programa za programe koji se prodaju u velikim količinama (npr. za program Windows izdaju se zakrpe, npr. SP1, SP2,...))
-**dokumentacija** je važan dodatak programu, a **sastoji se** od:
a) **uputa za instaliranje programa**
b) **priručnika za korisnike**
c) **tehničkog opisa programa**
-**programska struktura** opisuje **način i redoslijed izvršavanja pojedinih radnji** koje dovode do **konačnog rješenja zadatka**

1.3. Algoritam - teorija i vježbe

-računalo postavljeni zadatak može riješiti samo ako dobije **upute kako to učiniti**
-pritom se ta uputa mora sastojati samo od različitih **operacija**, npr. aritmetičkih (+, -, *,/,...), relacijskih (<, >, =,...), logičkih (I, ILI, NE,...),...) i **pamćenja** vrijednosti koje uvrštavamo u program (**ulazni podaci**) ili dobivamo kao **međurezultate** i **rezultate (izlazni podaci)**
-niz takvih uputa tvore **algoritam** (engl. **algorithm**)
-izraz **algoritam** nastao je po nadimku jednog **arapskog matematičara i astronoma** iz 9. stoljeća koji se bavio brojevima i aritmetikom
-**cilj algoritma** je cjelokupni zadatak svesti na **niz jednostavnih, manjih postupaka** koji svojim **kombiniranjem** rješavaju cijeli zadatak
-**izvršavanjem** tih **osnovnih radnji** moguće je na temelju **ulaznih podataka** dobiti **rezultat**
-na sljedećoj slici prikazan je suodnos **ulaznih podataka, algoritma i rezultata rada programa**

-većina zadataka se može riješiti na **više različitih načina** pa je za njihovo rješenje moguće napisati **više različitih algoritama**
-autor algoritma redovito nastoji pronaći algoritam koji **najbrže, najučinkovitije i najsigurnije** dovodi do rezultata
-algoritam se mora pisati **jasno i detaljno** tako da ga svaka zainteresirana osoba može shvatiti, a ne samo npr. autor algoritma
-mada to nismo zvali tako, tijekom učenja matematike u osnovnoj i srednjoj školi rješavali smo mnoge zadatke na određeni način, a taj određeni način je u stvari algoritam za rješavanje nekog zadatka
-algoritmima se , kao **uputama za rješenje** nekog zadatka, koriste **stručnjaci skoro svih zvanja**, npr. liječnici, inženjeri, serviseri,...
-npr. algoritam za provjeru da li je neki prirodan broj djeljiv s 3 svodi se na zbrajanje znamenki zadanog broja i djeljenje toga zbroja s 3; ukoliko je ostatak toga djeljenja 0, tada je broj djeljiv s 3
-kod pisanja algoritma bitno je da on mora svojim izvršavanjem od početka do kraja **uvijek za iste ulazne podatke dati isti rezultat** (inače to nije algoritam)
-kod pisanja algoritma služimo se **svakodnevnim govorom**, a ne koristimo naredbe nekog programskog jezika
-kod **algoritma** moraju biti **zadovoljeni ovi uvjeti**:
a) postoje **jasno definirani ulazni podaci** (tzv. **početni objekti**)
b) **završetkom** algoritma moramo dobiti **rezultat** (**izlazni podaci** ili tzv. **završni objekti**)
c) algoritam **za iste ulazne podatke uvijek mora dati iste rezultate**
d) algoritam mora imati **konačan broj postupaka tijekom izvođenja** (tj. **konačni broj koraka** (**instrukcija**) algoritma)
e) zbog potrebe za konačnim brojem koraka svaki **algoritam završava u konačnom vremenu**, tj. mora postojati **kraj algoritma**

f) **svaki korak** (instrukcija) u algoritmu mora biti **izvediv** (npr. u tijeku izvođenja algoritma za djeljenje dva broja, ne smije se javiti slučaj djeljenja s 0, jer takvo djeljenje ne daje definirani rezultat; primjerice, 89/0 ne daje nikakav rezultat)
g) sve **instrukcije algoritma** moraju biti zadane tako da budu **jednoznačne**, tj. ne smije se ostaviti mogućnost da rezultat nekog koraka nije uvijek isti za iste ulazne podatke
-npr. ne smije postojati naredba poput ***povećaj a za 3 ili ga smanji za 1***, jer to nije jednoznačno zadavanje - moguća su dva rezultata koja ovise o nečijoj dobroj volji (čovjeka), dok računalu to ne znači ništa (računalo nema svoje "dobre volje", ipak je riječ samo o stroju)
-**algoritmi po namjeni** mogu biti:
a) **specijalizirani**
-namjenjeni su za rad **samo za neke ulazne podatke** (npr. algoritam za zbrajanje prirodnih brojeva ne koristi se razlomcima kao ulaznim podacima, jer neće dati dobar rezultat)
b) **općeniti**
-kod njih se definira **više skupina** (klasa) **ulaznih podataka** za koje algoritam vrijedi (npr. algoritam za zbrajanje kompleksnih brojeva primjenjiv je i na prirodne, cijele i realne brojeve)
-često se pod pojmom **algoritma** misli samo na **određene postupke** (često matematičke) koji se koriste kao **dijelovi programa**
-npr. imamo algoritam za rotiranje nacrtane kocke na ekranu koji možemo upotrijebiti kao dio programa za crtanje, Euklidov algoritam za traženje najvećeg zajedničkog djelitelja dva prirodna broja itd.
-dakle, osim što cijeli program zovemo algoritmom, često se **specijalizirani i bitni dijelovi programa zovu algoritmima**
-kod izvođenja algoritma **ulazni podaci, međurezultati i rezultati** pamte se pomoću **proizvoljnih imena** (**čim kraćih** radi bržeg pisanja, npr. slova abecede, poput a, d, l, m, riječi poput broj, prvi, drugi,...) koje zovemo **varijablama**
-taj naziv kod **programiranja** označava **mjesto u memoriji** čiji se **sadržaj u tijeku izvođenja programa može mijenjati**
-kod programiranja (i u algoritmima) nastojimo koristiti **čim manje varijabli** da bi program na računalu trošio **čim manju količinu memorije**
-u tom slučaju algoritam može biti **brži u izvođenju**, ali to nije pravilo - lako se može desiti da algoritam s više varijabli bude brži
-kod algoritama je zato često potrebno paziti na to što je **bitnije**: **veća brzina izvršavanja** algoritma (**uobičajeno**) ili **manji utrošak memorije** (**rijeđe**, osim kada se radi o velikoj količini memorije, npr. kod rada sa slikama i videom)
-**primjeri algoritama**:
1.) Napišite algoritam koji unosi dva broja a i b, računa njihov zbroj i ispisuje rezultat na ekranu
-rješenje:

učitaj a
učitaj b
c=a+b
ispiši c na ekranu

2.) Napišite algoritam za izračunavanje vrijednosti kvadratne funkcije $y=5x^2+3x-4$, a rezultat ispišite na ekranu za unešeni x (pretpostavite da u jednom koraku algoritma možete obaviti samo jednu osnovnu matematičku operaciju (zbrajanje, oduzimanje, množenje, djeljenje) nad dva operanda).
-rješenje:

učitaj x
*a=x*x*
*b=5*a*
*c=3*x*
d=b+c
y=d-4
ispiši y na ekranu

-u prijašnjem primjeru **u svakom koraku** koristi se **zasebna varijabla**, što je **nepotrebno trošenje varijabli**
-bitno je napomenuti da mi **istu varijablu možemo koristiti za bilo koliko radnji**, ukoliko nam **njena prijašnja vrijednost više nije potrebna**
-u prijašnjem primjeru dovoljne su nam samo dvije varijable x i y za cijeli zadatak
-**isti primjer riješen sa samo dvije varijable**

učitaj x
*y=x*x //izračunali smo x^2*
*y=5*y //$5x^2$*
*x=3*x //3x*
y=y+x //$5x^2+3x$
y=y-4 //$5x^2+3x-4$
ispiši y na ekranu

-osvrt na prijašnji primjer:
a) **dvije kose crte** (**//**) označavaju da **iza njih slijedi opis trenutnog koraka**
-kod programiranja to zovemo **komentarom**
b) **matematički izrazi** poput **y=y-4 nemaju smisla**, jer nemaju rješenja u skupu konačnih brojeva
-kod ovakvih izraza **ista varijabla s lijeve i desne strane znaka = nema isto značenje**
-**varijabla s desne strane znaka =** označava vrijednost te varijable **neposredno prije trenutnog koraka**
-**varijabla s lijeve strane** označava vrijednost varijable **nakon izvršenja trenutnog koraka**, tj. nakon izračunavanja svega s desne strane znaka =
-u izrazu y=y-4 za y=10 najprije se od prijašnje vrijednosti y (10) oduzme 4
-dobije se 6 i to se pamti kao rezultat ovog koraka algoritma pod imenom y
c) izrazom x=3*x **izgubili smo vrijednost učitanog broja x**, jer nakon toga računanja varijablom x označavamo međurezultat 3*x
-bitno je znati da li se u nekom programu neka od **unešenih ili izračunanih vrijednosti** treba **još negdje koristiti ili nam njena vrijednost više nije potrebna**
-ukoliko nam **vrijednost neke varijable više nije potrebna**, smijemo je **iskoristiti za neku drugu namjenu**
-ipak, ukoliko nam štednja memorije nije presudna u programu, puno je bolje **ostaviti barem ulazne varijable nepromijenjenima**, jer se tada **lakše vrše promjene u programu**, a sam **program je lakši za održavanje i pregledniji**
-slijedi primjer računanja gdje možemo **međurezultat upotrijebiti za ubrzanje rada programa** (manje izračunavanja), a ujedno **i za štednju memorije**
3.) Napišite algoritam za izračunavanje vrijednosti funkcije $z=9x^6+4x^4-7x^3+2x^2+8x-5$, a rezultat za unešeni x ispišite na ekranu (pretpostavite da u jednom koraku algoritma možete obaviti samo jednu osnovnu matematičku operaciju (zbrajanje, oduzimanje, množenje, djeljenje) nad dva operanda).
-rješenje:

učitaj x
*a=x*x //x^2*
*b=a*x //x^3*
*a=2*a //$2x^2$; x^2 nam više ne treba izračunan, pa varijablu a koristimo za iduće međurezultate*
*c=8*x //uveli smo varijablu c za međurezultate, ovdje za 8x*
a=a+c //$2x^2+8x$
a=a-5 //$2x^2+8x-5$
*c=-7*b //$-7x^3$*
a=a+c //$-7x^3+2x^2+8x-5$
*c=b*x //x^4*
*c=4*c //$4x^4$*
c=c+a //$4x^4-7x^3+2x^2+8x-5$
*b=b*b //x^6*
*b=9*b //$9x^6$*

b=b+c //$9x^6+4x^4-7x^3+2x^2+8x-5$
ispiši b na ekranu

-ovakvim korištenjem varijabli program je postao dosta **nepreglednij**i, ali uštedjeli smo na količini memorije potrebne za izvršavanje programa

4.) Napišite algoritam koji će odrediti manjeg od dva učitana broja x i y te ga ispisati na ekranu, a većeg umanjiti za 3 i taj iznos ispisati na ekranu. Ukoliko su oba ulazna podatka ista, treba napisati poruku "Brojevi su isti. Kraj programa." na ekranu te završiti izvođenje programa. U programu upotrijebite samo osnovne matematičke operacije i operacije usporedbe <, > i =.

-rješenje:

učitaj x
učitaj y
c=y-x //traži se razlika da se vidi u kojem su odnosu x i y; isto bi se moglo i pomoću djeljenja, ali
*//**operacija djeljenja je puno sporija od oduzimanja** pa se **izbjegava kada god je to moguće***
ako je c=0, tada napiši na ekranu "Brojevi su isti. Kraj programa." i završi program
ako je c>0, tada na ekranu ispiši x // c je >0, ako je x manji od y, tj. x je manji pa ga ispisujemo
inače //ako je c<0
c=y-3 //veći broj umanjujemo za 3
ispiši y na ekranu

-osvrt na program:

a) najprije smo provjeravali da li je c=0, jer smo tako **uštedjeli jednu dodatnu provjeru** za slučaj da c nije pozitivan (tada može biti c=0 ili je c negativan, a to moramo provjeriti)

-da to nismo tako napravili, program bi u općem slučaju bio **sporij**i, nego da smo to provjeravali na kraju programa

-ujedno je dobra praksa **u programima odvojiti posebne slučajeve od uobičajenih** (u ovom slučaju je posebni slučaj kada su oba broja ista, a uobičajeni kada su različita)

b) kada smo odredili koji je od brojeva veći, više nam **nije potrebna varijabla** c pa smo je na kraju iskoristili za izračun kojim se veći broj umanjuje za 3

1.4. Dijagram tijeka - teorija i vježbe

-stara izreka kaže da jedna slika vrijedi 1000 riječi

-slično bi se moglo reći i za **dijagram tijeka** (engl. **flow chart**)

-on nam služi za **grafičko predočavanje algoritama**, pri čemu je prikazom pomoću **jednostavnih grafičkih simbola spojenih usmjerenim crtama** (strelicama) bitno **lakše pratiti** način funkcioniranja algoritma

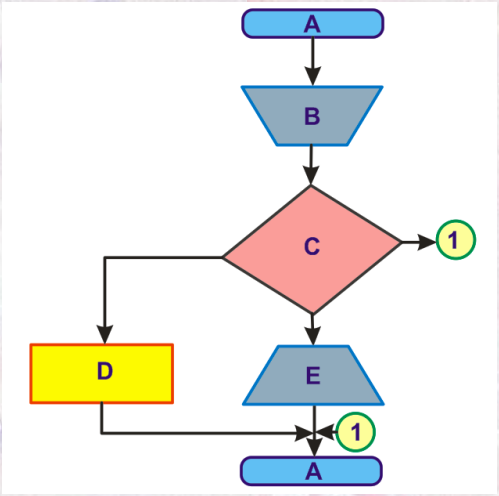
-**usmjerene crte** (**strelice**) pokazuju **tijek rješavanja zadatka**

-dijagram tijeka **olakšava kasniju izradu programa**, a pomoću njega se **lakše uklanjaju pogreške** u algoritmu

-ujedno je pogodan za **analizu problema** i **traženje najboljih rješenja** nekog zadatka

-on je pomoćno sredstvo koje je **neovisno o programskom jeziku i računalu**, dakle, univerzalno je primjenljiv

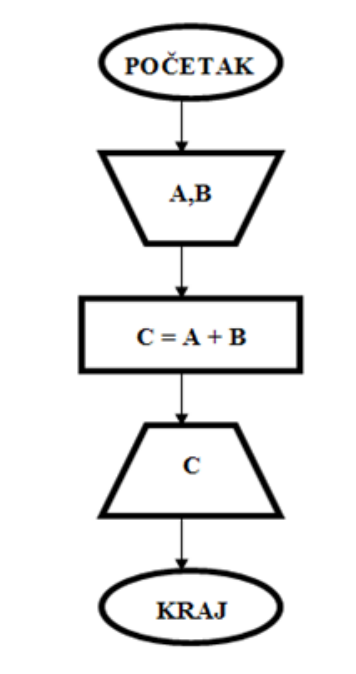
-na idućoj slici su slovima i brojkom označeni **najčešće korišteni simboli u dijagramu tijeka**



-**boje** likova su upotrijebljene samo zbog lakšeg predočavanja vrsta grafičkih likova, a u stvarnim dijagramima tijeka ne moramo ih koristiti
-**slovo A** (ovalni lik nalik na **pravokutnik zaobljenih vrhova** ili **elipsa**) predstavlja **oznaku početka, prekida ili kraja programa**
-taj simbol se **uvijek** koristi barem na dva mjesta u programu: za **početak i kraj**
-ukoliko u nekom dijelu algoritma treba **prekinuti izvršavanje programa**, a ne završiti program, tada se ovaj simbol može upotrijebiti kao **oznaka prekida programa**
-**slovo B** (**trapez s dužom gornjom stranicom**; možemo ga zapamtiti kao lijevak u koji se nešto ulijeva - u ovom slučaju neki podaci) označava **unošenje podataka u program** (tipkovnicom i sl.)
-vrlo je bitan **simbol romba** označen **slovom C**
-on predstavlja simbol tzv. **grananja** (engl. **branching**) u programu
-**grananje** označava da se **ovisno o onome što piše unutar romba** (**uvjet ili vrijednost neke varijable**) uvijek **program nastavlja samo s jednom od strelica koje izlaze iz romba**
-time se može **promijeniti način odvijanja programa**, ovisno na temelju čega se odvija grananje
-**grananje** se **najčešće** odvija tako da postoje **samo dva izlaza** iz romba koji odgovaraju **točnom** (**DA**) ili **netočnom** (**NE**) odgovoru na neko pitanje postavljeno u rombu
-**pitanje postavljeno u rombu** zovemo **uvjet** (engl. **condition**)
-**uvje**t je najčešće **neka provjera** poput one da li je nešto veće od nečega, manje, jednako i sl.
-**grananje ovisno o uvjetu** zove se **uvjetno grananje** (engl. **conditional branching**)
-ukoliko **iz romba izlaze više od dvije strelice**, tada se radi o tzv. **višestrukom odabiru** (engl. **multiple choice**)
-tu se **pitanje u rombu** svodi na to da li je **neki podatak jednak nekoj unaprijed zadanoj cijeloj vrijednosti** (npr. 1, 3, 23, -89, 8990,...), pri čemu se **vrijednost** za koju se **traži jednakost piše na početku strelice koja izlazi iz romba**
-npr. ukoliko se traži da neka varijabla A bude jednaka 11, tada se uz strelicu piše broj 11 i tako dalje za ostale zadane brojeve
-obično se za slučaj **višestrukog odabira** pitanje u rombu zadaje u obliku **varijabla=?**
-u prijašnjem primjeru u rombu bi pisalo **A=?**, a na izlazima iz romba bile bi navedene brojčane vrijednosti
-**simbolom običnog pravokutnika** (slovo **D**) označava se **bilo koja naredba** (**operacija**) ili **više njih** (**osim** onih za koje postoje **posebni simboli**, poput grananja, početka programa,...)
-često se njime prikazuju **različite matematičke operacije**, pri čemu se **unutar pravokutnika može napisati jedna ili više njih**
-ukoliko se u pravokutniku napiše **više operacija**, tada je bitno da se one moraju pisati **od gore prema dolje redom kako se izvršavaju**
-zbog **preglednosti** je lakše da se u pravokutnik upiše samo **jedna operacija ili naredba**
-**simbol trapeza s dužom donjom stranicom** (slovo **E**) označava **izlaženje podataka iz programa** (npr. ispis na ekranu ili printeru)
-**kružići s upisanim brojem** (u ovom primjeru je to broj 1) predstavljaju **priključne točke za povezivanje raznih dijelova programa u cjelinu**
-brojeve u njima zovemo **veznim brojevima**
-**priključne točke** koristimo kod **dužih algoritama** kada se **program proteže na nekoliko stranica** pa je nemoguće spajanje crtama
-na gornjoj slici priključne točke su upotrijebljene samo zbog predočavanja njihove uloge, a ne zbog potrebe
-u promatranom primjeru praktičnije je upotrijebiti samo crtu
-**usmjerene crte** (**crte sa strelicama**) zovu se **linije tijeka programa**
-one **označavaju kojim redom se naredbe izvršavaju** pa je zato potrebno crtati **kamo je strelica okrenuta**
-postoji **još desetak geometrijskih likova** koji se rabe pri izradi dijagrama tijeka, ali su na slici prikazani samo oni koji su **najčešće u upotrebi**
-slijedi nekoliko dijagrama tijeka za zadane algoritme
-**primjer 1**: Unesi dva broja A i B, te ispiši njihov zbroj C.

-algoritam:

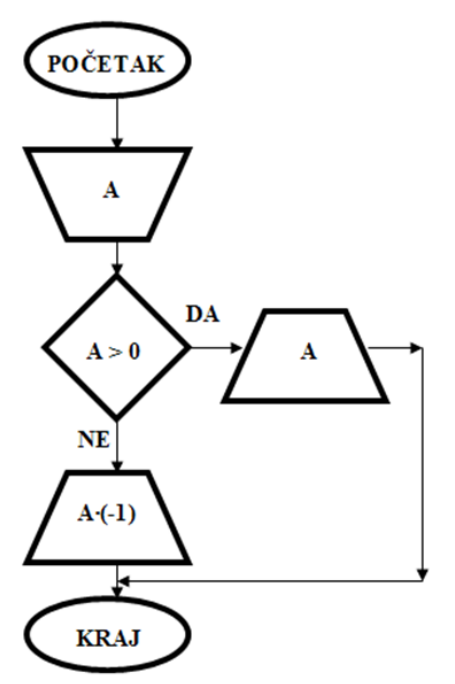
početak
  upiši brojeve A i B
  C=A+B
  ispiši C
kraj



-**primjer 2**: Ispiši apsolutnu vrijednost unešenog broja A.

-algoritam:

početak
  upiši broj A
  ako je A>0
      tada ispiši A
     inače ispiši A(-1)*
kraj



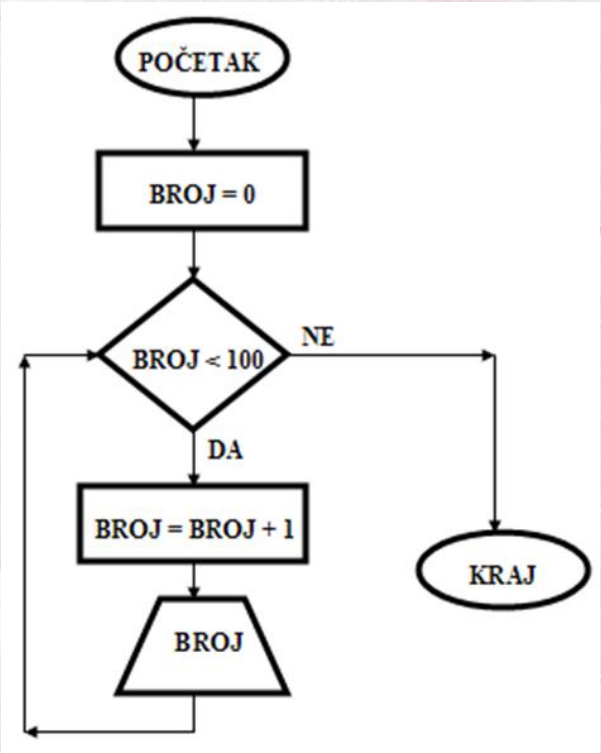
-**primjer 3**:Na ekranu redom ispiši prvih 100 prirodnih brojeva.

-algoritam:

početak
  broj=0
  sve dok je broj<100 ponavljaj
     broj=broj+1 //povećaj broj za 1
     ispiši broj
  kraj ponavljanja
kraj

-napomena: **uvlačenja pojedinih izraza u algoritmima** napravljena su zbog **preglednosti i grupiranja zajedničkih dijelova**

-takva **uvlačenja uobičajeno se koriste u algoritmima**



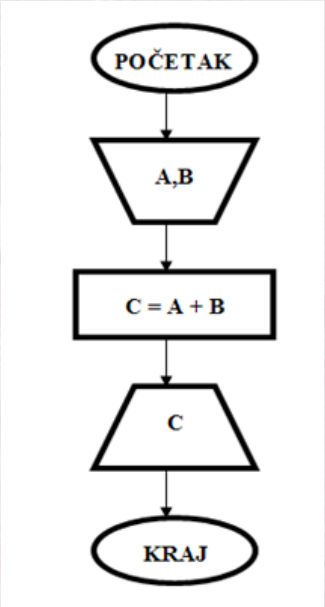
1.5. Programske strukture

-**programske strukture** (**osnovni algoritamski postupci**) definiraju **načine odvijanja programa** zadanog algoritmom
-uobičajeno se koriste ove **tri programske strukture**:
a) **slijed**
b) **grananje**
c) **ponavljanje (petlja)**

1.5.1. Slijed

-**slijed** ili **niz** (engl. **sequence**) odnosi se na slučaj kada **naredbe slijede jedna iza druge i redom se izvršavaju od prve do zadnje bez preskakanja ijedne od njih**
-to je **najjednostavnija programska struktura** koja se koristi kada treba **jednom obaviti neke neuvjetovane radnje**
-primjer broj 1 iz prijašnje cjeline (Dijagram tijeka) primjer je **slijeda**
-evo još jednom njegovog prikaza algoritmom i dijagramom tijeka

početak
upiši brojeve A i B
C=A+B
ispiši C
kraj



1.5.2. **Grananje** (engl. **branching**)

-**grananje** je programska struktura koja omogućuje **različit tijek odvijanja programa ovisno o rezultatu nekog postavljenog uvjeta ili o iznosu nekog cijelog broja**

-o grananju se može ponoviti sve rečeno u cjelini 1.4. (Dijagram tijeka), uključujući primjer algoritma i dijagrama tijeka

-**grananje** se **u dijagramu tijeka** prikazuje **rombom**, a u **algoritmu** se rabi izraz **ako**

-**grananje** označava da se **ovisno o onome što piše unutar romba** (**uvjet ili vrijednost neke varijable**) uvijek **program nastavlja samo s jednom od strelica koje izlaze iz romba**

-time se može **promijeniti način odvijanja programa**, ovisno na temelju čega se odvija grananje

-**grananje** se **najčešće** odvija tako da postoje **samo dva moguća nastavka programa** koji odgovaraju **točnom** (**DA**) ili **netočnom** (**NE**) odgovoru na neko pitanje postavljeno pri grananju

-**pitanje postavljeno kod grananja** zovemo **uvjet** (engl. **condition**)

-**uvje**t je najčešće **neka provjera** poput one da li je nešto veće od nečega, manje, jednako i sl.

-**grananje ovisno o uvjetu** zove se **uvjetno grananje** (engl. **conditional branching**)

-ukoliko **postoje više od dva moguća nastavka programa**, tada se radi o tzv. **višestrukom odabiru** (engl. **multiple choice**)

-tu se **pitanje u algoritmu** svodi na to da li je **neki podatak jednak nekoj unaprijed zadanoj cijeloj vrijednosti** (npr. 1, 3, 23, -89, 8990,...)

-obično se za slučaj **višestrukog odabira** pitanje u algoritmu zadaje u obliku **ako je** pa se zatim **ispod redom navode vrijednosti varijable** i radnje koje se poduzimaju

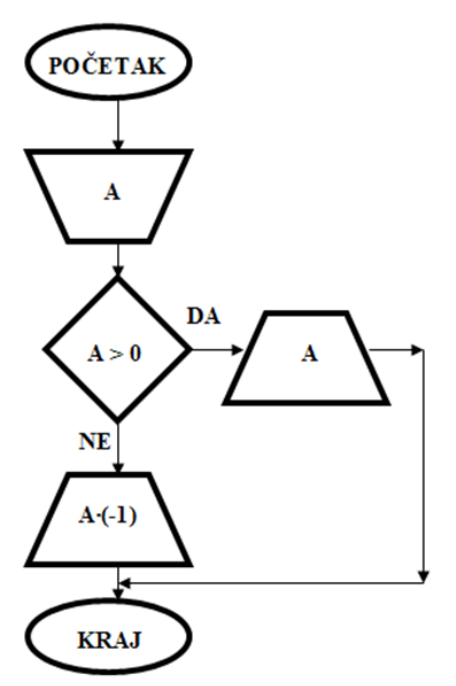
-primjer:

početak
upiši A
ako je
  A=1
  ispiši A
A=234
  *A=A*2*
  ispiši A
A=-23
  A=A+2
  ispiši A
kraj

-slijedi prikaz primjera **grananja** zadanog algoritmom i dijagramom u poglavlju 1.4.

-algoritam:

početak
  upiši broj A
  ako je A>0
      tada ispiši A
     inače ispiši A(-1)*
kraj



1.5.3. **Ponavljanje (petlja)**

-**ponavljanje** (engl. **repeating**) omogućuje da se **određeni niz naredbi izvršava više puta bez da te naredbe moramo ponovo pisati**

-time se **štedi vrijeme** kod programiranja, a ujedno se **smanjuje količina memorije** potrebna za pamćenje programa

-**niz naredbi koje se ponavljaju zajedno s naredbama kojima se definira broj ponavljanja** najčešće se zove **petljom** (engl. **loop**)

-**pri ponavljanju** moguća su ova **dva slučaja**:

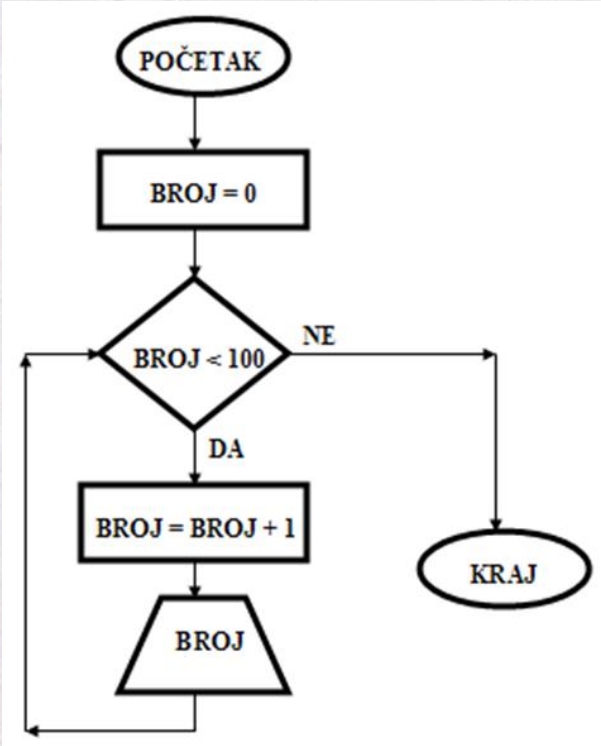
a) **prije početka ponavljanja unaprijed se točno zna koliko puta se vrši ponavljanje**

b) **broj ponavljanja ovisi o rezultatu izvršavanja niza naredbi koje se ponavljaju** pa se zato **unaprijed ne zna broj ponavljanja**

-u tom slučaju **ponavljanje se vrši sve dok se neki uvjet ne ispuni**

-slijedi primjer algoritma i dijagrama tijeka iz poglavlja 1.4. koji odgovaraju programskoj strukturi ponavljanja

početak
broj=0
sve dok je broj<100 ponavljaj
broj=broj+1 //povećaj broj za 1
ispiši broj
kraj ponavljanja
kraj



1.6. Pseudokod - teorija i vježbe

-**pseudokod** (**pseudo jezik**, engl. **pseudocode**, od grč. **pseudos** = laž) predstavlja zapis algoritma koji nalikuje na računalni program, ali **nije napisan programskim jezikom**

-sastoji se od **kratkih izraza na govornom jeziku** koji opisuju i ukratko objašnjavaju pojedine radnje algoritma

-pri zapisivanju algoritma pseudo jezikom treba nastojati zadatak **razložiti na što manje radnje**

-pseudokod omogućuje **preglednij i jednostavniji prikaz** načina na koji se zadatak može riješiti, jer u zapisu **nema ograničenja koja nameću programski jezici**

-na temelju algoritma zapisanog pseudo jezikom programer može napisati program **u bilo kojemu programskom jeziku**

-ne postoji opće prihvaćena norma načina pisanja pseudo jezika, već svaki autor može u načelu rabiti svoj način pisanja, ali postoje neka **načela pisanja** koja se odnose na:

a) **varijable**

-**ime varijable** se u pseudo jeziku može zadati **proizvoljno**, ali bilo bi dobro **ne koristiti dijakritičke i posebne znakove** (npr. ć, Š, Đ, >, =, *….), jer se oni **ne smiju koristiti u programskim jezicima**

-pošto ćemo učiti jezik C/C++, bilo bi dobro da se **međusobno razlikuju velika i mala slova u imenima varijabli**, budući da u C/C++ programskom jeziku varijable napisane velikim i malim slovima nisu iste

-od **posebnih znakova** može se koristiti **podvlaka** (**donja crta**, npr. prvi_broj), dok se **ne smije koristiti razmak** (space) kao dio imena varijable (npr. prvi broj), jer se to smatra **znakom razdvajanja** elemenata u C/C++ programskom jeziku

-varijabli se **vrijednost pridružuje** pomoću **operatora pridruživanja** koji se označava **kombinacijom znaka dvotočke i znaka jednakosti** **(:=)**

-treba razlikovati **običan znak jednakosti** koji se u **pseudokodu** označava znakom **jednako** (a=5 u pseudokodu označava provjeru da li je a jednako 5, dok a:=5 označava da će nakon pridruživanja vrijednosti a postati jednako 5)

-s druge strane**, u nekim programskim jezicima** (ne u programskom jeziku C/C++) **znak jednakosti** (=) označava **i operaciju pridruživanja vrijednosti i operaciju usporedbe vrijednosti**, dok kompajler prepoznaje vrstu operaciju ovisno o mjestu pojave znaka jednakosti i ovisno o tome čime je taj znak okružen

b) **kraj naredbe**

-algoritam zapisan pseudo jezikom sastoji se od niza naredbi

-**svaka naredba završava** znakom **točka-zarez** **(;)**

-**kod složenih naredbi** (**ako je … tada … inače** i **programske petlje**) ovim znakom **ne završavaju dijelovi složenih naredbi**, već **sama naredba**

-primjer:

```
ako je a>5 tada čini //tu ne ide ;, jer je to dio građe složene naredbe
    brojac:=brojac+1; //tu ide ;, jer je ovo obična naredba umetnuta u složenu naredbu
inače čini //tu ne ide ;, jer je to dio građe složene naredbe
    brojac:=brojac-1; //tu ide ;, jer je ovo obična naredba umetnuta u složenu naredbu
završi ako; //ti ide ;, jer je to kraj složene ako - inače naredbe
```

c) **uvlačenje naredbi**

-na prijašnjem primjeru ako – inače naredbe vidi se da se **radi preglednosti uvlače pojedine naredbe**

-to se radi s ciljem da se **lakše uoči na što se odnose te naredbe**

-to **nije potrebno raditi za kompajler**, ali nama **olakšava pisanje programa**

-primjer:

```
ako je b<3 tada čini
a:=a+2;
inače čini
a:=a-2;
ako je b>7 tada čini
a:=b+6;
inače čini
a:=b*2;
završi ako;
završi ako;
```

-ovaj primjer je **nepregledan**, jer koristi dvije ako – inače naredbe (svaka mora završiti sa završi ako; (zato su te dvije naredbe napisane uzastopce)) i nije lako na brzinu uočiti da li prva ako naredba za slučaj inače u sebi uključuje drugu ako – inače naredbu (u promatranom primjeru to je slučaj) ili nakon prve kompletne ako – inače naredbe počinje druga

-isti primjer napisan pomoću **uvlačenja** dijelova koda puno je pregledniji:

```
ako je b<3 tada čini
    a:=a+2;
inače čini
    a:=a-2;
    ako je b>7 tada čini
        a:=b+6;
    inače čini
        a:=b*2;
    završi ako;
završi ako;
```

d) **operatore**

-**operatori** (engl. **operators)** su **simboli** koji **predstavljaju** (**zamjenjuju**) **određene matematičke ili logičke operacije**

-**uobičajeni operatori** se mogu podijeliti u skupine **prema vrsti operacije** koju predočuju na:

1. **aritmetičke operatore**
2. **logičke operatore**
3. **operatore uspoređivanja** (**relacijske operatore**)

-za ispravno zapisivanje algoritma pseudo jezikom treba poznavati **značenje pojedinih operatora**

-pritom može zbunjivati **sličnost operatora** pseudo jezika i operatora kojima se koristimo u matematici ili u programskim jezicima (ponekad izgledaju isto, ali imaju različita značenja)

-u idućim tablicama dani su prikazi uobičajenih operatora u **pseudo jeziku, jeziku Pascal i u jeziku C/C++**

Aritmetički operatori

Opis	Pseudo jezik	Pascal	C/C+ +
Zbrajanje	+	+	+
Oduzimanje	-	-	-
Množenje	*	*	*
Dijeljenje	/	/	/
Cjelobrojno dijeljenje	DIV	DIV	/
Ostatak cjelobrojnoga dijeljenja	MOD	MOD	%

-**napomena**: operator / u **pseudo jeziku** označava **dijeljenje realnih brojeva** (rezultat ima decimalnu točku), dok u **C/C++** jeziku isti znak (/) označava **i cjelobrojno i dijeljenje realnih brojeva**, ali rezultat (time i vrsta provedene operacije) ovisi o **vrsti brojeva koji se nalazi oko znaka /**

-ako je **bilo koji** od njih **realan** broj, rezultat je isto **realan** broj, a vrši se dijeljenje **realnih** brojeva, dok u slučaju da su **oba broja cijela**, mora i rezultat biti **cijeli** broj (vrši se dijeljenje **cijelih** brojeva)

-primjer u **pseudo** jeziku:

```
a:=25;
b:=4;
c:=10.0;
d:=a/b; // d je 6.25 – radi se o dijeljenju realnih brojeva
e:= c DIV b; // d je 2 – radi se o dijeljenju cijelih brojeva
```

-sličan primjer u **C/C++** programskom jeziku daje druge rezultate (koristimo operator / umjesto DIV)

```
a:=25;
b:=4;
c:=10.0;
d:=a/b; // d je 6 – radi se o dijeljenju cijelih brojeva
e:= c/b; // d je 2.5 – radi se o dijeljenju realnih brojeva
```

Logički operatori

-**logički podaci** su podaci koji mogu poprimiti samo jednu od **dvije moguće vrijednosti**

-to su na primjer **true/false** (**istina/laž**), **da/ne**, **1/0** i sl.
-**varijabla** u koju se pohranjuju podaci ove vrste može poprimiti vrijednosti **true** (**1**) ili **false** (**0**)
-ta dva načina označavanja (**true/false** i **1/0**) mogu se **ravnopravno** koristiti u **pseudo** jeziku i kod **programiranja**
-**uobičajeno** se koristi **kraći zapis**, dakle **brojčani**
-za rad s logičkim podacima postoje **logičke operacije**
-logičke operacije zapisuju se **logičkim operatorima**
-rezultat rada **logičkih operatora** je podatak **logičkog tipa**
-slijedi prikaz **načina zadavanja logičkih operatora** i njihove **tablice stanja**
-napomena: **tablice stanja** definiraju **ponašanje logičkih operatora**

Opis	Pseudo jezik	Pascal	C/C+ +
Logički I	I	AND	&&
Logički ILI	ILI	OR	||
Logički NE	NE	NOT	!

Tablica stanja operatora I

A	B	A I B
0	0	0
0	1	0
1	0	0
1	1	1

Tablica stanja operatora ILI

A	B	A ILI B
0	0	0
0	1	1
1	0	1
1	1	1

Tablica stanja operatora NE

A	NE A
0	1
1	0

-primjer upotrebe **logičkih operatora** (**zagrade** su **obavezne oko logičkih izraza**):

```
a:=0;
b:=1;
e:=(a I b);
f:=(a ILI b);
g:=(NE a);
```

-rezultati logičkih operacija iz prijašnjeg primjera su:
e=0
f=1
g=1

Operatori uspoređivanja (relacijski operatori)

-dva se podatka mogu **uspoređivati** s ciljem **donošenja nekih odluka**
-ako je napisani izraz **istinit**, rezultat usporedbe će biti **1** (**true**), a **ako nije**, rezultat će biti **0** (**false**)
-uspoređuje se uporabom **operatora usporedbe** čiji je pregled dan idućom tablicom

Opis	Pseudo jezik	Pascal	C/C+ +
Manje	<	<	<
Manje ili jednako	<=	<=	<=
Veće	>	>	>
Veće ili jednako	>=	>=	>=
Jednako	=	=	= =
Različito	<>	<>	!=

-primjer uporabe operatora uspoređivanja (izrazi su u zagradama):

```
a:=(5<13);
b:=(8<=8);
c:=(5=8);
```

d:=(5<>(2+3)); //tu se prvo napravi zbrajanje (unutrašnje zagrade), pa se zatim vrši usporedba

Rezultati usporedbe će biti:
a=1
b=1
c=0
d=0 //5 nije različito od 5 (5 =2+3)

e) redoslijed izvršavanja operatora

-pri zapisivanju **složenih izraza** pseudo jezikom važno je imati na umu **redoslijed izvršavanja operatora** (**prioritet izvršavanja operacija**)

-**izrazi u zagradama imaju najviši prioritet** (kao i u matematici, **unutrašnje zagrade imaju prednost pred vanjskim u složenim izrazima**)

Redoslijed izvršavanja	Operatori
1.	( )
2.	NE
3.	* / DIV MOD I
4.	+ - ILI
5.	<, <=, >=, <>, =

-**svi operatori unutar iste grupe** (npr. 3. grupa * / DIV MOD I) imaju **isti prioritet**, a **redoslijed izvršavanja ovisi o tome koji je operator napisan bliže lijevoj strani izraza**

-primjer redoslijeda izvršavanja operatora:

*x := 22 DIV 5 * 11 MOD 3;*

-pošto su svi su **operatori ravnopravni**, izraz se izvršava **s lijeva u desno** ovim redoslijedom:

1. 22 DIV 5 = 4
2. 4 * 11 = 44
3. 44 MOD 3 = 2 (ostatak dijeljenja 44/3)
4. x=2

-primjer redoslijeda izvršavanja operatora u izrazu gdje postoje **zagrade**:

*x := (22 DIV 5) * (11 MOD 3);*

-zagrade poništavaju prioritete operatora pa se izraz izvršava ovim redoslijedom:

1. 22 DIV 5 = 4
2. 11 MOD 3= 2
3. 4 * 2 = 8
4. x = 8

-još jedan primjer redoslijeda izvršavanja operatora:

*x := 3*4+6/3 – (55 MOD 6);*

-obzirom na prioritete, operacije se izvršavaju ovim redoslijedom:

1. 55 MOD 6 = 1
2. 3 * 4 = 12
3. 6 / 3= 2
4. 12 + 2 = 14
5. 14 – 1 = 13
6. x = 13

f) **funkcije**

-to su **izdvojeni nizovi naredbi** koji čine **logičke cjeline**, a obavljaju **točno utvrđene zadatke**

-moguće je stvoriti **vlastite funkcije** pa ih zatim rabiti u svom programu ili **koristiti već postojeće** (**ugrađene**), za uporabu pripremljene funkcije

-tendencija je da se **cijeli program sastoji od niza funkcija** čime se dobija **pregledniji i kraći** program koji se **lakše održava**

-slijedi prikaz nekih **čestih matematičkih funkcija**:

Opis	Pseudo jezik	Pascal	C/C+ +
Apsolutna vrijednost realnoga broja	Abs(x)	Abs(x)	abs(x)
Drugi korijen realnoga broja	Sqrt(x)	Sqrt(x)	sqrt(x)
Zaokruživanje realnoga broja na najbliži cijeli broj	Round(x)	Round(x)	round(x)
Najveći cijeli broj manji ili jednak od x	Trunc(x)	Trunc(x)	ceil(x)

2. Izrada programa u programskom jeziku C++

2.1. Osnovni pojmovi i nastanak programa

-jezik **C++** je jezik **opće namjene** za profesionalnu uporabu čiji nastanak seže u osamdesete godine 20.tog stoljeća (autor jezika je Bjarne Stroustrup)
-na tržištu postoji **više inačica** (**verzija**) prevoditelja jezika C++
-neke su **komercijalne** (plaćaju se), a neke **besplatne**
-**najpoznatije** su:

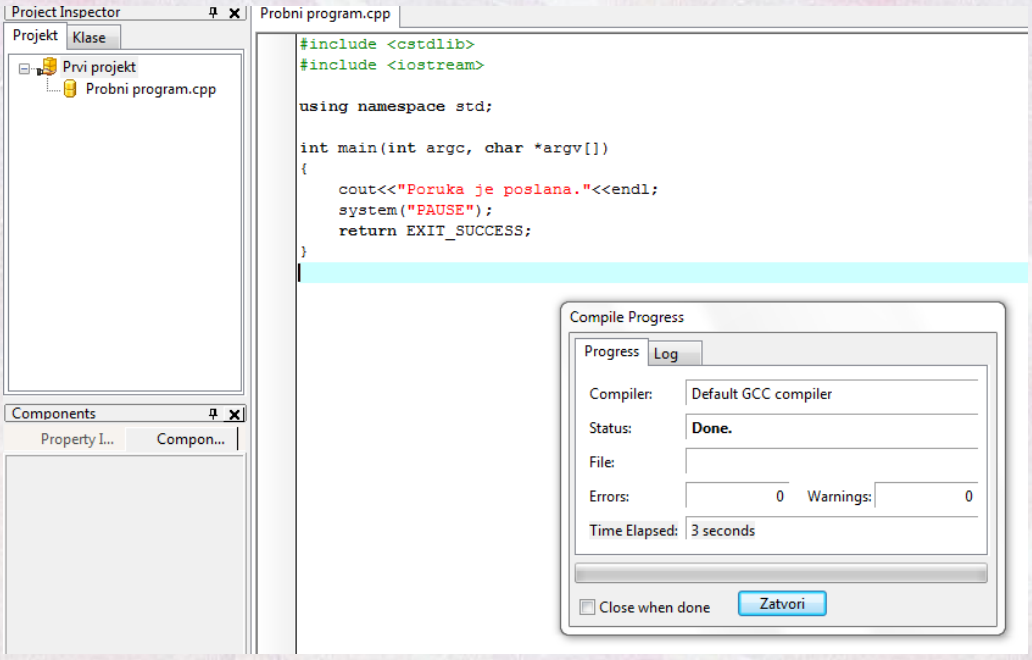
a) **Microsoft Visual C++**
b) **Borland C++ Builder**
c) **wxDev-C++** (to je vezija u kojoj ćemo mi raditi; sve upute se dalje odnose na tu verziju)

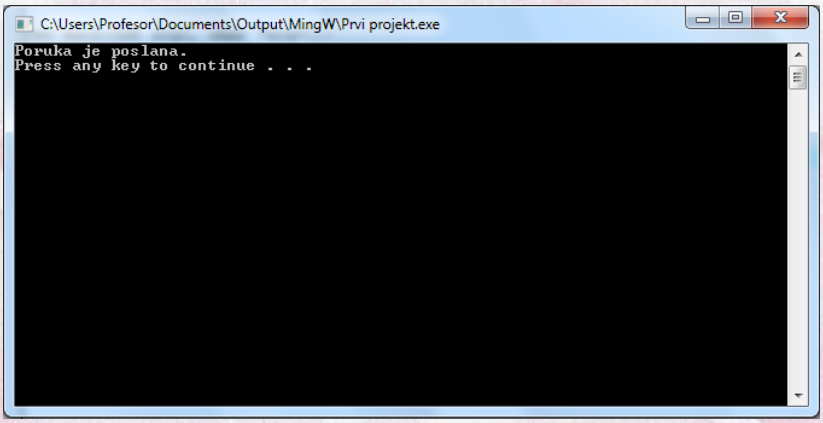
-**postupak izrade programa** može se podijeliti na tri dijela:

a) **pisanje izvornog koda** (engl. **source code**)
b) **prevođenje izvornog koda** (engl. **compiling**)
c) **povezivanje u izvršni kod** (engl. **linking**)

-mada postoje posebni programi za svaki od navedenih koraka izrade programa, danas se uglavnom rabe **integrirana razvojna okruženja** (engl. **Integrated Development Environment**, **IDE**)
-**IDE objedinjuju** programe za **pisanje izvornog koda, prevođenje, povezivanje, pohranu, izvršenje i pronalazak pogrešaka**
-zapisivanjem naputka za rješavanje zadatka naredbama programskog jezika (**kodiranjem**) nastaje **datoteka izvornog programa** (engl. **source code**)
-izvorni je kod moguće pisati u bilo kojem **programu za uređivanje teksta** (engl. **text editor**), ali je za tu svrhu puno **prikladnije** koristiti **uređivač teksta** koji je dio **IDE-a**, jer on **prepoznaje pojedine ključne riječi i druge dijelove sintakse jezika C++**
-takav uređivač **prepoznate dijelove boji različitim bojama i/ili ih uvlači** da bi se **lakše mogli snaći u napisanom programu**
-**datoteka izvornog koda** ima nastavak ***.cpp**
-**program prevoditelj** (**kompajler**, engl. **compiler**) **prevodi izvorni kod iz simboličkog jezika visoke razine** u tzv. **objektni kod** (engl. **object code**) te **provjerava sintaksu** (**pravila pisanja i upotrebe naredbi**) napisanog izvornog koda
-**objektni kod** je **međukorak do strojnog jezika**, jer u njemu **nedostaju veze** potrebne za stvaranje završne verzije programa (npr. nema ubačenih funkcija koje smo koristili, npr. za računanje kvadratnog korijena (***sqrt()***)
-**prevođenjem nastaje datoteka objektnog koda** s nastavkom ***.obj**
-ako kompajler pronađe **sintaktičke pogreške** (**pogrešno napisane naredbe** - engl. **syntax error**), **ispisuje poruke i upozorenja o njima**
-obično se navede **broj linije** u kojoj je došlo do pogreške i **vrsta pogreške**
-to može biti **vrlo korisno** programeru, ali te poruke dosta puta znaju navesti **krivo mjesto pogreške** pa treba dosta iskustva da se nađe stvarni uzrok pogreške
-kompajler može otkriti **samo dio pogrešaka**, prije svega **krivo napisane naredbe**, ali **ne može ispravljati logičke pogreške** (npr. pogrešku u algoritmu)
-**otkrivene pogreške** treba **ispraviti** pa **ponovo pokrenuti kompajler**, sve **dok više nema poruka o pogreškama**
-**datoteka objektnog koda nije izvršni** (engl. **executable**) **program** i ne može se izravno izvršiti na računalu
-**u izvršni je oblik pretvara tzv. program povezivač** (engl. **linker**)
-on **povezuje objektnu datoteku s knjižnicama** (**bibliotekama** – engl. **library**) i **drugim potrebnim datotekama** u kojima su već **unaprijed isprogramirane funkcije koje koristimo u programu**
-dakle, **biblioteke** su datoteke koje **sadrže gotove** (**kompajlirane**) **funkcije** koje se **umetnu na traženo mjesto u programu**

-pod pojmom **funkcije** podrazumijeva se **dio programa koji obavlja točno utvrđeni zadatak**, a napravljen je kao **zasebna cjelina** koja se u programu prepoznaje po svojem **imenu** (npr. funkcija abs() određuje apsolutnu vrijednost nekog broja)
-treba naglasiti da postoji **dosta gotovih funkcija** koje su raspoređene u **veći broj biblioteka**, a svaki korisnik može definirati i **vlastite funkcije** i po potrebi ih staviti u neku **vlastitu biblioteku funkcija**
-**veći dio funkcija** se **nalazi u jednoj velikoj datoteci** (**standardna biblioteka funkcija**), ali se **prilikom povezivanja u izvršnu datoteku povezuje samo dio kompletne biblioteke**
-inače bismo dobili **vrlo velike datoteke** koje bi se **duže pokretale i sporije izvršavale**
-zbog toga se **funkcije grupiraju u datoteke po sličnosti upotrebe** (npr. matematičke), a u programu mi moramo **znati** u koju dio **biblioteke** spada **funkcija** koju smo upotrijebili i **taj dio biblioteke** na početku programa **navesti** da bi se kod povezivanja ta datoteka mogla **uključiti**
-ako se **pri povezivanju pojavi pogreška** (engl. **link-time error**), o tome će se **ispisati poruka** (npr. upotrijebili smo funkciju za koju nismo naveli u kojem se dijelu biblioteke nalazi pa nije mogla biti ubačena u izvršnu verziju programa)
-**pogrešku** treba **ispraviti** pa **ponovno** pokrenuti **prevođenje i povezivanje**
-**rezultat uspješnog povezivanja** je **izvršna datoteka** (***.exe**)
-**izvršnoj datoteci** nisu potrebni nikakvi **dodaci** pa se može izvršavati i bez izvornog programa, kompajlera, povezivača, biblioteka itd.
-tijekom rada se osim sintaktičkih pogrešaka i pogrešaka povezivanja mogu javiti i **logičke pogreške**
-za otkrivanje **logičkih pogrešaka** (engl. **run-time error**) potrebno je **provjeriti program s podacima za koje je unaprijed poznat krajnji rezultat**
-**ispravljanje pogrešaka** nastalih u ovoj fazi je **najteže**
-kod **traženja i ispravljanja tih pogrešaka** vrlo je **koristan** **program za ispravljanje pogrešaka** (engl. **debugger**) koji omogućuje **pokretanje samo željenog dijela programa i/ili izvršavanje naredbi liniju po liniju**
-osim pogrešaka, kompajler i povezivač mogu javiti i **upozorenja** (engl. **warnings**)
-upozorenja **ne sprečavaju prevođenje, povezivanje** (**kompajliranje**) i **izvršavanje programa**, već ukazuju na dijelove programa koji bi **pod određenim okolnostima** mogli **prouzročiti pogrešku** pa je stoga najbolje ukloniti njihove uzroke
-na idućim slikama prikazan je izgled prozora s **izvornom datotekom i porukom o uspješnom kompajliranju i izgled programa koji je pokrenut**
-zatim su namjerno napravljene **pogreške** u izvornom programu (nedostaje ;) da bi se vidjelo **poruke o pogrešci**





[*] Probni program.cpp

```
#include <cstdlib>
#include <iostream>

using namespace std;

int main(int argc, char *argv[])
{
    cout<<"Poruka je poslana."<<endl
    system("PAUSE");
    return EXIT_SUCCESS;
}
```

Probni program.cpp

```
#include <cstdlib>
#include <iostream>

using namespace std;

int main(int argc, char *argv[])
{
    cout<<"Poruka je poslana."<<endl
    system("PAUSE");
    return EXIT_SUCCESS;
}
```

Greške

Greške | Resursi | Ispis kompajlera | Debug | Rezultati traženja | To-Do List

Linija	Lokacija	Poruka
	C:\Users\Profesor\Documents\Pr...	In function `int main(int, char**)':
9	C:\Users\Profesor\Documents\Pr...	expected `;' before "system"
	C:\Users\Profesor\Documents\M...	[Build Error] [Objects/MingW/Probni program.o] Error 1

-da bi se stvorila izvršna datoteka potrebno je pokrenuti **nekoliko programa**, a svaki od njih stvara **nekoliko datoteka**

-programeru **ne bi bilo lako zapamtiti** koje sve radnje treba pokrenuti za stvaranje izvršne datoteke

-stoga mu je posao znatno **olakšan objedinjavanjem svih datoteka** vezanih za jedan program u **projekt** (engl. **project**)

-**projekt** je **skup međusobno povezanih datoteka**

-projekt "brine" o svemu što je potrebno učiniti da bi **od izvornog kôda nastala datoteka izvršnog kôda**

-**projekt** se **stvara** aktiviranjem naredbe **Datoteka->Nova->Projekt** (simbol -> označava da se radi o naredbama u meniju)
-postupak stvaranja **novog projekta** (nazvanog Prvi projekt) prikazan je idućom slikom

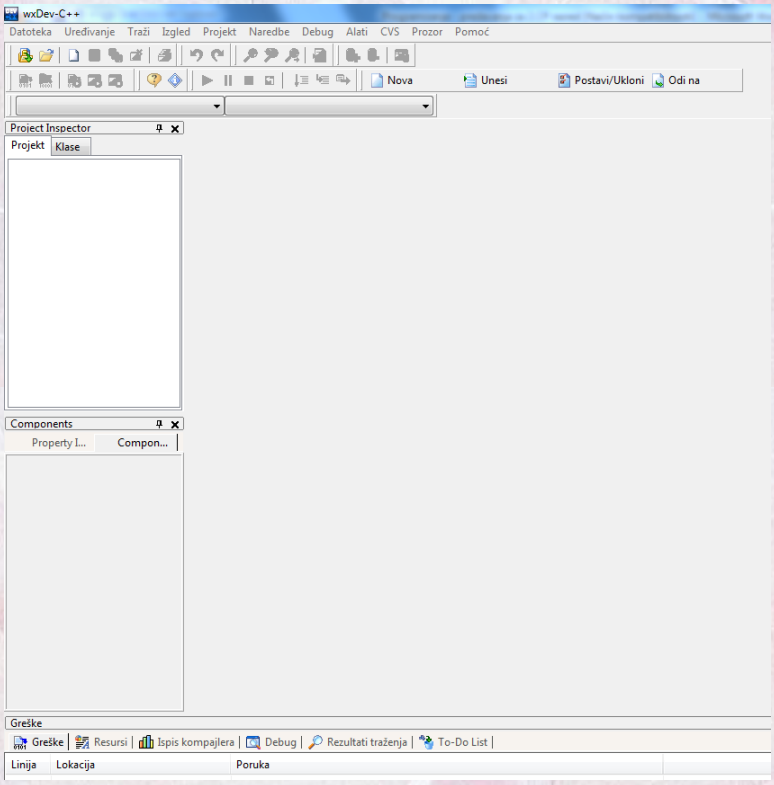
-na toj slici se vidi da **osim biranja imena projekta biramo i vrstu izvršne datoteke**
-mi ćemo se služiti izborom tzv. **Console Application projekta** koji kao rezultat daje **crni prozor s rezultatom rada programa** (**DOS prozor** ili **prozor programa Command Prompt**)
-postojeći projekt možemo **otvoriti** (naredba **Datoteka->Otvori projekt ili datoteku), spremiti (naredba Datoteka->Spremi sve), zatvoriti (Datoteka->Zatvori), stvoriti izvršnu verziju (Naredbe->Kompajlaj sve) i pokrenuti je (Naredbe->Pokreni**)
-povrh toga može se napraviti **dosta drugih radnji** koje nam za sada nisu potrebne
-**osnovni prozor IDE-a** sastoji se od tri cjeline:

a) **glavnog prozora**
-u njemu **pišemo** program

b) **okvira alata**
-tu **biramo različite alate** koji pomažu pri izradi i prevođenju programa (npr. alat za kompajliranje, za traženje pogrešaka, povezivanje itd.)

c) **okvira poruka**
-u njemu se **prikazuju različite poruke** koje govore npr. o tijeku prevođenja ili povezivanja datoteke, o pogrešakama i upozorenjim itd.

-iduća slika prikazuje **izgled cijelog prozora IDE-a wxDev-C++**



2.2. Varijable, konstante i tipovi podataka

-sve **podatke** u programu dijelimo u dvije osnovne grupe:

a) **varijable** (engl. **variable**)
-vrijednost im se u programu **može mijenjati**

b) **konstante** (engl. **constant**)
-vrijednost u programu im se **ne može mijenjati**

2.2.1. Varijable

-da bi se obavila bilo kakva izračunavanja u programu ili bilo koja operacija koja daje neki rezultat, trebamo moći **zapamtiti** jednu ili više vrijednosti tijekom izvršavanja programa
-to nam omogućuju mjesta u **radnoj memoriji** (RAM) nazvana **varijable**
-svaka varijabla ima **jedinstvenu memorijsku adresu** (engl. **memory address**), tj. **ne postoje dvije varijable s istom adresom**
-za računalo je **memorijska adresa varijable** (mjesto u memoriji u koju je upisan sadržaj varijable) zadana u obliku **dugog binarnog broja** (npr. 10100000010101001010101111000 01)
-korisnicima (programerima) je takav način zadavanja varijabli **neprikladan** pa se radi lakšeg pamćenja uvodi označavanje varijabli **simboličkim imenima** (engl. **symbolic name**), tj. pomoću njezina naziva ili kraće - pomoću **imena varijable**
-zbog toga računalo na osnovu imena varijable može pronaći gdje se točno u memoriji nalazi sadržaj neke varijable
-ime varijable često se puta naziva i **identifikatorom** (engl. **identificator**)
-kod **izbora imena varijable** moramo poštovati ova **pravila**:

a) smiju se rabiti **velika** (A-Z) i **mala** (a-z) **slova engleske abecede**, **znamenke** (0-9) i **znak** _ (**podcrtavanje** - engl. **underscore**)
b) ime mora **početi slovom ili znakom potcrtavanja** (_), tj. **ne smije početi znamenkom**
-izravna posljedica ovog zahtjeva je da **ne može broj služiti kao ime varijable** (npr. *14532* ne može biti ime varijable, jer je prvi znak znamenka)

-**pravila za određivanje simboličkog imena**:

a) **ne smije** se rabiti **razmak** kao dio imena varijable (u tom slučaju kompajler smatra da se radi o dva imena te će sigurno javiti **poruku o pogrešci**)
-primjer:
parno (dobro), *nije parno* (pogrešno)
b) budući da je programski jezik C++ razvijen u engleskom govornom području, **ne smiju se rabiti naši dijakritički znakovi** (č, Č, ć, Ć, ž, Ž, š, Š, đ, Đ)
-primjer:
pozar (dobro), *požar* (pogrešno)
c) **ne smiju** se rabiti **ključne riječi** (npr. goto) ili **oznake operatora** kao imena varijabli (npr. +)
-dobra je praksa da za imena varijabli koristimo **hrvatska imena** (**bez dijakritičkih znakova**, tj. s umjesto Š, c umjesto Č itd.) pa onda nećemo doći u opasnost da upotrijebimo za ime varijable neku ključnu riječ
-od svih ključnih riječi programskog jezika C++ samo riječi **auto**, **do** i **operator** koriste se u hrvatskom jeziku te njih **ne smijemo koristiti**
-sve druge riječi našeg jezika su nam na raspolaganju za izbor imena varijable
-**ključna riječ može biti bilo koji dio imena varijable**, ali sama ključna riječ ne može biti jedini dio imena varijable
-primjer:
do (pogrešno), *do_sada* (dobro), *auto* (pogrešno), *auto1* (dobro), *operator* (pogrešno), *operator1* (dobro)

d) program **razlikuje velika i mala slova** (engl. **case sensitive**) za razliku od nekih drugih programskih jezika (npr. QBasic) gdje se ista velika i mala slova smatraju istim znakovima (engl. **case insensitive**)
-primjer:
brojač (pogrešno), *brojac* (dobro), *Brojac* (dobro), *BROjac* (dobro), *BROJAC* (dobro), *brojaC* (dobro)

e) **broj znakova u imenu** (**dužina**) **nije ograničen**, ali će pojedini kompajler **u obzir uzimati samo određeni broj prvih znakova** (npr. 14), a **ostale će zanemariti**
-primjer za kompajler koji razlikuje imena duljine do 14 znakova:
brojac_okretaja_motora, brojac_okretaja, brojac_okretaja_motora1
-sva imena varijabli u prijašnjem primeru smaraju se istima, jer im je prvih 14 znakova isto, a ostali znakovi se zanemaruju kod kompajliranja
-kod programiranja trebamo paziti da **ne koristimo preduga opisna imena** (treba duže vrijeme za otipkati takvo ime varijable), ali puno je **bolje imati duža imena varijabli** (unutar broja znakova koje kompajler smatra različitima), nego ih **skratiti tako da ne znamo koje je značenje varijable** u programu
-svako **ime varijable treba čim bolje upućivati na njezinu upotrebu**
-primjer:
brojac1, *brojac1_za_USA* (nepotrebno dugo ime), *s23dfdewew* (ime ne označava upotrebu varijable)
-obično se kod **brojanja ponavljanja u petljama** i kod **pamćenja nebitnih međurezultata** koristimo **imenima varijable duljine jednog znaka**
-primjer:
a, b, i, j, k, l, m, n,...

f) ako se koristi **ime sastavljeno od više riječi**, one se mogu **odvojiti znakom za podcrtavanje** ili **pisati spojeno s velikim početnim slovom za svaku riječ** da bi se **lakše uočila upotreba varijable**
-primjer:
broj_1_start, Broj1Start

-primjeri **ispravnih** imena varijabli:
x
promjer_kruga
_kontrola1
DatumUpisa
Valuta23MK_X

-primjeri **neispravnih** imena varijabli:

y[1], x(2)	(ne smije sadržavati zagrade)
Datum Upisa	(ne smije sadržavati razmak)
auto	(ne smije biti ključna riječ)
Brojač1	(ne smije sadržavati naše dijakritičke znakove)

-postoje **različite vrste podataka**, npr. cijeli brojevi, realni brojevi, logički podaci, znakovi, nizovi znakova itd.
-svakoj varijabli osim imena trebamo dodijeliti i **oznaku tipa podatka** koji će u nju biti smješten
-to je potrebno zato da računalo zna **koliko mjesta u memoriji treba predvidjeti za pohranu zadanog podatka**

-**uvođenjem tipova** podataka programer treba uložiti **dodatan napor**, ali time se **dobiva**:

a) **bolje iskorištenje memorije**
b) **kraće programe**
c) **brže programe**, jer se **operacije nad svim tipovima podataka ne obavljaju jednako brzo**

-postupak **pridjeljivanja simboličkog imena varijabli i određivanje tipa podatka** naziva se **deklariranje** ili **najava vrijednosti** (engl. **declaration**)
-**deklaracija** se piše u obliku:

oznaka tipa podatka simboličko ime podatka;

-primjer:

int a;
float mjera;

-ukoliko imamo **više varijabli istog tipa**, možemo **samo jednom napisati oznaku tipa varijable**, a nakon toga **popis varijabli odvojenih zarezom**
-primjer:

int a, b, broj, ostatak;

-**u istoj naredbi možemo deklarirati više varijabli različitih tipova**
-primjer:

int a, b, c, float i, j, k;

-značenje oznaka tipa int i float objašnjeno je u nastavku
-**deklariranoj varijabli** se može **pridružiti vrijednost operatorom pridruživanja** (znak **jednako**, =)
-ukoliko se **zadaje početna vrijednost** varijable tada to nazivamo **inicijalizacijom** (engl. **intialization**)
-**inicijalizacija se može provesti istovremeno s deklaracijom, ali i ne mora**
-primjer:

int a=3; //deklarirali smo varijablu a i u tijeku inicijalizacije pridružili joj vrijednost 3

-ukoliko se u programu koriste **neinicijalizirane varijable**, trebate znati kako ih tretira vaš **kompajler**
-obično ih kompajleri smatraju **jednakim 0**, ali to **ne mora biti uvijek tako**
-zbog toga je preporuka da se **svim varijablama zada početna vrijednost**
-kao što smo vidjeli prije, znak = više ne označava izjednačavanje (jednakost) kao u matematici
-njegovo značenje je da **objektu s lijeve strane operatora pridruživanja pridružuje vrijednost s njegove desne strane**
-**objekti s lijeve strane operatora pridruživanja moraju biti varijable**
-primjer:

a=a+3; //vrijednost varijable a uvećaj za 3

-u prijašnjem primjeru vidi se da se podatku koji se nalazi u varijabli a dodaje vrijednost 3 i zatim taj zbroj pohrani u varijablu a kao nova vrijednost
-**u istoj se naredbi** može koristiti i **više operatora pridruživanja radi kraćeg zapisa** (**smjer operacije ide s desna na lijevo**)
primjer:

a=b=c=5; //c poprimi vrijednost 5, varijabla b poprimi vrijednost varijable c, varijabla a poprimi
//istu vrijednost koju ima varijabla b

-**podaci** se mogu **podijeliti** u:
a) **osnovne tipove**
b) **ostale tipove**
-nas za sada zanimaju samo **osnovni tipovi**
-**osnovni tipovi podataka** su:
a) **brojevi**
b) **znakovi**
c) **logički tip**

2.2.1.1. Brojevi (numerički podaci)

-C++ razlikuje **dvije osnovne vrste brojeva**

-to su:
a) **cijeli brojevi** (engl. **integer**)
b) **realni brojevi** (engl. **floating point**)

2.2.1.1.1. Cjelobrojne varijable

-ako je podatak **cijeli broj**, njegova **oznaka tipa** je **int**
-dakle, varijabla označena s **int** je **cjelobrojna**
-**cjelobrojnoj varijabli** može se **pridijeliti samo cijeli broj**
-za pohranu cijelog broja u memoriji su predviđena **4 bajta** (32 bita)
-prvi bit je rezerviran za predznak (+ ili -), pa za pohranu broja ostaje 31 bit
-time se omogućava pohranu brojeva iz raspona:

$[-2^{31}, 2^{31}-1]$ to jest od -2.147.483.648 do 2.147.483.647 (**približno od -2 do +2 milijarde**)

-sve cjelobrojne varijable mogu biti **deklarirane s ili bez predznaka**
-ako se deklarira **cijeli broj bez predznaka** potrebno je **ispred oznake tipa staviti ključnu riječ unsigned** (**nepredznačeni tip**, tj, **samo sa pozitivnim vrijednostima**)
-primjer:

```
unsigned int brojac=100;
```

-u slučaju cijelog broja bez predznaka bit za predznak više nije potreban
-najveću vrijednost sada je moguće prikazati sa 32 bita pa je raspon vrijednosti ovog tipa od 0 do 4294967295, tj. **približno od 0 do 4 milijarde**
-želimo li koristiti **još veće cijele brojeve**, **ispred oznake tipa int** dodajemo riječ **long** (**dugo**)
-primjer:

```
long int broj_atoma;
```

-često želimo radi **bržeg računanja** koristiti **manje cijele brojeve** nego što nam to omogućuje tip int (npr. kod **brojanja ponavljanja u petljama**)
-tada **ispred oznake tipa int** dodajemo riječ **short** (**kratko**)
-taj tip podataka troši **dva bajta** u memoriji (16 bitova), a njime se prikazuju vrijednosti od -**32786 do 32767**
-oznaku **unsigned** možemo koristiti i **u kombinaciji s oznakama long i short**, pri čemu za **short** tip dobivamo opseg vrijednosto **od 0 do 65535**
-primjer:

```
short int brojac;
unsigned short int brojilo;
unsigned long int masa_Zemlje;
```

2.2.1.1.2. Realne varijable

-ako je podatak **realni broj** njegova **oznaka tipa** je **float**
-**realni brojevi** mogu se **prikazati**:
a) **s nepomičnom decimalnom točkom**
b) **s pomičnom decimalnom točkom** (engl. **floating point**), tj. u **eksponencijalnom prikazu**
-u C++ za odjeljivanje cjelobrojnog od decimalnog dijela broja rabimo **decimalnu točku**, a **ne zarez**
-kada se realne brojeve prikazuje u **eksponencijalnom prikazu** (**s pomičnom decimalnom točkom**), oni su **oblika**:

$$\mathbf{M \cdot 10^E}$$

-tu **M** označava dio broja koji se naziva **mantisa**, a **E** je **eksponent baze 10** (**10 na E-tu potenciju**)
-**mantisa** se zapisuje tako da je **prva znamenka različita od nule lijevo od decimalne točke**
-primjeri zapisa mantise:

6.345, 1236.345, 0.000765

-realni brojevi u prijašnjim primjerima mogu se zapisati kao:

$6.345 = 6.345 \cdot 10^0$ = 6.345e0 //zadnji i prvi zapis su u C++ jeziku

$1236.345 = 1.236345 \cdot 10^3$ = 1.236345E+3 //zadnji i prvi zapis su u C++ jeziku

$0.000765 = 7.65 \cdot 10^{-4}$ = 7.65e-4 //zadnji i prvi zapis su u C++ jeziku

-za pohranu **realnog broja** u memoriji predviđena su **4 bajta** (32 bita)

-time je omogućena pohrana brojeva u rasponu od **±3.4*10^{38} do ±1.17*10^{-38}**
-u **običnu** (**float**) **realnu varijablu** sprema se **samo 7 decimalnih znamenki mantise**
-ako se **unese više od sedam znamenki**, prilikom **prevođenja** će biti **zanemarene najmanje vrijedne decimalne znamenke** (po potrebi se broj **zaokružuje**)
-primjer:

float a=1.23344568909001; (broj koji se pamti je 1.233446; zadnja znamenka dobivena je zaokruživanjem)

-uobičajeno se **realni brojevi prikazuju s do 6 znamenaka**, **računajući od prve različite od 0**
-primjer:

0.2334, 0.2, 1.34565

-ako se broj ne može prikazati s toliko znamenaka, bit će **prikazan u eksponencijalnom prikazu**
-primjer:

123.4567 prikazuje se kao 1.234567e+2

-za oznaku **eksponenta** možemo proizvoljno koristiti **e** ili **E**
-predznak + iza **e** (**E**) **nije potrebno** pisati, ali predznak - **moramo pisati**
-primjer:

-2.345E+3 je isto što i -2.345e3, ali nije isto što i -2.345E-3

-ako nas navedena **točnost ne zadovoljava** (to se rijetko zbiva) ili ako se žele koristiti brojevi manji od 10^{-38} ili veći od 10^{38}, mogu se rabiti **varijable veće točnosti**
-to su **varijable tipa**:
a) **double** (eksponent do ±308), s točnošću do **15** decimalnih znamenki mantise
b) **long double** (eksponent do ±4932), s točnošću do **18** decimalnih znamenki mantise
-primjer:

double atom=1.62343223233e-29;
long double broj_atoma=1.243432432423423E35;

2.2.1.2. Znakovi

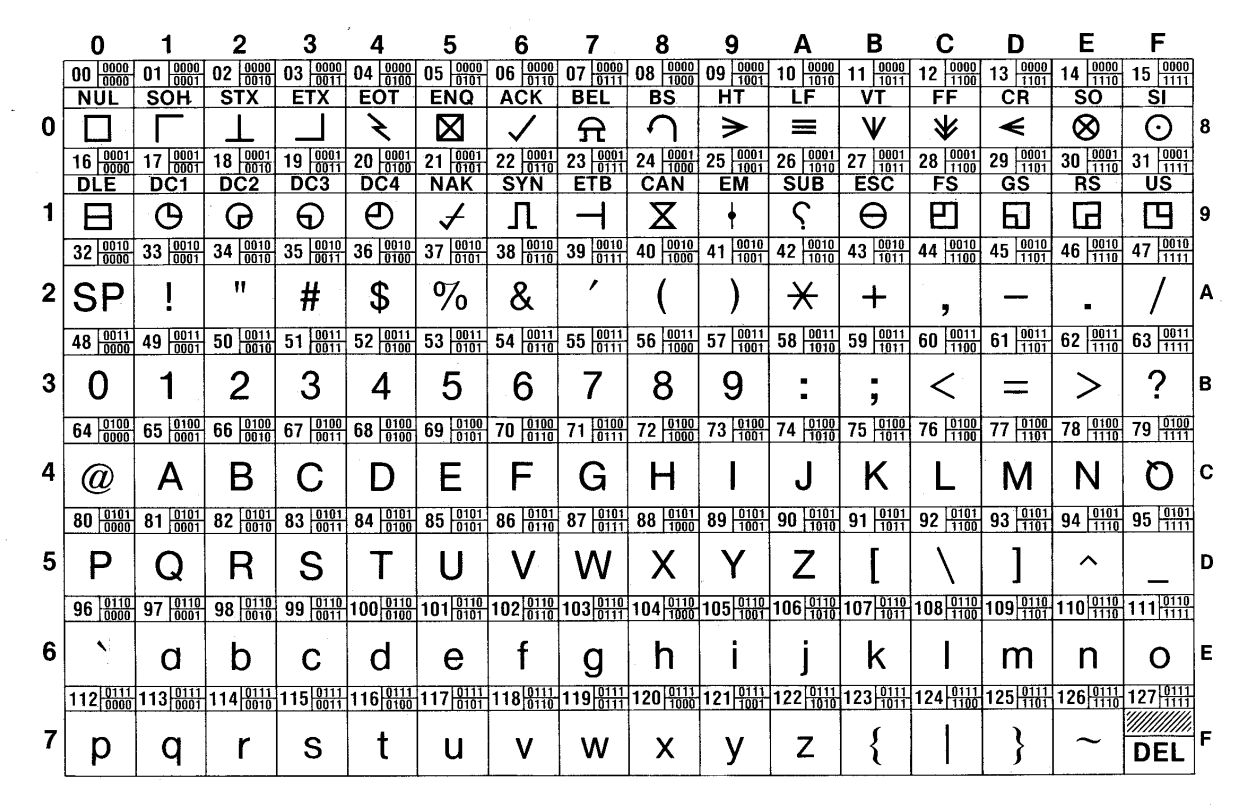
-ako je podatak **znak**, njegova oznaka tipa je **char** (skraćeno od engl. **character** = **znak**)
-podatak tipa **char prikazuje** se:
a) **jednim znakom unutar jednostrukih navodnika**
-primjer:

char znak='A';

b) **ASCII vrijednošću tog znaka** (u **dekadskom** obliku)
-primjer:

char znak=65; //to je slovo A kao i u prikazu pomoću jednostrukih navodnika

-za pohranu znakovnog podatka je u memoriji predviđen **1 bajt** (8 bitova)
-pošto je 2^{8} = 256, moguće je prikazati **256 različitih znakova**
-znak se **pohranjuje** kao broj koji predstavlja **ASCII vrijednost odabranog znaka**
-u nastavku je prikazana skraćena tablica ASCII kodova (128 znakova)



-iz nje se vidi da su u tablici prvo **nevidljivi znakovi** (npr. 7 označava zvučni signal), potom od 48 do 57 slijede **znamenke 0 do 9**, zatim od 65 do 90 su **velika slova A do Z** i na kraju od 97 do 122 **mala slova a do z**

-ako je podatak **znak koji se ne može prikazati na zaslonu** (znakovi iz ASCII tablice od 0 do31), koristi se **slijed koji počinje lijevom kosom crtom** (engl. **backslash**) nakon koje **slijedi neko od unaprijed određenih slova ili određeni znak** (npr. a, n, ? ")

-primjer:

char znak=\a; //znak za zvučni signal

-za **pohranu teksta dužeg od jednog znaka** se koriste **znakovni nizovi** (engl. **character strings**)

-za sada je dovoljno znati da se **sadržaj znakovnog niza navodi unutar para dvostrukih navodnika**

-primjer:

"Ovo je znakovni niz"

2.2.1.3. Logički tip

-varijable **logičkog tipa** mogu poprimit samo dvije vrijednosti: **istina** ili **laž**

-u C++ se koristi logički tip koji se označava s **bool**

-**istina** se označava s **true**, a **laž** s **false**

-primjer:

bool a=true, b=false;

-vrlo često se **logičke vrijednosti prikazuju cijelim brojevima**, pri čemu broj **0** označava **false**, a **bilo koji drugi broj true**

-primjer:

bool a=123, b=0, c=-23220; //a i c su true, b je false

2.2.1. Konstante

-u programima se ponekad rabe **simboličke veličine čija se vrijednost tijekom izvođenja programa ne smije mijenjati**

-takve se simboličke veličine nazivaju **konstantama** (npr. fizikalne ili matematičke konstante)

-najčešće koristimo **brojevne konstante**, mada možemo i **znakovne**

-**brojevne konstante prevoditelj pohranjuje** u obliku nekog od **osnovnih brojevnih tipova podataka**

-tako:
a) **realne brojevne konstante postaju tipa double**
b) **cjelobrojne brojevne konstante postaju tipa int**
-ako se **u programu pokuša promijeniti vrijednost konstante**, **prilikom prevođenja će prevoditelj javiti pogrešku**
-**konstante se često koriste**, jer se **programi pomoću njih lako modificiraju** (promijeni se samo na jednom mjestu vrijednost konstante, a ne mijenjamo vrijednosti brojeva na većem broju mjesta u programu)
-**konstante** se obično navode **na početku programa**, a one se **zadaju pomoću ključne riječi const**, **imena i vrijednosti konstante**
-primjer:

```
const pi=3.14159;
```

2.3. Osnovna struktura programa

-da bi znali pisati programe na čim bolji način, bilo bi dobro analizirati **strukturu tipičnog jednostavnog programa**
-idući program prikazuje **osnovnu strukturu** jednog programa:

```
#include <cstdlib>
#include <iostream>

using namespace std;

int main(int argc, char *argv[])
{
  int a, b , c, float d; //deklaracija varijabli
  //računamo formulu d=(a·b)/c+3.33
  a=13;
  b=234;
  c=11;
  d=a*b;
  d=d/c;
  d=d+3.33;
  cout <<"Rezultat je "<<d<<"."<<endl;
  system("PAUSE")
  return EXIT_SUCCESS; //može se koristiti i naredbu return 0;
}
```

-u prijašnjem programu možemo uočiti slijedeće dijelove:
a) **naredbe za uključivanje funkcija iz biblioteka u program**
-ove naredbe služe da **umjesto imena funkcija** u fazi **povezivanja ubace veze na stvarne naredbe** koje čine traženu funkciju
-primjerice, ako se radi o funkciji *abs(x)*, kod za nju bi mogao glasiti:

```
if (x<0)
  {
    x=x*(-1);
  }
```

-u prijašnjem primjeru bi se u cijelom programu svaka upotreba funkcije abs(x) zamijenila vezama na naredbe koje su navedene u prijašnjem primjeru
-pri **kompajliranju** u program se ubacuje **dio cijele standardne biblioteke funkcija** (taj dio se pritom **kompajlira**), a pri **povezivanju** se **kreiraju veze** našeg programa na dio kompajliranog koda iz biblioteke
-većinom koristimo gotove funkcije iz **standardne biblioteke funkcija**, mada možemo napraviti i **vlastite biblioteke** i uključiti i funkcije iz njih

-koncentrirat ćemo se na uobičajeni slučaj, a to je upotreba funkcija iz **standardne biblioteke funkcija**

-za **uključivanje** koristimo **naredbu #include** iza koje slijedi **ime** tzv. **datoteke zaglavlja** (engl. **header file**) u kojoj se **nalaze deklaracije** (**najave**) **funkcija** iz standardne biblioteke funkcija

-**datoteka zaglavlja** definira **koji dio standardne biblioteke funkcija ubacujemo u naš program**

-u standardnoj biblioteci funkcija postoje **32 različita zaglavlja** (npr. complex (rad s kompleksnim brojevima), string (rad s tekstom), itd.)

-**naziv datoteke zaglavlja** piše se između znakova < i >, a pritom se unutar njih **ne smiju pisati razmaci**

-mi ćemo najčešće koristiti funkcije iz datoteke zaglavlja **cstdlib** (**standardne funkcije** koje su identične i u C jeziku) i one iz datoteke zaglavlja **iostream** (funkcije za **ispis i unos podataka** iz/u računalo - obično je **standardni ulaz tipkovnica**, a **standardni izlaz ekran monitora**)

b) **naredba za izbor standardnih imena funkcija iz biblioteka**

-naredbu ***using namespace std;*** upotrebljavamo ukoliko bi se dogodilo da neka **funkcija iz standardne i neke druge biblioteke imaju isto ime**, a obavljaju **različite zadaće**

-upotrebom ove naredbe uvijek se u navedenom slučaju **bira funkcija iz standardne biblioteke funkcija**

c) **glavna funkcija**

-naredba ***int main(int** argc, **char** *argv[]) {}* **glavna je funkcija** programa i njezino **ime** (**main**, engl. glavna) **ne smije se koristiti** kao ime **neke druge funkcije** (drugim riječima, **u programu mora biti samo jedna main funkcija**)

-za sada je bitno reći da **u njoj pišemo kompletni program**

-dio naveden **u zagradi main funkcije** (**int** argc, **char** *argv[]) služi za to da **operativnom sustavu** (npr. Windows 7) **nakon izvršenja programa pošalje status** (stanje) **programa** (da li je program **uspješno završio** ili je došlo do neke **greške**)

-naše **naredbe u main funkciji pišemo unutar vitičastih zagrada koje su obavezne**

d) **deklaracija varijabli**

-uobičajeno je da **deklaraciju** svih varijabli napravimo **na početku main funkcije** (radi **preglednosti**), mada možemo i **bilo gdje u programu**, ali svakako **prije upotrebe promatranih varijabli** (inače kompajler javlja **pogrešku**)

e) **inicijalizacija varijabli**

-dobra je praksa **na početku main funkcije napraviti inicijalizaciju varijabli**, jer **o kompajleru ovisi** koje će biti **početne vrijednosti** varijabli (obično 0, ali može se raditi o prijašnjem sadržaju neke memorijske lokacije)

f) **komentari**

-komentari nam **olakšavaju analizu programa**, jer se njima opisuje njegovo **funkcioniranje**

-često se komentarima opisuje **namjena programa**, **uloga pojedinih varijabli** i **bitni dijelovi algoritma**

-s komentarima **ne treba pretjerati**, jer ćemo **izgubiti preglednost**

-**kompajler izbacuje sve komentare** (kao **i razmake i prelaske u novi red**) pri kompajliranju i oni **ne povećavaju duljinu programa** pa ih možemo **koristiti po volji često**, ali **ne smijemo izgubiti preglednost**

-komentarima se može **privremeno izbaciti neki dio programa** da se vidi **ponašanje programa bez toga dijela**

-obično se radi o **dijelu programa koji uzrokuje pogreške kod kompajliranja** pa izbacivanjem pojednih dijelova programa nastojimo **izbjeći poruke o pogrešci** kako bismo otkrili **problematičan dio** programa

-**dvije su vrste komentara**:

a) **komentar u liniji** (engl. **inline comment**)

-ovaj komentar piše se s **dva uzastopna znaka dijeljenja** (//) i sve **od ta dva znaka do kraja reda postaje komentarom** i kompajler to **ignorira**

-takvi komentari obično se koriste za **opis programske linije** lijevo od komentara (**komentar je na kraju reda**) ili za **izbacivanje cijele linije koda kod traženja pogrešaka**
-primjeri:
a=c+d;//tu zbrajamo - ovaj komentar opisuje naredbu lijevo od njega u istom redu
//a=c+d; - tu smo komentarom izbacili cijelu programsku liniju iz kompajliranja

b) **blok komentar** (engl. **block comment**)
-njime se obično **opisuje namjena programa i uloga varijabli** ili se **izbacuje nekoliko uzastopnih linija koda iz kompajliranja**
-takav komentar **počinje** kombinacijom znakova **/***, a **završava** kombinacijom ***/**
-primjer:
/ int a;*
a=1;
*cout<<a; */*
-ovakvim komentarom **sve navedene naredbe su izbačene** kod postupka kompajliranja
-kod komentiranja se **promijeni boja naredbi** koje su **izbačene komentiranjem** pa tako lakše pratimo ostatak programa

g) **naredbe programa za realizaciju traženog algoritma**
-to su **bilo koje naredbe koje su potrebne za realizaciju našeg algoritma**

h) **ispis/unos vrijednosti iz/u program**
-u tu svrhu **obično se koristimo naredbama cin za unos podataka i cout za ispis podatka na ekranu**

i) **zaustavljanje programa do pritiska na neku tipku**
-naredba ***system("PAUSE")*** je korisna, jer **sprječava** da se crni prozor nastao izvršavanjem programa **odmah nakon završetka programa zatvori**
-zbog toga možemo po volji dugo **gledati rezultat rada programa**, a onda ga (kao što i kaže poruka **press any key**) **zatvoriti pritiskom na bilo koju tipku**

j) **naredba za vraćanje statusa programa nakon izvršenja**
-dio naveden u zagradi main funkcije (***int** argc, **char** *argv[]*) skupa s naredbom **return** EXIT_SUCCESS; (može se koristiti i naredbu **return** 0;) služi za to da **operativnom sustavu** (npr. Windows 7) **nakon izvršenja programa pošalje status** (stanje) **programa** (da li je program **uspješno završio** ili je došlo do neke **greške**)

-vidi se da su **različiti dijelovi izvornog koda radi lakšeg i bržeg snalaženja obojani različitim bojama** (npr. uključivanje zaglavnih datoteka je zeleno, konstante crveno, komentari plavo, dok su **ključne riječi podebljane**)
-primjerice, ukoliko **krivo napišemo ključnu riječ**, ona **neće biti podebljana** pa tako **lakše uočavamo da smo je krivo napisali**

2.4. Funkcije

-često se nađemo u situaciji da u programima imamo **nizove naredbi** koje se **ponavljaju** na više mjesta u programu
-time program postaje **nepreglednij**i i povećava se vjerojatnoća da ćemo prilikom unosa ponovljenih naredbi nešto **pogriješiti**
-dobra ideja bi bila da takvom nizu naredbi damo neko **ime** i potom ga pomoću toga imena **ubacimo** na odgovarajuća mjesta u programu
-upravo to nam omogućavaju tzv. **funkcije** (engl. **function**)
-možemo reći ovako: **funkcije** zamjenjuju **blokove naredbi**, a u program ih ubacujemo **navođenje**m njihova **imena**
-primjer:
ukoliko slijedeće naredbe nazovemo imenom ***racun***, navođenjem toga imena kopiramo te naredbe gdje želimo

```
a=3;
b=324;
a=a+b;
```

```
c=2*a;
d=c/23;
```

-funkciju možemo upotrijebiti i na način da se u njoj nešto **izračuna** i da koristimo tu **vrijednost** kod **naredbi pridruživanja**
-u tom slučaju u funkciji moramo birati koje **ulazne podatke** treba i koji **tip rezultata** nam **vraća**
-dakle, ako sumiramo prije izneseno, **funkcije po vraćanju izračunane vrijednosti** mogu biti:

a) **funkcije koje ne vraćaju vrijednost**
-**ne** mogu se upotrijebiti **za pridruživanje vrijednosti** (kao desna strana iza znaka pridruživanja (=))
-takve funkcije **samo zamjenjuju blok naredbi**
-za one koji su programirali u **Pascalu**, tamo se takve funkcije zovu **procedurama** (engl. **procedure**)

b) **funkcije koje vraćaju vrijednost**
-**mogu** se upotrijebiti kao **desna strana iza znaka pridruživanja** (=)
-primjer:

```
a=abs(b);
```

-u prijašnjem primjeru funkcija *abs()* izračuna apsolutnu vrijednost varijable *b*, a rezultat se pridružuje varijabli *a*

-kod **biranja imena funkcije** služimo se **istim pravilima** kao i kod izbora **imena varijabli ili konstanti**, pri čemu - pojednostavljeno kazano - to **ne smije** biti isto ime kao **ime bilo koje funkcije**, **varijable** ili **konstante** u programu
-**upotrebom funkcija dobijamo**:

a) **podjelu programa na manje dijelove** koje je **lakše koristiti**
-time program postaje **modularan**, tj. sastoji se od **dijelova** koji se mogu **slagati u cjelinu** poput **mozaika**

b) program koji je **čitljiviji**, tj. **lakše** se **prati rad** programa

c) **razumljivije ponašanje** programa, tj. **lakše** je vršiti **promjene** ili **ispravljati pogreške** u programu

-kod **upotrebe funkcija** razlikujemo **tri koraka**:

a) **deklaraciju funkcije** (engl. **function declaration**)

b) **definiciju funkcije** (engl. **function definition**)

c) **upotrebu funkcije**, tj. njezin **poziv** (engl. **function call**)

2.4.1. Deklaracija funkcije

-deklaracija funkcije služi **kompajleru** da **rezervira prostor u memoriji** za potrebne **varijable**
-deklaracija se mora **pisati izvan funkcije** *main()*, dakle **prije** ili **poslije** nje
-uobičajeno je da se **deklaracije svih funkcija** pišu **prije** funkcije *main()*, a **nakon** ostalih tzv. **pretprocesorskih naredbi** (*#include*, *using namespace*...)
-deklaracija se **piše** u ovom **obliku**:

```
tip_vraćene_vrijednosti ime_funkcije(tip argumenta1, tip argumenta2,...);
```

-objašnjenja prijašnjih **oznaka**:

1.) **tip_vraćene_vrijednosti**
-koristi se **samo** u slučajevima kada **funkcija vraća neku vrijednost**
-ukoliko **funkcija ne vraća vrijednost** (npr. funkcije za **unos i ispis podataka**), možemo to naznačiti upotrebom ključne riječi **void** ispred imena funkcije
-kada funkcija **vraća** neku **vrijednost**, navedemo **tip vraćene vrijednosti** (npr. int, float, unsigned short int,...)
-primjer deklaracije funkcije koja **ne vraća** vrijednost:

```
void ispis_kvadrata_broja(int x);
```

-primjer deklaracije funkcije koja **vraća** rezultat tipa **float**:

```
float kvadrat(float x);
```

2.) **ime_funkcije**

-vrijedi sve rečeno o **izboru imena varijabli**, s time da ime funkcije **ne smije biti isto** kao **ime neke varijable, konstante ili druge funkcije**

3.) **()**

-**unutar** ovih **zagrada** pišu se **tipovi argumenata** koje koristi **funkcija**

-zagrade možemo smatrati **dijelom imena funkcije**

4.) **tip_argumenta1, tip argumenta2,...**

-u zagradama se zadaju **tipovi i imena varijabli** koje koristimo u funkciji

-**broj** tih **varijabli** i njihov **tip** je **proizvoljan**

-u stvari, pošto ta **imena varijabli vrijede samo u funkciji** za potrebe **rezerviranja prostora u memoriji**, možemo koristiti **bilo koja dozvoljena** (čak **i ako postoje u ostatku programa**)

-dozvoljeno je i **izostaviti imena varijabli**, a pisati **samo** njihove **tipove** (tako se **obično i radi**)

-**tipovi (i imena) varijabli** u deklaraciji **odvajaju se zarezom**

-**deklaracija** funkcije **završava** standardnim znakom, tj. sa znakom **;**

-primjeri **liste argumenata** neke funkcije (pisano na **dva načina**, uz isti efekat)

(int a, float b, unsigned long int c, char slovo);

(int, float, unsigned long int, char);

-**broj argumenata**, njihov **redoslijed** i **tip** zovemo **potpisom funkcije** (engl. **function signature**)

-**potpis funkcije** je vrlo bitan za njezinu **pravilnu upotrebu**

-možemo pojednostavljeno reći da se **uvijek moramo služiti istim brojem, redoslijedom i tipom varijabli**, tj. imamo **funkcije istog potpisa kod deklaracije, definicije i upotrebe**

-zaključno, **deklaracija funkcije** zadaje samo **prototip** (engl. **prototype**) funkcije (**kako izgleda po pitanju primanja i vraćanja vrijednosti**), a **ne što i kako radi**

2.4.2. Definicija funkcije

-za razliku od deklaracije koja samo rezervira prostor u memoriji, **definicija funkcije** točno **zadaje naredbe koje tvore promatranu funkciju**

-definicija se piše **izvan** ***main()*** **funkcije**, obično **iza nje**, dakle **odvojeno od njezine deklaracije**

-**definicija funkcije** piše se na ovaj način:

```
tip_vraćene_vrijednosti ime_funkcije(tip argumenta1, tip argumenta2,...)
    {
    naredba 1;
    naredba 2;
    .
    .
    naredba n;
    return vrijednost;
    }
```

-vidljivo je da se **definicija i deklaracija funkcije** jednim dijelom pišu **slično**, a **razlike** definicije u odnosu prema deklaraciji su u ovom:

a) **imena varijabli u listi argumenata mogu biti bilo koja dopuštena** (**vrijede samo unutar** promatrane **funkcije**), ali se **obavezno moraju navesti**

-pomoću tih **varijabli** tvore se **naredbe** koje definiraju **ponašanje** funkcije

b) **iza liste argumenata ne piše se znak ;**, već znakovi ***{*** i ***}*** (**početak i kraj bloka naredbi**)

-tu **nije potrebno** pisati znak **;**, jer program zna **odrediti kraj i početak definicije funkcije** zbog upotrebe znakova ***{*** i ***}***

c) **unutar vitičastih zagrada** pišu se **sve naredbe** koje određuju **što** funkcija **radi**, a **svaka završava** znakom **;** (**osim** ako neka **naredba koristi oznake** ***{*** i ***}*** (početak i kraj bloka naredbi)

d) **zadnja naredba** mora biti **naredba** ***return*** iza koje **može slijediti**:

1.) **konstantna vrijednost**

-primjer:

return 0;

2.) **ime varijable**
-primjer:
return a;
3.) **neki izraz koji se izračunava**
-primjer;
return b*h;
-ukoliko se koristi **funkcija** koja **ne vraća vrijednost**, **ne treba** se koristi **naredba** ***return***
-treba reći da se **imenima varijabli** koja su zadana **u definiciji** funkcije koristimo samo za **definiranje ponašanja funkcije**, a **ne** i za njezin **poziv** (**izvršavanje**)
-zato se takva imena varijabli zovu **formalnim argumentima** (engl. **formal argument**)
-primjeri **definicija** funkcije:
-funkcija ***kvadrat()*** za izračun kvadrata broja:

```
int kvadrat (int broj)
{
int iznos; //deklariramo varijablu iznos koju ćemo moći koristiti samo unutar definicije ove funkcije
iznos=broj*broj;//ovdje kvadriramo zadani broj (varijablu) i pamtimo je kao varijablu iznos
return iznos;
}
```

-funkcija ***ispis_kuba()*** za računanje i ispis treće potencije zadanog broja:

```
void ispis_kuba (int broj)
{
int iznos; //deklariramo varijablu iznos koju ćemo moći koristiti samo unutar definicije ove funkcije
iznos=broj*broj*broj;//ovdje kubiciramo zadani broj (varijablu) i pamtimo je kao varijablu iznos
cout<<iznos;
//vidimo da nemamo naredbu return, jer vršimo ispis sadržaja varijable iznos, pa vraćanje vrijednosti
//nije potrebno
}
```

2.4.3. Poziv (upotreba) funkcije

-da bi se funkcija mogla iskorisiti, moramo je **upotrijebiti** (**pozvati na izvršenje**)
-funkciju **pozivamo** na izvršenje **unutar neke druge funkcije**, **uobičajeno** je to unutar funkcije ***main()***
-čak možemo napraviti da **funkcija poziva samu sebe** na izvršenje
-**poziv funkcije** je ovog oblika:

ime_funkcije(tip argumenta1, tip argumenta2,...);

-treba napomenuti da **u listi argumenata** sada koristimo **stvarne argumente** (engl. **actual argument**)
-stvarni argumenti navode se **istim redoslijedom** i moraju biti **istog tipa** kao i oni **zadani u deklaraciji i definiciji funkcije**, samo je njihov **iznos konkretan**
-**stvarni argumenti** navode se uobičajeno kao:

a) **konstantne vrijednosti**
b) **kao ime varijabli**

-**unutar poziva** iste funkcije možemo **miješati imena varijabli i konstantne vrijednosti**
-primjer:

```
izracun_formule(a, 10, 2.4, broj, 1.2e-3);
formula(a, 10, d, broj, malo);
```

-primjer funkcije za kvadriranje (cijeli program):

```
#include <cstdlib>
```

```
#include <iostream>
using namespace std;
float kvadrat(float);//deklaracija funkcije
int main(int argc, char *argv[])
{
  float broj, float d; //deklaracija varijabli
 broj=23.45;//inicijalizacija varijable
 d=kvadrat(broj);//poziv funkcije
  cout <<"Rezultat je "<<d<<"."<<endl;
  system("PAUSE")
  return EXIT_SUCCESS; //može se koristiti i naredbu return 0;
}
float kvadrat(float a) //definicija funkcije
{
 return a*a;
}
```

2.5. Ulazni i izlazni tokovi

-da bi programi ispunili svoju ulogu, moraju **komunicirati s okolinom**
-u C++ programskom jeziku to nam omogućuju tzv. **ulazni i izlazni tokovi** (engl. **input and output stream**)
-oni omogućuju **vezu** između našeg **programa** te **ulaznih i izlaznih uređaja** (**tipkovnica i zaslon**, po potrebi može to biti i neka **memorija**, npr. **hard disk** ili **USB flash memorija**)
-da bi u našem programu mogli **koristiti tokove**, moramo upotrijebiti naredbu *#include <iostream>* kojom definiramo koji **dio biblioteke standardnih funkcija** sadrži **funkcije za rad s tokovima** (dio **iostream**)
-uobičajeno se koriste ovi **tokovi**:

a) **cin**
-služi za **unos podataka** pomoću **tipkovnice** (**unos završava** pritiskom na tipku **Enter**)

b) **cout**
-služi za **ispis podataka** na **ekran monitora**

c) **cerr**
-namjena mu je **ispis poruka o pogreškama** pri izvršavanju programa na **ekranu** monitora
-**rijetko** se koristi

2.5.1. Tok cin

-**tok cin** namijenjen je za **unos podataka** pomoću **tipkovnice**
-**učitavanje podataka** ostvaruje se upotrebom tzv. **operatora izlučivanja** (engl. **extraction operator**)
-**operator izlučivanja** piše se ovako: >> (**dva znaka veće od** pisana **bez razmaka**)
-pri upotrebi toka **cin** zadajemo **ime varijable** u koju se **prenosi podatak** koji smo unijeli **tipkovnicom**
-**tip unesenog podatka** i **tip varijable** u koju se prenosi uneseni podatak moraju se **poklapati** (npr. ukoliko imamo varijablu tipa int, a otipkali smo slovo d, unos podatka neće se obaviti), **inače** se **upis vrijednosti ne obavlja** (kao da **nismo upotrijebili naredbu**)
-**unos podataka završava** se pritiskom na tipku **Enter**
-**način pisanja** pri upotrebi **toka cin** je ovaj:

cin>>ime_varijable;

-možemo radi **lakšeg pamćenja** zamisliti da je operator >> **strelica** usmjerena **udesno** (→) koja nam govori da se **podatak** iz toka **cin** (tok cin predstavlja **tipkovnicu**) **preslikava** u **varijablu** napisanu nakon operatora

-primjer:

```
float broj;
cin>>broj;//tipkovnicom unešeni broj tipa float upisuje se u varijablu broj
```

-upotrebom toka možemo **istodobno upisivati više podataka u više varijabli**
-u tu svrhu **svaki put** koristimo **operator** >> i **ime varijable** u koju se preslikava unešena vrijednost
-pritom unešeni **podaci** moraju biti **odvojeni prazninama**, a po potrebi se mogu **brisati** tipkom **Backspace**
-kada smo **unijeli sve** tražene **podatke**, **unos završavamo** pritiskom na tipku **Enter**
-nakon pritiska te tipke program **unešene podatke** smješta u za to **predviđene varijable**
-primjer:

```
int a, float b, char znak, unsigned long int broj;
cin>>a>>b>>znak>>broj;
```

-čest je slučaj da za svaki **podatak** koji **unesemo** upotrijebimo **ispis neke poruke**
-tada za **ispis poruke** koristimo tok **cout**, a **nakon** svakoga od njih za **upis** koristimo tok **cin**
-primjer nalik prijašnjem, ali uz upotrebu **tekstovnih** poruka:

```
int a, float b, char znak, unsigned long int broj;
cout<<"Unesite broj a "; //tu možemo staviti endl, ako želimo da se utipkani broj pojavi
//na početku novog reda ili ga izostavljamo, ako želimo da se broj pojavi u istom redu iza
//poruke
cin>>a;
cout<<endl;//tu samo početak iduće poruke prebacujemo u novi red
//endl smo mogli staviti i na početak iduće cout naredbe za ispis poruke
cout<<"Unesite broj b ";//ispis poruke
cin>>b;
cout<<endl;//skok na početak novog reda
cout<<"Unesite znak ";
cin>>znak;
cout<<"."//nakon unešenog znaka ispiše se točka pa cijeli red predstavlja rečenicu
cout<<endl;//skok na početak idućeg reda
cout<<"Unesite broj ";
cin>>broj;
cout<<endl;//skok na početak nove linije
```

2.5.2. Tok cout

-tok **cout** suprotan je toku cin, te služi za **ispis podataka na ekranu**
-vrijedi većina rečenog za tok cin, osim što tu koristimo **operator umetanja** (engl. **insertion operator**) za **slanje podataka na ispis**
-**operator umetanja** piše se kao << (**dva znaka manje od** pisana bez **razmaka**)
-možemo zamisliti da se radi o **strelici** usmjerenoj **ulijevo** (←) koja opisuje da se podatak s njezine **desne strane** šalje **na ekran** kojeg predstavlja riječ **cout**
-ukoliko **u istoj liniji** vršimo **ispis više podataka** (npr. **tekst i sadržaj varijable**), tada koristimo **po jedan operator** << za **ispis svakog podatka** poslanog na cout
-**nepromjenjivi tekst** piše se **unutar dvostrukih navodnika** (npr. *"Ispisujem vrijednost. "*)
-primjer:

```
float a=2.34343;
cout<<"Ovo je probni ispis.";//ispis običnog nepromjenjivog teksta
cout<<a;//tu ispisujemo iznos varijable a
```

-primjer ispisa **nepromjenjivog teksta i sadržaja varijable** u istom retku:

```
float a=2.34343;
cout<<"Ovo je probni ispis."<<a;//tu ispisujemo tekst i iznos varijable a
```

-**način ispisa** pomoću toka **cout** mijenja se upotrebom tzv. **manipulatora** (engl. **manipulator**)
-radi se o **riječima posebnog značenja** koje se **šalju na ispis** kao i svaka druga konstanta ili varijabla, a za **umetanje** koristimo operator **<<**
-češće korišteni **manipulatori** su:

a) **setw(broj_znamenki)**
-to je skraćenica od engl. **set width** (**postavi širinu ispisa cijelih brojeva**)
-dakle, njime zadajemo **koliko znamenki ispisujemo** (broj znamenki je **pozitivan cijeli broj**)
-ukoliko zadamo **broj znamenki manji od broja znamenki broja** koji ispisujemo, tada se **manipulator ignorira**
-time možemo **poravnavati ispis brojeva različite duljine** i sl.
-manipulator vrijedi **samo za jedan ispis** (**ne** nužno **za cijeli red**), nakon čega se vrši **uobičajeni ispis**
-primjer:

```
int a=233424;
float b=234.5;
cout<<"Broj a je <"<<setw(10)<<">."<<endl;//prebacujemo ispis u novi red
//ispisujemo 4 praznine, budući da je broj dug samo 6 znamenki
cout<<"Broj b je <"<<setw(2)<<">."<<endl;//premali broj mjesta, naredba se
//preskače i vrši se ispis svih znamenki
```

-rezultat prijašnjeg primjera je:
< 233424> (s lijeve strane su 4 razmaka između znaka < i znamenke 2)
<234.5>

b) **dec**
-zadaje da se varijabla ispisuje u **dekadskom** brojnom sustavu
-to je **podrazumijevani način** (engl. **default**) **ispisa** pa taj manipulator **ne moramo navoditi**

c) **hex**
-zadaje se da se varijabla ispisuje u **heksadekadskom** brojnom sustavu (baza **16**, znamenke **0** do **9**, **a**, **b**, **c**, **d**, **e**, **f**)
-taj način ispisa vrijedi **samo za jedan ispis**
-primjer:

```
int a=15;
cout<<hex<<a;
```

-rezultat je 0xf
-tu **0x** označava da se radi o **prikazu heksadekadskog** broja, a **f** je njegova vrijednost

d) **oct**
-zadaje se da se varijabla ispisuje u **oktalnom** brojnom sustavu (baza **8**, znamenke **0** do **7**)
-taj način ispisa vrijedi **samo za jedan ispis**
-primjer:

```
int a=15;
cout<<oct<<a;
```

-rezultat je 017
-tu **0** označava da se radi o **prikazu oktalnog broja**, a **17** je njegova vrijednost

e) **endl**
-ovaj manipulator vrši **prebacivanje ispisa u novi red**
-primjer:

```
int b=13;
cout<<"Broj je"<<endl<<b;
```

-rezultat je Broj je

2.5.3. Tok cerr

-ovaj tok **rijetko** se koristi, a namijenjen je za **ispis poruke o pogrešci u radu programa** na **zadani uređaj** (većinom je to **monitor**)

2.5.4. Ostali tokovi

-nama su zanimljivi tokovi za **upis i ispis podatka u datoteku**
-u tu svrhu koriste se tokovi **ifstream** (**i** skraćeno od **input**, **f** skraćeno od **file**) za **učitavanje podataka iz datoteke**, **ofstream** (**o** skraćeno od **output**, **f** skraćeno od **file**) za **upis podataka u datoteku** i tok **fstream** kojim se može vršiti **čitanje i upis u datoteku**
-da bi mogli koristiti navedene tokove moramo na početku programa zadati naredbu za **uključenje dijela biblioteke standardnih funkcija** koji nosi oznaku **fstream** (dakle, naredba je **#include <fstream>**)

2.7. Aritmetički operatori

-**aritmetičke operacije** obavljaju se u skoro svakom iole dužem programu
-zato je vrlo bitno znati se njima služiti u programskom jeziku C++
-znanje iz **matematike** (vrste operacija, operatori, pravila za računanje izraza, prednost operacija,...) može se u velikom opsegu primijeniti kod upotrebe aritmetičkih operacija u programskom jeziku C++
-za pojedine aritmetičke operacije zadani su **operatori** koje možemo koristiti za realizaciju promatranih operacija
-**aritmetičke operatore** dijelimo u **dvije grupe**, ovisno na koliki **broj operanada** djeluju
-to su:

a) **unarni operatori**
-djeluju samo na **jedan** operand
b) **binarni operatori**
-djeluju na **dva** operanda
-tu se pojam **binarno** ne smije brkati s pojmom binarni brojni sustav, jer ovdje pojam binarno označava samo činjenicu da operator vrši operaciju nad **dva** operanda

2.7.1. Unarni aritmetički operatori

-u ovu grupu spadaju sljedeći operandi:

a) unarni plus
b) unarni minus
c) uvećaj nakon
d) uvećaj prije
e) umanji nakon
f) umanji prije

2.7.1.1. Unarni plus

-ovaj operator **mijenja predznak broja u pozitivni**
-**sintaksa** mu je:

+ime_varijable

-dakle, **ispred** varijable piše se znak **+** (**bez razmaka**)
-primjer:

```
a=+b;//varijabla b postaje pozitivna i njezina vrijednost pamti se pod imenom a
```

-posebno treba pripaziti na to da se **ne** napiše **obrnuto** (što nije pogrešno sa stanovišta kompajlera pa neće javiti poruku o pogreški), tj. `a+=b;//to je isto što i a=a+b;`

2.7.1.2. Unarni minus

-ovaj operator **mijenja predznak** broja u **negativni**

-**sintaksa** mu je:

-ime_varijable

-dakle, **ispred** varijable piše se znak **-** (**bez razmaka**)

-primjer:

```
a=-b;//varijabla b postaje negativna i njezina vrijednost pamti se pod imenom a
```

-posebno treba pripaziti na to da se **ne** napiše **obrnuto** (što nije pogrešno sa stanovišta kompajlera pa neće javiti poruku o pogreški), tj. *a-=b;//to je isto što i a=a-b;*

2.7.1.3. Uvećaj nakon

-ovaj operator **uvećava** vrijednost varijable za **1**

-takva operacija uglavnom se vrši nad **cijelim** brojevima, tj. na tipu **int**

-**uvećanje** vrijednosti **cjelobrojne** varijable za **1** naziva se još i **inkrementiranje** (engl. **increment**)

-**način pisanja** naredbe:

ime_varijable++

-primjer:

```
a=12;//a ima početnu vrijednost 12
a++; //sada a postaje jednak 13
cout<<a<<endl; //na ekranu ispišemo 13 i prebacimo ispis na početak novog reda
```

-ova operacija često se koristi u **petlji** s **određenim brojem ponavljanja** (**for** petlja) kod **uzlaznog** brojanja (**od manjeg prema većem**)

-treba biti vrlo **oprezan** kod upotrebe ove naredbe kada se ona koristi za **dodjeljivanje vrijednosti**

-u tom slučaju **prvo** se vrši **dodjeljivanje** vrijednosti, a tek nakon toga varijabla se **poveća za 1**

-primjer:

```
b=3;
c=15;
b=c++;//tu se prvo napravi operacija pridruživanja b=c, a potom se c poveća za 1
//rezultat je da je b=15, a c 16, a ne da su oba 16 kako bi se na prvi pogled očekivalo
```

-problemi ovog tipa mogu se izbjeći ukoliko ovu operaciju koristimo **zasebno u liniji** (**bez pridruživanja**), a potom **u idućoj liniji** napravimo **pridruživanje**

-primjer nalik na prijašnji:

```
b=3;
c=15;
c++;//c je 16
b=c;//b i c su 16
```

2.7.1.4. Uvećaj prije

-ovaj operator **uvećava** vrijednost varijable za **1**

-takva operacija **uglavnom** se vrši nad **cijelim** brojevima, tj. na tipu **int**

-**način pisanja** naredbe:

++ime_varijable

-primjer:

```
a=12;//a ima početnu vrijednost 12
++a; //sada a postaje jednak 13
cout<<a<<endl; //na ekranu ispišemo 13 i prebacimo ispis na početak novog reda
```

-pri upotrebi ove naredbe, kada se ona koristi za **dodjeljivanje** vrijednosti, **prvo** se vrši **povećanje** vrijednosti za 1, potom **dodjeljivanje** vrijednosti (kod operatora za **uvećanje nakon** situacija je obrnuta)

-primjer:

```
b=3;
c=15;
b=++c; //tu se prvo napravi povećanje c za 1, a potom operacija pridruživanja b=c
//rezultat je da su b i c jednaki 16
```

2.7.1.5. Umanji nakon

-ovaj operator **umanjuje** vrijednost varijable za **1**
-takva operacija **uglavnom** se vrši nad **cijelim** brojevima, tj. na tipu **int**
-**umanjenje** vrijednosti cjelobrojne varijable za **1** naziva se još i **dekrementiranje** (engl. **decrement**)
-**način pisanja** naredbe:

ime_varijable--

-primjer:

```
a=12;//a ima početnu vrijednost 12
a--; //sada a postaje jednak 11
cout<<a<<endl; //na ekranu ispišemo 11 i prebacimo ispis na početak novog reda
```

-ova operacija često se koristi kod **petlje** s **određenim brojem** ponavljanja (**for** petlja) kod **silaznog** brojanja (od većeg prema manjem)
-vrlo **oprezan** treba se biti kod upotrebe ove naredbe kada se ona koristi kod **dodjeljivanja** vrijednosti
-u tom slučaju **prvo** se vrši **dodjeljivanje** vrijednosti, a tek nakon toga varijabla se **umanji** za **1**
-primjer:

```
b=3;
c=15;
b=c--;//tu se prvo napravi operacija pridruživanja b=c, a potom se c umanji za 1
//rezultat je da je b=15, a c 14, a ne da su oba 14 kako bi se na prvi pogled očekivalo
```

-problemi ovog tipa mogu se izbjeći ukoliko ovu operaciju koristimo **zasebno** u liniji (bez pridruživanja), a potom u idućoj liniji napravimo **pridruživanje**
-primjer nalik na prijašnji:

```
b=3;
c=15;
c--;//c je 14
b=c;//b i c su 14
```

2.7.1.6. Umanji prije

-ovaj operator **umanjuje** vrijednost varijable za **1**
-takva operacija uglavnom se vrši nad **cijelim** brojevima, tj. na tipu **int**
-**način pisanja** naredbe:

--ime_varijable

-primjer:

```
a=12;//a ima početnu vrijednost 12
--a; //sada a postaje jednak 11
cout<<a<<endl; //na ekranu ispišemo 11 i prebacimo ispis na početak novog reda
```

-pri upotrebi ove naredbe, kada se ona koristi kod dodjeljivanja vrijednosti, **prvo** se vrši **umanjenje** vrijednosti za **1**, a potom **dodjeljivanje** vrijednosti (kod operatora za **umanjenje nakon** situacija je obrnuta)
-primjer:

```
b=3;
c=15;
b=--c;//tu se prvo napravi umanjenje c za 1, a potom operacija pridruživanja b=c
//rezultat je da su b i c jednaki 14
```

2.7.2. Binarni aritmetički operatori

-to su:

a) **zbrajanje**
b) **oduzimanje**
c) **množenje**
d) **dijeljenje**
e) **ostatak cjelobrojnog dijeljenja**

2.7.2.1. Zbrajanje

-operator **zbrajanja** obavlja operaciju **zbrajanja** nad **dva** operanda
-ukoliko su **oba** operanda **cijeli** brojevi, rezultat je **cijeli** broj (tip **int**), inače je rezultat u pokretnom zarezu (tip **float**)
-kao i u matematici, operator zbrajanja je znak **+** napisan između dva operanda koji mogu biti **varijable i/ili konstante**
-**način pisanja**:

operand1 + operand2

-preporučljivo je zbog **preglednosti** između znaka + i operanda ostaviti **razmak**, ali nije obavezno
-primjer:

```
a=23;
b=17;
c=a + b; //c je 40
```

-u istom redu mi možemo obaviti **zbrajanje više operanada**, ali operacija djeluje uvijek na **po dva** operanda, počevši od znaka **jednakosti**
-primjer:

```
a=20;
b=12;
c=6;
d=31;
e=a + b + c + d; //e je 20+12+6+31=69
```

-pošto operacija uvijek djeluje na dva operanda, vrše se radnje kao da je izraz napisan u obliku:

```
e=((a + b) + c) + d;
```

-dakle, prvo se zbroje dva operanda najbliža znaku = (**a** i **b**), na taj zbroj doda se **c** i na taj zbroj pribrojimo **d** da bismo dobili konačni rezultat za **e**

2.7.2.2. Oduzimanje

-operator **oduzimanja** obavlja operaciju **oduzimanja** nad **dva** operanda
-ukoliko su **oba** operanda **cijeli** brojevi, rezultat je **cijeli** broj (tip **int**), inače je rezultat u pokretnom zarezu (tip **float**)
-kao i u matematici, operator oduzimanja je znak **-** napisan između **dva** operanda koji mogu biti **varijable i/ili konstante**
-**način pisanja**:

operand1 - operand2

-preporučljivo je zbog **preglednosti** između znaka - i operanda ostaviti **razmak**, ali nije obavezno
-primjer:

```
a=23;
b=17;
c=a - b; //c je 6
```

-u istom redu mi možemo obaviti **oduzimanje više operanada**, ali operacija djeluje uvijek na **po dva** operanda, počevši od znaka **jednakosti**
-primjer:

```
a=20;
b=12;
c=6;
d=31;
e=a - b - c - d; //e je 20-12-6-31=-29
```

-pošto operacija uvijek djeluje na dva operanda, vrše se radnje kao da je izraz napisan u obliku:

```
e=((a - b) - c) - d;
```

-dakle, prvo se oduzmu dva operanda najbliža znaku = (**a** i **b**), od te razlike oduzme se **c** i od nje oduzmemo **d** da bismo dobili konačni rezultat za **e**

2.7.2.3. **Množenje**

-operator množenja obavlja operaciju **množenja** nad **dva** operanda
-ukoliko su oba operanda **cijeli** brojevi, rezultat je cijeli broj (tip **int**), inače je rezultat u pokretnom zarezu (tip **float**)
-za razliku od matematike, operator množenja je znak ***** napisan između dva operanda koji mogu biti **varijable i/ili konstante**
-**način pisanja**:

operand1 * operand2

-preporučljivo je zbog **preglednosti** između znaka * i operanda ostaviti **razmak**, ali nije obavezno
-primjer:

```
a=20;
b=17;
c=a * b; //c je 340
```

-u istom redu mi možemo obaviti **množenje više operanada**, ali operacija djeluje uvijek na **po dva** operanda, počevši od znaka **jednakosti**
-primjer:

```
a=2;
b=4;
c=6;
d=3;
e=a * b * c * d; //e je 2*4*6*3=144
```

-pošto operacija uvijek djeluje na dva operanda, vrše se radnje kao da je izraz napisan u obliku:

```
e=((a * b) * c) * d;
```

-dakle, prvo se pomnože dva operanda najbliža znaku = (**a** i **b**), na taj umnožak doda se **c** i njega pomnožimo s **d** da bismo dobili konačni rezultat za **e**

2.7.2.4. **Dijeljenje**

-operator **dijeljenja** obavlja operaciju **dijeljenja dva** operanda
-ukoliko su oba operanda **cijeli** brojevi, vrši se operacija **cjelobrojnog dijeljenja** te je rezultat cijeli broj (tip **int**)
-ukoliko je bar **jedan** operand **realan** broj (u pokretnom zarezu), vrši se operacija **dijeljenja realnih** brojeva te je rezultat **realan** broj (tip **float**)
-za razliku od matematike, **operator dijeljenja** je znak **/** napisan između dva operanda koji mogu biti **varijable i/ili konstante**
-**način pisanja**:

operand1 / operand2

-preporučljivo je zbog **preglednosti** između znaka / i operanda ostaviti **razmak**, ali nije obavezno
-primjer:

```
a=20;
b=17;
c=a / b; //c je 1, jer su oba broja cijela pa je i rezultat cijeli broj
```

-primjer:

```
a=20.0;
b=17;
c=a / b; //c je približno 1.1765, jer je bar jedan broj u pokretnom zarezu (20.0) pa je i rezultat broj
//u pokretnom zarezu
```

-iz oba prijašnja primjera vidljiva je **velika razlika u rezultatu** ukoliko zaboravimo da li želimo **cjelobrojno dijeljenje** ili **dijeljenje brojeva u pokretnom zarezu**
-u istom redu mi možemo obaviti **dijeljenje više operanada**, ali operacija djeluje uvijek na **po dva** operanda, počevši od znaka **jednakosti**

-primjer:

```
a=200;
b=4;
c=6;
d=3;
e=a / b / c / d; //e je 200/4/6/3=2
```

-pošto operacija uvijek djeluje na dva operanda, vrše se radnje kao da je izraz napisan u obliku:

```
e=((a / b) / c) / d; // ((200/4)/6)/3=(50/6)/3=8/3=2
```

-dakle, prvo se podijele dva operanda najbliža znaku = (**a** i **b**), taj kvocijent podijeli se s **c** i potom s **d** da bismo dobili konačni rezultat za **e**

2.7.2.5. Ostatak cjelobrojnog dijeljenja

-operator za **ostatak cjelobrojnog dijeljenja** obavlja operaciju traženja ostatka cjeloborojnog dijeljenja **dva** operanda
-oba operanda moraju biti **cijeli** brojevi pa je i rezultat **cijeli** broj
-probamo li tu operaciju obaviti nad **realnim** brojem, kompajler će nam javiti poruku o **pogrešci**
-**matematički** se ta operacija bilježi oznakom **mod** ili **modulo**, dok mi u **programiranju** koristimo operand **%**
-izraz **a modulo b** daje rezultat u opsegu od **0 do b-1** (**0**, ako su **a** i **b** međusobno **djeljivi bez ostatka**)
-operandi mogu biti **varijable i/ili konstante**
-**način pisanja**:

operand1 % operand2

-preporučljivo je zbog **preglednosti** između znaka % i operanda ostaviti **razmak**, ali nije obavezno
-primjer:

```
a=20;
b=17;
c=a % b; //c je 3, jer je a/b=1, a ostatak je 3
```

-primjer:

```
a=20;
b=17;
c=a / b; //c je 1
```

-iz oba prijašnja primjera vidljiva je **velika razlika u rezultatu** ukoliko pobrkamo operatore / i **%**
-u istom redu mi možemo obaviti traženje ostatka cjelobrojnog dijeljenja za **više operanada**, ali operacija djeluje uvijek na **po dva** operanda, počevši od znaka **jednakosti**
-primjer:

```
a=200;
b=24;
c=5;
d=3;
e=a % b % c % d; //e je 200 mod 24 mod 5 mod 3=0
```

-pošto operacija uvijek djeluje na dva operanda, vrše se radnje kao da je izraz napisan u obliku:

```
e=((a % b) % c) % d; // ((200 mod 24) mod 5) mod 3=(8 mod 5) mod 3=3 mod 3=0
```

-dakle, prvo se traži ostatak dijeljenja dva operanda najbliža znaku = (**a** i **b**) i tako redom do **d**

2.7.3. Prednost operatora i upotreba zagrada

-slično kao i u matematici, dogovoreni je **redoslijed izvršavanja aritmetičkih operacija**, ako ne koristimo zagrade
-takav **dogovoreni redoslijed** djelovanja operatora nazivamo **hijerarhijom** (**prioritetom**) **operatora**
-po **hijerarhiji** su aritmetički operatori podijeljeni u **grupe** unutar kojih su operatori **istog prioriteta**
-to su ove **grupe** s pripadajućim operatorima (prva grupa ima najviši prioritet):

a) **uvećaj nakon** (**x++**), **umanji nakon** (**x--**)

b) **uvećaj prije** (**++x**), **umanji prije** (**--x**), **unarni plus** (**+x**), **unarni minus** (**-x**)
c) **množenje** (*****), **dijeljenje** (**/**), **ostatak cjelobrojnog dijeljenja** (**%**)
d) **zbrajanje** (**+**), **oduzimanje** (**-**)

-ukoliko koristimo **zagrade**, one **mijenjaju prioritete** operatora kao što vrijedi i u matematici
-ukoliko imamo **zagrade u zagradama**, izrazi u njima računaju se prvi, a potom redom izrazi **prema vanjskim zagradama**
-primjer **bez zagrada**:

```
int a=130;
int b=12;
int c=7;
int d;
d= a * b / c + a * c / b;
```

-po prioritetima operatora izraz se počinje izvršavati od znaka = prema desnoj strani
-budući da su sve operacije osim zbrajanja istog prioriteta, najprije se vrši množenje a*b (130*12=1560), a potom se taj rezultat dijeli s **c** (1560/7=222)
-nakon toga ne vrši se zbrajanje trenutnog međurezultata s **a** (~~222+130~~), jer se vrši računanje drugog izraza zbog prioriteta (a*c=130*7=910, potom 910/12=75)
-na kraju se zbrajaju oba međurezultata i dobiva se 297 (222+75=297)
-isti izraz napisan upotrebom **zagrada** daje drukčiji rezultat
-primjer:

```
int a=130;
int b=12;
int c=7;
int d;
d= a * b / ( c + a) * c / b;
```

-najprije se izračuna izraz u zagradi (c+a=137), a potom se računa s lijeve na desnu stranu redom po dva operanda, tj. a*b/137*c/b=1560/137*c/b=11*c/b=11*7/b=77/b=77/12=6
-**zaključak**: ako nismo sigurni u **prioritet** izraza, upotrijebimo **zagrade** i time definiramo **vlastite prioritete** u izračunima
-ukoliko ni u to nismo sigurni, **složeni izrazi** se mogu rješavati **razlomljeni** na **više jednostavnijih dijelova**

2.7.4. Operatori obnavljajućeg pridruživanja za aritmetičke operacije

-kod programiranja često se koristimo operacijama kod kojih **mijenjamo** izračunom **vrijednost jedne varijable**, a nova vrijednost te varijable **pamti** se pod **istim imenom** kao i stara
-primjer:

```
int a=17;
a=a+3;
//tu povećavamo vrijednost a za 3 i pamtimo ju pod imenom a
//a na desnoj strani znaka jednakosti je vrijednost a prije ovog izraza (a=17), dok je a na lijevoj
//strani znaka jednakosti a nakon izračunavanja promatranog izraza (a=17+3=20)
```

-ovakve izraze možemo **kraće zapisati** pomoću tzv. **operatora obnavljajućeg pridruživanja** (engl. **update assignment**)
-time **nismo** ništa dobili na **brzini** izvršavanja naredbi, ali smo **skratili** njen **zapis**
-postoje slijedeći **aritmetički operatori obnavljajućeg pridruživanja**:

a) **+=** (isto što i a=a+)
b) **-=** (isto što i a=a-)
c) ***=** (isto što i a=a*)
d) **/=** (isto što i a=a/)
e) **%=** (isto što i a=a%)

-primjeri:

```
a+=3;//a=a+3;
```

a-=3;//a=a-3;
a=3;//a=a*3;*
a/=3;//a=a/3;
a%=3;//a=a%3;

-moramo **pripaziti** na dvije stvari kod njihove upotrebe:

a) da se kod operatora **+=** i **-=** ne zabunimo u **poretku** njihovih znakova
-naime, =+ i =- znače promjenu predznaka i nakon toga pridruživanje, a to su dozvoljene operacije i kompajler **neće javiti pogrešku**

b) operatori obnavljajućeg pridruživanja imaju **niži prioritet** od svih ostalih aritmetičkih operacija i možemo **pogriješiti** ukoliko ih koristimo **u sklopu nekog složenijeg izraza**
-ako nismo sigurni u njihove prioritete, najbolje je da ih upotrijebimo kao **zasebno izračunane izraze** ili da ih **ne koristimo**

-primjer:

*a=-32;//**a** postaje jednak -32*
a-=32;//a=a-32;, tj. a=-32-32=-64

-primjer:

int a=23, int b=3, int c=7;
a-=c-b;//tu se prvo vrši operacija c-b (7-3=4), a potom a=a-4;, tj. a=0

-kada bi bili isti prioriteti onda bi se izraz pretvorio u a= a-(c-b)
-kada bismo željeli izračunati a=a-b-c, a napisali a-=b-c, to bi bilo **pogrešno**, jer je jednako a= a-(c-b)

2.7.5. Brzina izvršavanja aritmetičkih operacija

-u nekim primjenama vrlo je bitno dobiti **čim brži** program
-u tome si možemo pomoći ukoliko smo svjesni u kojim odnosima su **brzine izvršavanja operacija** na računalima
-trebamo znati da **brzine operacija** nad brojevima u **pokretnom zarezu** i nad **cijelim** brojevima **nisu iste** (obično su operacije s **cijelim** brojevima **brže**, ali ne uvijek)
-osim toga, **unutar istog prikaza** broja postoji **razlika u brzini**, ovisno o tome koliko **memorije** troši neki podtip (npr. varijabla tipa int, tipa short int i long int troše različitu količinu memorije te su operacije različite brzine)
-čim tip troši **više memorije** za prikaz, **operacije** nad njim su **sporije** od onih na kraćim tipovima
-**unarni** operatori su tipično **brži** od **istih binarnih**
-primjer:

p++;//brža operacija
p=p+1;//sporija operacija

-ukoliko se radi **o istom tipu** podataka, otprilike možemo ustvrditi ove **odnose brzina izračunavanja** (od **bržeg prema sporijem**):

a) **zbrajanje, oduzimanje**
-obje operacije su **podjednako brze**
-nastojimo ih upotrijebiti **umjesto drugih operacija**, ako je to moguće
-primjer:

*y=3*x;//sporija naredba*
y=x+x+x;//brža naredba

b) **množenje**
-**dosta** je **sporija** operacija od **zbrajanja i oduzimanja**, oko **10-ak puta**
-neka množenja mogu se zamijeniti puno bržim operacijama nad cijelim brojevima (množenje s potencijama broja 2 svodi se na pomak bitova broja za određeni broj mjesta ulijevo)

c) **dijeljenje i ostatak cjelobrojnog dijeljenja**
-ovo su **najsporije operacije** pa ih nastojimo pogodnim načinom **izbjeći** ili **smanjiti njihov broj**
-primjerice, ukoliko **dijelimo s konstantom**, tu operaciju zamjenjujemo **množenjem** s unaprijed izračunatom **inverznom vrijednošću**
-primjer:

float a=2.5;

```
float b;
b=b/2.5//sporije izvođenje
b=b*0.4;//puno brže izvođenje
```

-slično kao i kod množenja, dijeljenje cijelih brojeva s potencijama broja 2 svodi se na pomak bitova za određeni broj mjesta udesno

-za dobijanje **brzih programa** nužno je izraz koji trebamo izračunati **preoblikovati** u oblik koji omogućuje brže izvršavanje

-novodobiveni izraz pritom može biti **dužeg zapisa**, ali **brži** pri izvršavanju

-bitno je samo upotrijebiti **čim manje sporih** operacija, a po potrebi **povećati broj bržih** operacija

-primjer: Izračunajte izraz $y=x^3+3x^2+4x+2$.

-zadani izraz može se zapisati kao: $y=(x+1)^3+x+1=(x+1)\{(x+1)^2+1\}$

-primjer programa za originalni izraz (ne koristimo funkciju za potenciranje, već operator množenja):

```
pomoc=x*x;//upotrijebili smo pomoćnu varijablu za izračun kvadrata varijable x
y=x*pomoc; //računamo x^3
pomoc=pomoc+pomoc+pomoc;//brža varijanta, nego 3*pomoc (izbjegli smo množenje)
y=y+pomoc;// x^3+3x^2
pomoc=x+x+x+x//tri zbrajanja brža su od jednog množenja
y=y+pomoc; //x^3+3x^2+4x
y=y+2
//upotrijebili smo dva množenja i osam zbrajanja
```

-isti primjer riješen pomoću **modificiranog** izraza

```
y=x+1;//x+1
pomoc=y*y;//(x+1)^2
pomoc=pomoc+1;// (x+1)^2+1
y=y*pomoc;// (x+1)^3+x+1
//upotrijebili smo dva množenja i dva zbrajanja
//razlika u brzini bila bi veća da u prvom primjeru umjesto množenja nismo upotrijebili zbrajanja
```

2.7.6. Realizacija potenciranja

-C++ **nema** ugrađeni **operator potenciranja** kao neki drugi programski jezici

-umjesto toga programer može napisati **svoju funkciju**, ako mu je bitna **brzina** izvođenja operacije potenciranja

-ne želimo li pisati svoju verziju funkcije za brzo potenciranje, na raspolaganju nam je **ugrađena funkcija** za **potenciranje** čija **deklaracija** glasi

float pow(float baza, float eksponent)

-dakle, funkcija vraća **rezultat float** tipa, a istog tipa su **baza** i **eksponent** potencije

-da bismo mogli koristiti tu funkciju, na **početku** programa moramo napisati naredbu za **uključenje** dijela standardne **biblioteke** funkcija u kojoj se nalazi definicija **pow()** funkcije

-naredba za to je:

#include <cmath>

-primjer:

```
#include <cstdlib>
#include <iostream>
#include <cmath>

using namespace std;

int main(int argc, char *argv[])
{
   float x,y,z;
   x=3.0;
```

```
    y=4.0;
    z=pow(x,y);
    cout<<z<<endl;
    system("PAUSE");
    return EXIT_SUCCESS;
  }
```

-rezultat programa je 81 (3^4=81)

2.8. Vježbe - aritmetički operatori

-u ovoj cjelini učimo upotrebu **aritmetičkih operatora** na **jednostavnim** primjerima zadataka

-primjer: Izračunajte y=(x+2)(2-x)+1 uz x=1.5.

-rješenje:

```
#include <cstdlib>
#include <iostream>

using namespace std;

int main(int argc, char *argv[])
{
    float x,y;
    x=1.5;
    y=(x+2)*(2-x)+1;
    cout<<y<<endl;
    system("PAUSE");
    return EXIT_SUCCESS;
}//rezultat je 2.75
```

-primjer: Izračunajte y= 5x-3/x za x=1.5.

-rješenje:

```
#include <cstdlib>
#include <iostream>

using namespace std;

int main(int argc, char *argv[])
{
    float x,y;
    x=1.5;
    y=5*x-3/x;
    cout<<y<<endl;
    system("PAUSE");
    return EXIT_SUCCESS;
}//rješenje je 5.5
```

primjer: Izračunajte y=123 mod x +2x -3 uz x=12 (x je cjelobrojni).

-rješenje:

```
#include <cstdlib>
#include <iostream>

using namespace std;

int main(int argc, char *argv[])
{
```

```
    int x,y;
    x=12;
    y=123 % x +2*x -3;
    cout<<y<<endl;
    system("PAUSE");
    return EXIT_SUCCESS;
  }//rješenje je 24
```

primjer: Izračunajte $y=x^2-3x+2$ uz x=12 (x je cjelobrojni).
-rješenje:

```
  #include <cstdlib>
  #include <iostream>

  using namespace std;

  int main(int argc, char *argv[])
  {
    int x,y,pomoc;
    x=12;
    y=x*x;//x^2
    pomoc=3*x;
    y=y-pomoc;//x^2-3x
    y=y+2;
    cout<<y<<endl;
    system("PAUSE");
    return EXIT_SUCCESS;
  } //rješenje je 110
```

2.9. Logički operatori

-za operacije nad podacima logičkog tipa (**bool**) uvedeni su **operatori** koji obavljaju **osnovne logičke operacije**
-da se prisjetimo: u **logičkom** tipu postoje samo **dvije vrijednosti** (**stanja**)
-to su **true** (istina - češće se piše kao broj **1**, mada to može biti bilo koji cijeli broj osim 0) i **false** (laž - češće se piše kao broj **0**)
-**logičke** podatke i operacije uglavnom koristimo kod upotrebe **grananja** u programu (naredba **if**) u kombinaciji s **operatorima usporedbe**
-osnovne **logičke operacije** su:

a) **NE** (engl. **not**)
b) **I** (engl. **and**)
c) **ILI** (engl. **or**)

-**ponašanje** logičkih operatora možemo opisati **riječima** ili zadati tzv. **tablicama istine** (engl. **truth table**) u kojima se navode **sve kombinacije ulaznih vrijednosti** i njihova **izlazna stanja**

2.9.1. Logička negacija (NE)

-ova operacija definirana je samo za **jednog** operanda (**unarna** operacija)
-ona svaku **ulaznu vrijednost pretvara u njezinu suprotnu** vrijednost, dakle **0 u 1** i **1 u 0**
-**tablica istine** je:

A	NE A
0	1
1	0

-ukoliko pišemo **matematičke formule**, tada je ovaj operator **crtica iznad** (¯)imena varijable
-primjer: Napišite izraz za Z koji je negacija varijable A.
-rješenje: $Z = \bar{A}$
-ovu operaciju u C++ jeziku obavlja operator **!**
-treba istaknuti da ovaj operator ima **prednost** u odnosu na druga dva logička operatora, ukoliko u izrazu ne koristimo **zagrade**
-primjer:

```
bool x=0;
x=!x;//x postaje 1, tj. true
```

2.9.2. Logičko I

-ova operacija definirana je za **dva** operanda (**binarna** operacija)
-ona daje **1** **samo ako su oba operanda 1**
-drukčije izraženo: daje **0**, ako je **bilo koji** operand jednak **0**
-koristimo ga za situacije koje u svakodnevnom govoru izražavamo riječima poput: **mora biti i ovo, i ovo**
-**tablica istine** je:

A	B	A I B
0	0	0
0	1	0
1	0	0
1	1	1

-ukoliko pišemo **matematičke formule**, tada je ovaj operator znak **·** između imena varijabli (ili ga **ne pišemo** kao kod **množenja**)
-primjer: Napišite izraz za Z koji je jednak I operaciji između varijabli A i B.
-rješenje: Z=A·B=AB
-ovu operaciju u C++ jeziku obavlja operator **&&** (**bez razmaka** između znakova)
-treba istaknuti da ovaj operator i operator za logičku **ILI** operaciju imaju **iste prioritete**, ali **niže od NE** operatora
-primjer:

```
bool x=0, y=1, z;
z=x && y;//z je 0, tj. false
```

2.9.3. Logičko ILI

-ova operacija definirana je za **dva** operanda (**binarna** operacija)
-ona daje **0** **samo ako su oba operanda 0**
-drukčije izraženo: daje **1**, ako je **bilo koji** operand jednak **1**
-koristimo ga za situacije koje u svakodnevnom govoru izražavamo riječima poput: **može biti ili ovo, ili ovo**, odnosno **barem da je ovo**
-**tablica istine** je:

A	B	A ILI B
0	0	0
0	1	1
1	0	1
1	1	1

-ukoliko pišemo **matematičke formule**, tada je ovaj operator znak **+** između imena varijabli
-primjer: Napišite izraz za Z koji je jednak ILI operaciji između varijabli A i B.

-rješenje: Z=A+B
-ovu operaciju u C++ jeziku obavlja operator || (**bez razmaka** između znakova, dobije se kombinacijom tipki **Alt Gr** i **W**)
-treba istaknuti da ovaj operator i operator za logičku **I** operaciju imaju **iste prioritete**, ali **niže od NE** operatora
-primjer:

```
bool x=0, y=1, z;
z=x || y;//z je 1, tj. true
```

2.10. Vježbe - logički operatori

-u ovoj cjelini učimo **upotrebu logičkih operatora** na **jednostavnim** primjerima zadataka
-primjer: Realizirajte operaciju suprotnu I operaciji za ulazne podatke x i y jednake 0.

```
bool x=0, y=0, z;
z=x && y;//I operacija, z je 0
z=!z;//negiramo I operaciju, z je 1
//mogli smo riješiti i u jednoj liniji, ali uz upotrebu zagrada
//tada bi bilo z=!(x && y);
```

-dobili smo **suprotnu** operaciju od **I** koja se zove **NI** operacija (engl. **nand**)
-primjer: Realizirajte operaciju suprotnu ILI operaciji za ulazne podatke x i y jednake 1.

```
bool x=1, y=1, z;
z=x || y;//ILI operacija, z je 1
z=!z;//negiramo ILI operaciju, z je 0
//mogli smo riješiti i u jednoj liniji, ali uz upotrebu zagrada
//tada bi bilo z=!(x || y);
```

-dobili smo **suprotnu** operaciju od **ILI** koja se zove **NILI** operacija (engl. **nor**)
-primjer: Realizirajte dvaput za redom NE operaciju za ulazni podatak x jednak 1.

```
bool x=1, z;
z=!x;//negirano x, z=0
z=!z;//negiramo z, te je z=1
```

-vidi se da smo dobili **početnu vrijednost**
-dakle, **dvostruka negacija nema učinka** na promjenu vrijednosti operanda
-primjer: Negirajte varijable x i y, potom među njima provedite I operaciju, a na kraju napravite ILI operaciju toga međurezultata s konstantom 0. Početne vrijednosti su x=1 i y=0.

```
bool x=1, y=0, a, b, z;
a=!x;//negiramo x, x=0
b=!y;//negiramo y, y=1
z=x && y;//I operacija, z je 0
z=z || 0;// z je 0
```

2.11. Operatori uspoređivanja (relacijski operatori)

-pomoću njih postižemo **usporedbu dva podatka** (najčešće **brojeva**, ali može i znakova)
-**brojevi** koje uspoređujemo mogu biti **bilo kojeg tipa**, dok je **rezultat uvijek logičkog tipa**, tj. **0** (**false**) ili **1** (**true**)
-dakle, ako je **usporedba zadovoljena**, rezultat je **1** (**istina**), **inače 0** (**laž**)
-ove operatore uobičajeno koristimo kod **naredbi grananja** (**if**)
-pritom ih možemo **povezivati** pomoću **logičkih operatora** da dobijemo **složene uvjete**
-svi operatori usporedbe djeluju na **dva** operanda, tj. **binarni** su
-pošto imaju **najniži prioritet** od svih do sada uvedenih operatora, dobra navika je **pisanje cijelog izraza** koji uspoređujemo u **zagradama**
-postoje slijedeći **operatori usporedbe**:

a) **manje od**
b) **manje od ili jednako**

c) **veće od**
d) **veće od ili jednako**
e) **jednako**
f) **različito od**

2.11.1. Operator manje od

-ovaj operator uspoređuje **dva** broja
-ukoliko je broj s **lijeve** strane operatora **manji od** broja s **desne** strane, tada je rezultat **true**, inače je **false**
-**oznaka** operatora je ista kao u matematici, tj. <
-primjer:

```
bool z;
int a=23, b=17;
z=(a < b);//pošto nije a < b, to je z=false
```

2.11.2. Operator manje od ili jednako

-ovaj operator uspoređuje **dva** broja
-ukoliko je broj s **lijeve** strane operatora **manji od ili jednak** broju s **desne** strane, tada je rezultat **true**, inače je **false**
-**oznaka** operatora **nije** ista kao u matematici, jer ne postoji simbol na tipkovnici za njega
-zato koristimo kombinaciju znakova <= (**bez razmaka**)
-zapamtimo da znak **=** pišemo uvijek **s desne strane** ove kombinacije
-primjer:

```
bool z;
int a=23, b=17;
z=(a <= b);//pošto nije a ≤ b, to je z=false
```

2.11.3. Operator veće od

-ovaj operator uspoređuje **dva** broja
-ukoliko je broj s **lijeve** strane operatora **veći od** broja s **desne** strane, tada je rezultat **true**, inače je **false**
-**oznaka** operatora je ista kao u matematici, tj. >
-primjer:

```
bool z;
int a=23, b=17;
z=(a > b);//pošto je a > b, to je z=true
```

2.11.4. Operator veće od ili jednako

-ovaj operator uspoređuje **dva** broja
-ukoliko je broj s **lijeve** strane operatora **veći od ili jednak** broju s **desne** strane, tada je rezultat **true**, inače je **false**
-**oznaka** operatora nije ista kao u matematici, jer ne postoji simbol na tipkovnici za njega
-zato koristimo kombinaciju znakova >= (**bez razmaka**)
-zapamtimo da znak = pišemo uvijek s **desne** strane ove kombinacije
-primjer:

```
bool z;
int a=23, b=17;
z=(a >= b);//pošto je a ≥ b, to je z=true
```

2.11.5. Operator jednako

-ovaj operator uspoređuje da li su **dva** broja **jednaka**
-ukoliko su **jednaki**, tada je rezultat **true**, inače je **false**
-**oznaka** operatora nije ista kao u matematici, već koristimo kombinaciju == (bez razmaka)

-treba uočiti da znak = predstavlja **pridruživanje**
-primjer:

```
bool z;
int a=23, b=17;
z=(a == b);//pošto nije a = b, to je z=false
```

-kada bismo isti primjer napisali s krivom oznakom operatora jednakosti (= umjesto ==), kompajler **ne bi javio pogrešku**, ali bi **rezultat** bio **pogrešan**
-primjer:

```
bool z;
int a=23, b=17;
z=(a = b);
```

-zbog zagrade prvo se napravi pridruživanje a=b i **a** postane 17
-nakon toga se ta vrijednost pridruži varijabli **z**, tj. **z** je 17

2.11.6. Operator različito od

-ovaj operator uspoređuje da li su **dva broja nejednaka**
-ukoliko su **nejednaki**, tada je rezultat **true**, inače je **false**
-**oznaka** operatora nije ista kao u matematici, već koristimo kombinaciju != (**bez razmaka**)
-primjer:

```
bool z;
int a=23, b=17;
z=(a != b);//pošto je a ≠ b, to je z=true
```

2.12. Vježbe - operatori uspoređivanja

-u ovoj cjelini uvježbavamo upotrebu operatora uspoređivanja i njihovo kombiniranje s **logičkim operatorima** u jednostavne izraze
-primjer: Provjerite da li je cjelobrojna varijabla **a** veća ili jednaka 13. Rezultat ispišite na ekranu. Početna vrijednost za **a** je 2.
-rješenje:

```
bool z;
int a=2;
z=(a >= 13);//pošto nije 2≥13, to je z=false
cout<<z<<endl;
```

-primjer: Provjerite da li je cjelobrojna varijabla **a** manja ili jednaka 255. Rezultat provjere negirajte i ispišite na ekranu. Početna vrijednost za **a** je 142.
-rješenje:

```
bool z;
int a=142;
z=(a <= 255);//pošto je 142≤255, to je z=true
z=!z;//negiranje daje rezultat false
cout<<z<<endl;
```

-primjer: Provjerite da li je cjelobrojna varijabla **a** u opsegu od 13 do 255 (uključujući granice). Početna vrijednost za **a** je 142.
-rješenje:
-ukoliko moramo provjeriti da li je neka varijabla **u određenom intervalu**, to radimo tako da provjeravamo da li je **varijabla veća ili jednaka od donje granice**, te da li je **manja ili jednaka od gornje granice**
-ukoliko je **jedno i drugo** istina, tada se varijabla nalazi u zadanom intervalu
-budući da provjeravamo da li je istinita **i jedna i druga** tvrdnja, to za njihovo povezivanje koristimo operaciju **logičko I**
-dakle, zapamtimo: ako neka varijabla mora biti **u nekom intervalu**, provjerimo da li je **veća ili jednaka od donje granice i manja ili jednaka od gornje granice**, te ta dva uvjeta povežimo operacijom logičko **I**

-za one, kojima je matematika bliža, možemo reći da smo napravili **presjek dva intervala na brojevnom pravcu** i dobili zadani interval
-slijede naredbe za realizaciju objašnjenog zadatka:

```
int a=142;
bool x, y, z;
x=(a <= 255);//pošto je 142≤255, to je x=true
y=(a>=13);//pošto je 142≥13, to je y=true
z=(x && y);//x i y su true pa je i z true
cout<<z<<endl;
```

-primjer: Provjerite da li je cjelobrojna varijabla **a** izvan opsega od 13 do 255 (uključujući granice). Početna vrijednost za **a** je 142.
-rješenje:
-na temelju prijašnjeg zadatka, najjednostavnije je prvo provjeriti da li je varijabla **unutar** nekog intervala, a potom to **negirati**

```
int a=142;
bool x, y, z;
x=(a <= 255);//pošto je 142≤255, to je x=true
y=(a>=13);//pošto je 142≥13, to je y=true
z=(x && y);//x i y su true pa je i z true
z=!z;//z je false, tj. 142 nije izvan opsega od 13 do 255
cout<<z<<endl;
```

2.13. Vježbe - operatori

-u ovoj nastavnoj jedinici cilj je uvježbati **upotrebu** češće korištenih **operatora** na **složenijim zadacima**
-u prvom dijelu usmjerit ćemo se na zadatke koji zahtijevaju upotrebu **aritmetičkih operatora**, dok će u drugom prevladavati zadaci u kojima se većinom koriste **logički i operatori usporedbe**
-uz svaki zadatak dano je jedno moguće **rješenje**, a po potrebi i **komentar rješenja** i **uputa** za **drugačiji način rješavanja**

2.13.1. Upotreba aritmetičkih operatora u zadacima

-primjer: Izračunajte vrijednost izraza $y=(x^3+2)(2x-1)(5x^2-1)/4$. Neka su sve varijable tipa float, a za provjeru uzmite da je **x**=1.2. U rješenju smijete u jednoj liniji provesti samo jednu aritmetičku operaciju nad dva operanda. Broj pomoćnih varijabli nije ograničen. U zadatku se ne smije koristiti funkcija pow() za potenciranje.
-rješenje:

```
#include <cstdlib>
#include <iostream>

using namespace std;

int main(int argc, char *argv[])
{
   float x=1.2, y, pomoc;//deklaracija ulazne vrijednosti, varijable za rezultat i pomoćne varijable

   y=x*x;//x^2
   pomoc=y*x;//x^3
   y=5*y;//5x^2 - tu smo s istim učinkom mogli napisati y*=5;
   y--;//5x^2-1
   pomoc=pomoc+2;//x^3+2 - tu smo s istim učinkom mogli napisati pomoc+=2;
   y=y*pomoc;//(x^3+2)(5x^2-1) - tu smo s istim učinkom mogli napisati y*=pomoc;
   y=y*0.25;//(x^3+2)(5x^2-1)/4 - tu smo s istim učinkom mogli napisati y*=0.25;
   pomoc=2*x;//2x
```

```
    pomoc--;//2x-1
    y=y*pomoc;//(x^3+2)(2x-1)(5x^2-1)/4 - tu smo s istim učinkom mogli napisati y*=pomoc;

    cout<<"Rezultat je "<<y<<endl;
    system("PAUSE");
    return EXIT_SUCCESS;
}
```

-rješenje za y=8.08976

-**analiza primjera**:

a) rješenje je dobiveno upotrebom **samo jedne pomoćne varijable**, dakle pazilo se na **uštedu memorije**

b) iz komentara programskih linija vidljivo je da su se neke naredbe mogle **kraće napisat**i, čime se ništa **ne dobiva na brzini** izvršavanja programa, ali program postaje **teže čitljiv** za početnike

c) vidimo da smo izračunanu vrijednost za x^2 iskoristili za izračun izraza $(5x^2-1)$, a potom za dobivanje x^3, čime smo **ubrzali** program

d) u dva izraza iskoristili smo operaciju **dekrementiranja** (--), čime smo malo **ubrzali** računanje u odnosu na uobičajeno oduzimanje jedinice (npr. 2x-1)

e) **umjesto dijeljenja** cijelog izraza brojem 4 uveli smo **množenje njegovom inverznom vrijednošću** (0.25) te smo time **značajno ubrzali** program

f) ukoliko bismo ipak koristili dijeljenje brojem 4, pošto je međurezultat tipa float (realan broj), nije potrebno pisati 4.0
-kada bi međurezultat bio **cijeli** broj, a htjeli bismo nakon dijeljenja imati **realan** broj, morali bismo pisati **4.0**

-primjer: Izračunajte vrijednost izraza $y = \frac{(2x^4 - 5)(7x^2 + 3)}{3x - 4}$. Neka su sve varijable tipa float, a za provjeru uzmite da je **x**=2.5. U rješenju smijete u jednoj liniji provesti proizvoljan broj aritmetičkih operacija. Broj pomoćnih varijabli nije ograničen. U zadatku se ne smije koristiti funkcija pow() za potenciranje.

-rješenje:

```
#include <cstdlib>
#include <iostream>

using namespace std;

int main(int argc, char *argv[])
{
    float x=2.5, y, pomoc;//deklaracija i inicijalizacija

    y=x*x;//x^2
    pomoc=y*y;//x^4
    y=7*y+3;//7x^2+3
    pomoc=2*pomoc-5;//2x^4-5
    y=y*pomoc;//(2x^4-5)(7x^2+3) - kraći zapis naredbe je y*=pomoc;
    pomoc=3*x-4;//3x-4
    y=y/pomoc;//(2x^4-5)(7x^2+3)/(3x-4) - kraći zapis naredbe je y/=pomoc;

    cout<<"Rezultat je "<<y<<endl;
    system("PAUSE");
    return EXIT_SUCCESS;
}
```

-rezultat je y=976.741

-**analiza primjera**:

a) rješenje je dobiveno upotrebom samo **jedne** pomoćne varijable, dakle pazilo se na **uštedu memorije**
b) iz komentara programskih linija vidljivo je da su se neke naredbe mogle **kraće napisati**, čime se ništa **ne** dobiva na **brzini** izvršavanja programa, ali program postaje **teže čitljiv** za početnike
-mogli smo probati još **kraće** zapisati neke naredbe, ali **izgubili** bismo bitno na **preglednosti**
c) vidimo da smo izračunanu vrijednost za x^2 iskoristili za izračun izraza $(7x^2+3)$, a potom za dobivanje x^4, čime smo **ubrzali** program
d) mogli smo najprije izračunati oba izraza u brojniku i izraz u nazivniku te ih na kraju povezati u rezultat, ali za to bi trebali **više pomoćnih varijabli**

-primjer: Izračunajte vrijednost izraza $y = \frac{(7x^4 \bmod 5)(4x^3+3)-2x+4}{x \bmod 4}$. Neka su sve varijable tipa int, a za provjeru uzmite da je **x**=3. U rješenju smijete u jednoj liniji provesti proizvoljan broj aritmetičkih operacija potreban za računanje izraza unutar jedne zagrade. Broj pomoćnih varijabli nije ograničen. U zadatku se ne smije koristiti funkcija pow() za potenciranje.
-rješenje:

```
#include <cstdlib>
#include <iostream>

using namespace std;

int main(int argc, char *argv[])
{
   int x=3, y, pomoc;

   y=4*x*x*x+3;//4x^3+3
   pomoc=(7*x*x*x*x*x)%5;//7x^4 mod 5
   y=y*pomoc-2*x+4;//(7x^4 mod 5)(4x^3+3)-2x+4
   y=y/(x%4);//((7x^4 mod 5)(4x^3+3)-2x+4)/(x mod 4)

   cout<<"Rezultat je "<<y<<endl;
   system("PAUSE");
   return EXIT_SUCCESS;
}
```

-rezultat je y=73
-**analiza primjera**:
a) upotrebu samo **jedne pomoćne** varijable omogućilo je pisanje izraza pomoću **više operacija** u programskoj liniji
b) time smo dobili **kraći zapis** programa, ali smo **usporili** program, jer **ne koristimo međurezultate** (x^3) koji nam omogućuju ubrzanje programa
c) za definiranje **prioriteta** koristili smo **zagrade**, mada one nisu potrebne u izrazu *pomoc=(7*x*x*x*x*x)%5;*, jer bi se i bez njih prvo provela 4 množenja, a potom ostatak dijeljenja

-primjer:
Izračunajte vrijednost sličnog izraza kao u prijašnjem zadatku ($y = \frac{(7x^4-5)(4x^3+3)-2x+4}{x \bmod 4}$). Neka su sve varijable tipa **int**, a za provjeru uzmite da je **x**=3. U rješenju smijete u jednoj liniji provesti proizvoljan broj aritmetičkih operacija. Broj pomoćnih varijabli nije ograničen. U zadatku se mora upotrijebiti funkcija pow() za potenciranje. Napomena: nije se mogao zadati isti primjer, jer funkcija **pow()** vraća **realan broj** kao **rezultat** pa se operacija **ostatka cjelobrojnog dijeljenja na takvom broju ne može primijeniti**.
-rješenje:

```
#include <cstdlib>
#include <iostream>
#include <cmath>

using namespace std;

int main(int argc, char *argv[])
{
   int x=3, y;

   y=(((7*pow(x,4.0))-5)*((4*pow(x,3.0))+3)-2*x+4)/(x%4);

   cout<<"Rezultat je "<<y<<endl;
   system("PAUSE");
   return EXIT_SUCCESS;
}
```

-rezultat je y=20793

-**analiza primjera**:

a) vidi se da je cijeli zadatak riješen **jednom** programskom linijom

b) zbog **prioriteta** aritmetičkih operacija i zbog upotrebe **funkcije pow()** morao je biti korišten **veći broj zagrada** što vodi do **nepreglednosti** i lakše mogućnosti **pogreške**

c) ukoliko nismo upotrijebili **isti broj otvorenih i zatvorenih zagrada**, kompajler javlja **pogrešku**

d) mada u brojniku nismo mogli u zagradi koristiti operaciju **mod**, jer funkcija pow() daje kao rezultat realan broj, u nazivniku smo je mogli koristiti zato što je x **cjelobrojan**

-mogući **uzroci problema** u dosadašnjim primjerima:

a) **izostavljen** znak **;** na kraju naredbe
   -u tom slučaju kompajler javlja da nedostaje znak ; na kraju linije, ali pogrešku javlja u **prvoj liniji ispod** one u kojoj smo zaboravili taj znak

b) ukoliko **ne koristimo zagrade**, a nismo poštovali **pravila prioriteta** operatora, može se dogoditi da kompajler javi **pogrešku**
   -isto tako je moguće da kompajler **ne javi pogrešku**, jer je izraz **dobro napisan**
   -to je lošiji slučaj, jer nismo upozoreni na pogrešku, a program **radi pogrešno**

c) može se dogoditi da u programu zabunom **umjesto** operatora **%** upotrijebimo operator **/**
   -ovisno o vrsti varijabli kompajler može, ali i ne mora javiti pogrešku

d) kod **skraćivanja računanja** na nekoliko programskih linija poželjno je upotrijebiti **veći broj zagrada**, ako nam to **olakšava izračun**

e) ukoliko nismo sigurni u **prioritete operatora**, upotrijebimo **zagrade** oko dijela izraza koji želimo prije izračunati
   -**koliko** je **zagrada otvoreno**, toliko ih treba **zatvoriti**, inače kompajler javlja **pogrešku**

f) može se dogoditi da neka **ulazna** vrijednost negdje u izrazu izazove **dijeljenje s 0**
   -u tom slučaju pri pokretanju program (crni prozor) **zablokira** ("Program is not responding.") pa mišem moramo **zatvoriti** njegov prozor

g) ukoliko smo **pokrenuli** neki kompajlirani program, pa smo potom napisali **novi** program, a nismo zatvorili prozor izvršavanog programa (crni prozor), ne možemo pokrenuti kompajliranje, niti druge naredbe iz izbornika **Naredbe**
   -dakle, moramo **prozor** pokrenutog programa **zatvoriti prije kompajliranja i pokretanja novog programa**

2.13.2. Upotreba operatora usporedbe i logičkih operatora u zadacima

-primjer: Izračunajte izraz $Z = \bar{A}B + A\bar{B}$. U rješenju smijete u jednoj liniji provesti samo jednu operaciju nad jednim ili dva operanda. Broj pomoćnih varijabli nije ograničen. Početne vrijednosti su A=1 (true) i B=0 (false), sve varijable su logičkog tipa (bool).

-rješenje:

```
#include <cstdlib>
#include <iostream>

using namespace std;

int main(int argc, char *argv[])
{
   bool a=1, b=0, z, pomoc;
   z=!a;//negacija varijable a
   pomoc=!b;//negacija varijable b
   z=z&&b;//negirano a logičko I s varijablom b
   pomoc=a&&pomoc;//negirano b logičko I s varijablom a
   z=z||pomoc;//ILI operacija među oba međurezultata

   cout<<"Rezultat je "<<z<<endl;
   system("PAUSE");
   return EXIT_SUCCESS;
}
```

-rezultat je z=1 (true)

-**analiza primjera**:

a) upotrijebili smo varijable za **izračun negiranih vrijednosti** obje varijable

b) potom smo izračunali **oba izraza s I** operacijom i na kraju ih **povezali ILI** operacijom

-primjer: Izračunajte izraz $Z = (AB + \bar{A} + B)A$. U rješenju smijete u jednoj liniji provesti samo jednu operaciju nad jednim ili dva operanda. Broj pomoćnih varijabli nije ograničen. Početne vrijednosti su A=1 (true) i B=1 (true), a sve varijable su logičkog tipa (bool).

-rješenje:

```
#include <cstdlib>
#include <iostream>

using namespace std;

int main(int argc, char *argv[])
{
   bool a=1, b=1, z, pomoc;
   z=!a;//negacija od a
   pomoc=a&&b;//I operacija između a i b
   z=z||pomoc;//ILI operacija između negiranog a i ab
   z=z||b;//izračunan cijeli izraz u zagradi
   z=z&&a;//gotov je cijeli zadatak

cout<<"Rezultat je "<<z<<endl;
system("PAUSE");
return EXIT_SUCCESS;
}
```

-rezultat je z=1 (true)

-**analiza primjera**:

a) **prvo** moramo odrediti **izraz u zagradi**

b) u izrazu u **zagradi** prvo računamo **međurezultate** (negacija od a i I operacija između a i b)

c) oba međurezultata povežemo **ILI** operacijom i taj međurezultat povežemo **ILI** funkcijom s varijablom b

d) na kraju rezultat cijele zagrade povežemo **I** operacijom s varijablom a

-primjer: Izračunajte isti izraz kao u prijašnjem primjeru ($Z = (AB + \bar{A} + B)A$). U rješenju smijete u jednoj liniji provesti proizvoljan broj logičkih operacija. Broj pomoćnih varijabli nije ograničen. Početne vrijednosti su A=1 (true) i B=1 (true), sve varijable su logičkog tipa (bool).
-rješenje:

```
#include <cstdlib>
#include <iostream>

using namespace std;

int main(int argc, char *argv[])
{
   bool a=1, b=1, z, pomoc;
   z=((a&&b)||(!a)||b)&&a;

   cout<<"Rezultat je "<<z<<endl;
   system("PAUSE");
   return EXIT_SUCCESS;
}
```

-razumljivo, rezultat je i dalje z=1 (true)
-**analiza primjera**:
a) cijeli zadatak riješili smo u **jednoj programskoj liniji**
b) mada nismo trebali koristiti sve **zagrade** u izrazu, radi **preglednosti** ih je poželjno koristiti
-ukoliko nismo sigurni u **prioritet** operatora, koristimo **zagrade**

-primjer: Provjerite da li je cjelobrojna varijabla x unutar barem jednog od opsega vrijednosti: -2 do 3 i 79 do 388 (uključujući granice). Početna vrijednost varijable x je 80.
-rješenje:

```
#include <cstdlib>
#include <iostream>

using namespace std;

int main(int argc, char *argv[])
{
   int x=80;
   bool a, b, z;
   a=(x<=3)&&(x>=-2);//provjera da li je x u opsegu od -2 do 3
   b=(x<=388)&&(x>=79);//provjera da li je x u opsegu od 79 do 388
   z=a||b;//koristimo ILI operaciju

   cout<<"Rezultat je "<<z<<endl;
   system("PAUSE");
   return EXIT_SUCCESS;
}
```

-rezultat je z=1 (true), jer je broj 80 u opsegu od 79 do 388
-**analiza primjera**:
a) **opseg vrijednosti** provjerava se na način da tražimo da li je neka varijabla **veća ili jednaka od donje granice** i **istovremeno** **manja ili jednaka od gornje granice**
-matematički je to **presjek dva intervala**
-budući da obje provjere moraju biti **istovremeno** zadovoljene, za njihovo povezivanje koristimo **I** operaciju

b) u zadatku tražimo da broj bude **barem u jednom** od zadanih opsega vrijednosti, a to postižemo upotrebom **ILI** operacije na oba međurezultata

-primjer: Provjerite da li je istodobno cjelobrojna varijabla x pozitivna, izvan opsega vrijednosti od 5 do 332 i u opsegu vrijednosti od 500 do 1000 (ne uključujući granice u oba opsega). Početna vrijednost varijable x je 600.
-rješenje:

```
#include <cstdlib>
#include <iostream>

using namespace std;

int main(int argc, char *argv[])
{
   int x=600;
   bool a, b, c, z;
   a=(x>0);//provjera da li je x pozitivan (ne računajući 0 kao pozitivan broj)
   b=(x<332)&&(x>5);//provjera da li je x u opsegu od 5 do 332
   c=(x<1000)&&(x>500);//provjera da li je x u opsegu od 500 do 1000
   z=(a&&(!b)&&c);//koristimo I operaciju

   cout<<"Rezultat je "<<z<<endl;
   system("PAUSE");
   return EXIT_SUCCESS;
}
```

-rezultat je 1 (true), jer je x pozitivan, izvan intervala (5, 332) i u intervalu (500, 1000)
-**analiza primjera**:

a) kod provjere da li je x **u zadanom opsegu** koristimo operator > i <, jer **granice opsega nisu uključene** u provjeru (vidi tekst zadatka)
b) radi **preglednosti** **zasebno** vršimo **provjeru svakog intervala** (bilo da on treba ili ne treba biti zadovoljen), pri čemu rezultat pamtimo u **zasebnim varijablama**
c) na kraju uvjete povežemo **I** operacijom (jer svi moraju biti zadovoljeni **istovremeno**), pri čemu one koji **ne smiju** biti zadovoljeni **negiramo** (!b)

2.14. Zadaci za ocjenjivanje vježbi s operatorima

-u ovoj nastavnoj jedinici **ocjenjuju** se **dvije** vježbe:

a) **zadaci s aritmetičkim operatorima**
b) **zadaci s relacijskim i logičkim operatorima**

-svaka grupa (**A** i **B**) dobiva **dva** zadatka koja mora riješiti na računalu
-svaki zadatak sastoji se od **četiri dijela** od kojih se svaki točno riješen boduje s **jednim** bodom
-na kraju se **svi dijelovi povezuju** u završni **izraz** i to se boduje s **dodatnim** bodom
-**ocjena** je jednaka **broju bodova**, osim za 0 bodova (ocjena 1)

-zadaci grupe **A**:
1.) Izračunajte 4 dijela idućeg izraza u zagradama i pomoću njih riješite cijeli izraz. Sve varijable su tipa int. U jednoj programskoj liniji koristite samo jednu aritmetičku operaciju. Na raspolaganju je proizvoljan broj pomoćnih varijabli. Početna vrijednost za x=5. Izraz je dan ovom formulom:

$$y = \frac{(2x^2 + 1)(x^4 - 5)}{(x^3 + 8)(5x - 6)} + 1$$

-rezultat je y=639

2.) Logičkim i operatorima usporedbe odredite 4 međurezultata i to da varijabla x bude:
a) pozitivna

b) manja od 3000
c) izvan opsega od 100 do 200 (uključujući granice)
d) u opsegu od 300 do 600 (ne uključujući granice).

-nakon toga odredite y koji mora biti istina (true), ako su svi uvjeti od a) do d) istovremeno zadovoljeni uz početnu vrijednost x=365
-rješenje: y=1 (true)

-zadaci grupe **B**:
1.) Izračunajte 4 dijela idućeg izraza u zagradama i pomoću njih riješite cijeli izraz. Sve varijable su tipa int. U jednoj programskoj liniji koristite samo jednu aritmetičku operaciju. Na raspolaganju je proizvoljan broj pomoćnih varijabli. Početna vrijednost za x=4. Izraz je dan ovom formulom:

$$y = \frac{(3x^4 - 1)(x^2 + 4)}{(x^3 + 5)(2x - 3)} + 2$$

-rezultat je y=46

2.) Logičkim i operatorima usporedbe odredite 4 međurezultata i to da varijabla x bude:
a) pozitivna
b) manja od 4000
c) izvan opsega od 200 do 300 (uključujući granice)
d) u opsegu od 500 do 800 (ne uključujući granice).

-nakon toga odredite y koji mora biti istina (true), ako su svi uvjeti od a) do d) istovremeno zadovoljeni uz početnu vrijednost x=625
-rješenje: y=1 (true)

2.15. Pismena provjera znanja

-pismena provjera znanja obuhvaća početni dio predavanja iz programiranja u C++ programskom jeziku (do funkcija)
-sastoji se od čisto **teoretskog** dijela i nekoliko **primjera** u kojima se treba naći pogrešku u programu, odnosno **napisati** odgovarajući dio programa
-pismena provjera znanja različita je za **A** i **B** grupu, ali se nastoji da zadaci težinom i vrstom budu čim ujednačeniji u obje grupe
-svaki zadatak boduje se **jednim** bodom
-prag za pozitivnu ocjenu je tipično **40%**

2.16. Matematičke funkcije

-do sada smo naučili koje matematičke **operacije** u C++ programskom jeziku možemo postići upotrebom **operatora** (zbrajanje, oduzimanje, množenje, dijeljenje, ostatak cjelobrojnog dijeljenja)
-zbog potrebe za **potenciranjem** obradili smo i funkciju **pow()** koja nam omogućuje **potenciranje realnog broja** (float) **realnim eksponentom** (float)
-za sve druge matematičke funkcije možemo napisati **vlastite** funkcije ili ćemo koristiti neku od **ugrađenih** (**postojećih**) u jezik C++
-da se olakša rad s **ugrađenim matematičkim funkcijama** dobro je slijediti ova **pravila**:

a) sve te funkcije definirane su bilo u **cstdlib**, bilo u **cmath** dijelu knjižnice standardnih funkcija
-da ne bi morali pamtiti u kojoj je neka od njih definirana, najlakše je u svaki program u kojem koristimo neku od matematičkih funkcija **ubaciti oba** ova dijela standardne biblioteke funkcija (**cstdlib** se **skoro uvijek ubacuje** u svaki program)
-dakle, **prije main()** funkcije treba dodati ove dvije naredbe za ubacivanje dijela standardne biblioteke funkcija:

```
#include <cstdlib>
#include <cmath>
```

b) **većina** funkcija definirana je i za **cjelobrojne** (int) i za **realne** (float) vrijednosti
-funkcije većinom **vraćaju realni rezultat**

-iznimke od ovog pravila bit će posebno navedene, inače vrijedi ovo pravilo

c) **x** u sintaksi (načinu pisanja) funkcija označava da tu može stajati **konstanta vrijednost** (npr. 3.27) ili **ime varijable** (npr. broj1)

-češće korištene **ugrađene matematičke funkcije** su:

a) **abs(x) ili fabs(x)**

-svejedno je koje od ova **dva imena** koristimo

-funkcija izračunava **apsolutnu vrijednost broja** (udaljenost broja od ishodišta)

-primjer:

```
int a;
float b;
a=-12;
b=-324.3445;
a=abs(a);
b=fabs(b);
cout<<a<<endl;
cout<<b<<endl;//rezultat je ispis broja 12 i 324.3445
```

b) **ceil(x)**

-funkcija **zaokružuje** broj x na **najbliži veći cijeli broj**

-dakle, rezultat je **cijeli broj**

-treba paziti da je kod **negativnih** vrijednosti **veći** broj onaj s **manjom apsolutnom vrijednošću** (npr. -345 je manji od -340)

-primjer:

```
float a;
float b;
a=123.6;
b=-324.3445;
a=ceil(a);
b=ceil(b);
cout<<a<<endl;//rezultat je 124 (veći broj u odnosu prema 123.6)
cout<<b<<endl;//rezultat je -324 (veći broj u odnosu prema -324.3445)
```

c) **floor(x)**

-funkcija **zaokružuje** broj x na **najbliži manji cijeli broj**

-dakle, rezultat je **cijeli broj**

-treba paziti da je kod **negativnih** vrijednosti **manji** broj onaj s **većom apsolutnom vrijednošću** (npr. -315 je veći od -340)

-primjer:

```
float a;
float b;
a=123.6;
b=-324.3445;
a=floor(a);
b=floor(b);
cout<<a<<endl;//rezultat je 123 (manji broj u odnosu prema 123.6)
cout<<b<<endl;//rezultat je -325 (manji broj u odnosu prema -324.3445)
```

d) **rand()**

-ova funkcija **nema** nikakvu **ulaznu** vrijednost (**argument**), a vraća **pseudoslučajan cijeli broj** u opsegu **od 0 do gornje granice** koja **ovisi o kompajleru** za koji je funkcija napisana

-napomena: pojam **pseudoslučajan** označava da to nije slučajan broj u matematičkom smislu, ali za našu razinu upotrebe možemo ga zvati **slučajnim**

-primjer:

```
int a;
a=rand();
```

```
cout<<a<<endl;//rezultat je npr. 43
```

-**napomena**: pri svakoj **idućoj upotrebi** ove funkcije u **istom** programu **mijenja** se generirani pseudoslučajan broj

-međutim, pri **prvom pokretanju** ova funkcija uvijek daje **istu početnu vrijednost**

-to je za neke primjene (npr. igre) **nepraktično** i može se promijeniti, ali način postizanja toga nadilazi razinu programiranja koju mi obrađujemo

e) **pow(baza, eksponent)**

-ova funkcija određuje iznos **potencije** oblika

$$baza^{eksponent}$$

-pritom su **baza**, **eksponent** i **rezultat** **realni brojevi**, tj. tipa **float**

-pritom je bitno napomenuti da **baza mora biti pozitivna**, inače bi rezultat trebao biti **kompleksan broj**, a funkcija pow() nije namjenjena za rad s kompleksnim brojevima

-zato je najbolje tamo gdje se nakon izračunavanja nekog izraza dobije **negativna vrijednost baze**, upotrijebiti **apsolutnu vrijednost baze**, a tek potom potenciranje

-problem se može pojaviti tamo gdje tražimo **neparni korijen iz negativne baze**

-matematički je to negativna vrijednost korijena iz apsolutne vrijednosti baze (npr. $\sqrt[3]{-5} = -\sqrt[3]{5}$)

-problem je što mi koristimo **realni broj** kao **eksponent** potencije, a on zbog **konačnog broja znamenki** u prikazu broja na računalu nije jednak u potpunosti točnom iznosu eksponenta potencije u obliku razlomka

-zbog toga pri izračunu u prijašnjem primjeru ne dobivamo onaj rezultat koji smo očekivali, već pri ispisu se umjesto vrijednosti broja ispiše riječ **nan** koja označava da je rezultat **nemoguće izračunati**, jer je ili rezultat **izvan opsega** vrijednosti određenog brojčanog tipa, ili **funkcija nije definirana** za navedenu ulaznu vrijednost

-u slučaju da imamo zadatak kao u prijašnjem primjeru, moramo ručno **preurediti** zadatak u oblik koji je **matematički korektan**, a **potencira** se **pozitivna** baza (u prijašnjem primjeru treba napisati $\sqrt[3]{-5} = -\sqrt[3]{5}$ i potom to riješiti potenciranjem - tu je to y=-pow(5, 0.3333333);)

-primjer:

```
float a=2;
float b=3;
float rezultat;
rezultat=pow(a, b);
cout<<rezultat<<endl;//rezultat je 8.0
```

f) **sqrt(x)**

-ova funkcija računa **kvadratni korijen iz pozitivnog argumenta** (broja)

-istu stvar mogli bi izračunati i pomoću funkcije pow(x, 0.5), jer je potenciranje s 0.5 isto što i računanje kvadratnog korijena

-međutim, funkcija **sqrt()** je bitno **brža** od funkcije pow() pa zato je uobičajeno koristiti sqrt() za korijenovanje

-primjer:

```
float b=3;
float rezultat;
rezultat=sqrt(b);
cout<<rezultat<<endl;//rezultat je 1.73205
```

g) **exp(x)**

-ova funkcija računa potenciju e^x

-istu stvar mogli bismo izračunati upotrebom funkcije pow(2.71828, x), ali je ona bitno **sporija** od funkcije exp(x)

-primjer:

```
float b=3;
float rezultat;
rezultat=exp(b);
cout<<rezultat<<endl;//rezultat je 20.0855
```

h) **log(x), log10(x)**

-funkcija **log(x)** računa **prirodni logaritam** (po bazi **e**) od x, tj. **ln(x)**

-funkcija **log10(x)** računa **dekadski logaritam** (po bazi **10**) od x, tj. **log(x)**

-treba paziti da se **ne pobrkaju** ove dvije funkcije, jer log u matematici i kod programiranja ne označavaju isti logaritam

-primjer:

```
float b=2975;
float rezultat;
rezultat=log(b);
cout<<rezultat<<endl;//rezultat je 7.998
rezultat=log10(b);
cout<<rezultat<<endl;//rezultat je 3.47349
```

i) **sin(x)**

-funkcija računa **sinus** od argumenta x koji je zadan u **radijanima** (**stupnjeve u radijane** pretvaramo tako da broj stupnjeva pomnožimo s **π** i potom **podijelimo sa 180** ili kraće, tako da **broj stupnjeva podijelimo s približno 57.296**)

-**rezultat** je **realan broj u opsegu od -1 do 1**

-primjer:

```
float b=60;
b=60/57.296;//pretvaramo stupnjeve u radijane
float rezultat;
rezultat=sin(b);

cout<<rezultat<<endl;//rezultat je 0.866023, tj. √3/2
```

j) **cos(x)**

-funkcija računa **kosinus** od argumenta x koji je zadan u **radijanima** (stupnjeve u radijane pretvaramo tako da broj stupnjeva pomnožimo s π i potom podijelimo sa 180 ili kraće, tako da broj stupnjeva podijelimo s približno 57.296)

-**rezultat** je **realan broj u opsegu od -1 do 1**

-primjer:

```
float b=60;
b=60/57.296;//pretvaramo stupnjeve u radijane
float rezultat;
rezultat=cos(b);
cout<<rezultat<<endl;//rezultat je 0.5
```

k) **tan(x)**

-funkcija računa **tangens** od argumenta x koji je zadan u **radijanima** (stupnjeve u radijane pretvaramo tako da broj stupnjeva pomnožimo s π i potom podijelimo sa 180 ili kraće, tako da broj stupnjeva podijelimo s približno 57.296)

-primjer:

```
float b=60;
b=60/57.296;//pretvaramo stupnjeve u radijane
float rezultat;
rezultat=tan(b);
cout<<rezultat<<endl;//rezultat je približno 1.73203, tj. √3
```

l) **asin(x)**

-funkcija računa **arkus sinus** od argumenta x, tj. **inverzni sinus** od x

-x je u opsegu od **-1 do 1**, a **rezultat** je **kut u radijanima** u opsegu od $-\frac{\pi}{2}$ **do**

-primjer:

```
float b=0.5;
float rezultat;
rezultat=asin(b);
rezultat=57.296*rezultat;//pretvaramo radijane u stupnjeve
cout<<rezultat<<endl;//rezultat je približno 30°
```

m) **acos(x)**

-funkcija računa **arkus kosinus** od argumenta x, tj. **inverzni kosinus** od x

-x je u opsegu od **-1 do 1**, a **rezultat** **kut u radijanima** u opsegu od **0 do π**

-primjer:

```
float b=0.4;
float rezultat;
rezultat=acos(b);
rezultat=57.296*rezultat; //pretvaramo radijane u stupnjeve
                          //rezultat je približno 66.4221°
cout<<rezultat<<endl;
```

n) **atan(x)**

-funkcija računa **arkus tangens** od argumenta x, tj. **inverzni tangens** od x

-**rezultat** je **kut u radijanima** u opsegu od $-\frac{\pi}{2}$ do

-primjer:

```
float b=1;
float rezultat;
rezultat=atan(b);
rezultat=57.296*rezultat;//pretvaramo radijane u stupnjeve
cout<<rezultat<<endl;//rezultat je približno  45°
```

2.17. Matematičke funkcije - vježbanje

-cilj ove lekcije je uvježbavanje korištenja dijela **matematičkih funkcija** (bez logaritama i inverznih trigonometrijskih funkcija, jer još nisu obrađene iz matematike)

-radi preglednosti i lakšeg praćenja rješavanja zadataka svi primjeri rješavaju se pomoću većeg broja **pomoćnih varijabli**, a cijeli izraz se rješava **po dijelovima** u zasebnim programskim linijama

-sve varijable koje koristimo u vježbama su tipa **float** ili **double**, osim ako je drukčije navedeno

-napomena: za **potenciranje** cijelim brojem koristit ćemo funkciju **pow()**, a ne množenje

-primjer: za x=2.3 odredite vrijednost ovog izraza: $y=\sqrt[6]{1+x^2+2\sqrt[5]{\frac{x^3}{1+\sqrt{x}}}}$

-rješenje: prvi korak je da izraz napišemo **u obliku potencija**, osim za **kvadratni korijen** za koji postoji gotova funkcija **sqrt()**

-time dobivamo: $y=\left(1+x^2+2\left(\frac{x^{\frac{3}{5}}}{(1+\sqrt{x})^{\frac{1}{5}}}\right)\right)^{\frac{1}{6}}$

-da bismo mogli upotrijebiti funkciju **pow()** potrebno je sve **razlomke** zamijeniti **realnim brojem**

-time dobivamo: $y=\left(1+x^2+2\left(\frac{x^{0.6}}{(1+\sqrt{x})^{0.2}}\right)\right)^{0.1666667}$

-glavni dio programa za određivanje toga izraza je:

```
int main(int argc, char *argv[])
{
```

```
    double x, y, a, b, c, d;
    x=2.3;
    a=1+pow(x, 2.0);
    b=pow(x, 0.6);
    c=1+sqrt(x);
    d=pow(c, 0.2);
    d=b/d;
    y=a+2*d;
    y=pow(y, 0.1666667);
    cout<<"Rezultat je "<<y<<endl;
    system("PAUSE");
    return EXIT_SUCCESS;
}
```

-rezultat za y je 1.44308

-analiza rješenja: vidi se da je primjer rješavan postupno, počevši od zbroja u zagradi (, potom se računaju potencije koje omogućuju izračunavanje razlomka u zagradi ($x^{\uparrow}0.6/(1+\sqrt{x})^{\uparrow}0.2$) , zatim se riješi cijela zagrada i to se na kraju potencira

-primjer: Za x=2.25 odredite vrijednost ovog izraza: $y = \frac{|\sqrt[3]{x + 2\sin 2x}|}{2^x - e^{x-1}}$

-analiza rješenja:prilagodimo izraz upotrebi postojećih matematičkih funkcija uklanjanjem trećeg korijena i njegovom zamjenom razlomkom, odnosno decimalnom vrijednošću

-dobivamo: $y = \frac{|(x + 2\sin[2x)]^{0.3333333}|}{2^x - e^{x-1}}$

-sada možemo napisati program za rješenje tog izraza:

```
int main(int argc, char *argv[])
{
    double x=2.25, y, a, b, c, d;
    a=x+2*sin(2*x);
    b=pow(a,0.3333333);
    b=abs(b);
    c=pow(2.0,x);
    d=exp(x-1);
    d=c-d;
    y=b/d;
    cout<<"Rezultat je "<<y<<endl;
    system("PAUSE");
    return EXIT_SUCCESS;
}
```

-rješenje: y=0.525587

-analiza rješenja: u primjeru je vidljivo da u jednoj liniji provodimo sve operacije i funkcije koje čine neku **zaokruženu cjelinu** (mogli bismo i veći dio izraza napisati u jednoj programskoj liniji, ali uz gubitak preglednosti)

-**napomena**: ukoliko bi izraz **pod trećim korijenom** bio **negativan** tada bismo dobili poruku **nan** umjesto konkretne vrijednosti pri ispisu rezultata

-ta poruka je znak da rezultat **nije u dozvoljenom opsegu** ili da se radi o **nedozvoljenoj ulaznoj vrijednosti u funkciji**

-primjer: Za x=0.32 odredite vrijednost ovog izraza: $y = \sin\left(\frac{2x-1}{3}\right) tan^2\left(1 - \frac{x}{2}\right) - \frac{2^{3x}}{|x-1|}$

-rješenje:

```
int main(int argc, char *argv[])
{
   double x=2.25, y, a, b, c, d;
   a=(2*x-1)/3;
   a=sin(a);
   b=1-(x/2);
   b=tan(b);
   b=pow(b,2.0);
   b=a*b;
   c=3*x;
   c=pow(2,c);
   d=fabs(x-1);
   d=c/d;
   y=b-d;
   cout<<"Rezultat je "<<y<<endl;
   system("PAUSE");
   return EXIT_SUCCESS;
}
```

-rezultat je y=-86.0933

-primjer:Generirajte tri puta za redom slučajan broj u opsegu od 0 do 100.

-rješenje:

```
int main(int argc, char *argv[])
{
   int y, x;
   x=rand();
   y=x%101;
   cout<<"Rezultat je "<<y<<endl;
   x=rand();
   y=x%101;
   cout<<"Rezultat je "<<y<<endl;
   x=rand();
   y=x%101;
   cout<<"Rezultat je "<<y<<endl;
   system("PAUSE");
   return EXIT_SUCCESS;
}
```

-rješenje je niz brojeva 41, 85 i 72

-analiza primjera: da bismo broj sveli u opseg od 0 do 100 najjednostavnije je izračunati **ostatak cjelobrojnog dijeljenja** sa 101 (ostatak dijeljenja sa 101 je u opsegu od 0 do 100), jer je slučajan broj generiran funkcijom rand() u opsegu od 0 do velikog broja ovisnog o kompajleru

-primijetite da se **uzastopnim pozivanjem** funkcije **rand() ne dobivaju isti brojevi**

2.18. Jednostruko uvjetno grananje

-do sada nismo u programima mogli odlučivati kako će se program odvijati, već su svi programi išli **bez grananja od početka do kraja** (**linearno**)

-da bismo napisali neki iole koristan program, moramo imati mogućnost **različitog odvijanja programa**, **ovisno** o nečemu (npr. o iznosima varijabli ili nečem sličnom)

-to nam omogućuju naredbe za **upravljanje tijekom programa** (**kontrolu tijeka programa**, engl. **program flow control**)

-najjednostavnija takva naredba je ona za realizaciju **jednostrukog uvjetnog grananja**

-tu naredbu zovemo **if** naredbom

-**način pisanja if naredbe** je:

if *(uvjet)*

```
blok naredbi 1
else
blok naredbi 2
```

-objašnjenja značenja pojedinih **oznaka**:

a) **uvjet**

-radi se o **konstanti** (npr. 2), **varijabli** (npr. x) ili **izrazu** (npr. $a > b$) koji su ili daju vrijednost **logičkog tipa** (moguće su vrijednosti **true** ili **false**)

-uvjet se **obavezno** piše **u zagradama**

b) **blok naredbi 1**, **blok naredbi 2**

-**blok naredbi** predstavlja **jednu ili više naredbi**

-**svaka naredba završava** znakom **;** (npr. *x=a+5;*)

-ukoliko se koristi **više od jedne naredbe u bloku naredbi**, tada je obavezno **blok omeđiti vitičastim zagradama** (**početak bloka** je znak **{**, a **kraj** znak **}**), inače je dovoljno naredbu završiti znakom **;**

-primjer bloka naredbi:

```
{
a=a+3;
b=f-a;
}
```

-objašnjenje **funkcioniranja if** naredbe:

a) ako je vrijednost za **(uvjet)** jednaka **true** (ili **cijeli broj različit od 0**) tada se izvršavaju **sve naredbe iz prvog bloka naredbi**, a **sve se naredbe iz drugog bloka preskaču** te time **if naredba završava**

b) ukoliko je vrijednost za **(uvjet)** jednaka **false** (ili **0**) tada se **preskaču sve naredbe iz prvog bloka naredbi**, a **redom izvršavaju sve naredbe iz drugog bloka naredbi**

c) upotreba ključne riječi **else** i **drugog bloka naredbi nije obavezna**

d) bilo koji **blok naredbi** (ili **oba**) možemo u naredbi **izostaviti**

e) **uvjet** se **obavezno** piše **unutar okruglih zagrada**

f) najčešće se za **uvjete** koriste **operatori usporedbe** (npr. >) koje možemo povezati u **složene izraze** pomoću **logičkih operatora** (npr. &&)

g) za potrebe pretvaranja **algoritma** u program **if** prevodimo kao **ako**, a **else** kao **inače**

h) radi **preglednosti** potrebno je **uvlačiti blokove naredbi za nekoliko znakova**

-jednostavan primjer upotrebe if naredbe s **jednom naredbom** u bloku naredbi: Ako je **a<b** tada povećaj **a** za 2, inače **b** udvostruči.

h) radi **preglednosti** potrebno je **uvlačiti blokove naredbi za nekoliko znakova**

-rješenje:

```
if (a<b)
  a=a+2;
else
  b=2*b;
```

-primjer upotrebe **više naredbi** u svakom bloku: Ako je **a** pozitivan tada ga povećaj za 4 i ispiši njezinu vrijednost, inače umanji **a** za 2 i ispiši njezinu vrijednost.

-rješenje:

```
if (a>0)
{
 a=a+4;
 cout<<a<<endl;
}
else
{
 a=a-2;
 cout<<a<<endl;
}
```

2.19. Jednostruko uvjetno grananje - vježbanje

-cilj ove nastavne jedinice je uvježbati upotrebu **jednostrukog grananja** na složenijim primjerima upotrebom **složenijih uvjeta** u if naredbi

-primjer: Unesite cijeli broj, odredite njegovu apsolutnu vrijednost i, ako je broj paran, ispišite tekst „Broj je paran.“

-rješenje:

```
int main(int argc, char *argv[])
{
   short int x;
   cout<<"Unesite broj->";
   cin>>x;
   x=abs(x);
   x=x%2; //ostatak cjelobrojnog dijeljenja s 2
   if (x==0) //za slučaj da je broj paran, jer je ostatak 0
      cout<<endl<<"Broj je paran."<<endl;
   system("PAUSE");
   return EXIT_SUCCESS;
}
```

-primjer: Unesite ocjenu (pozitivan cijeli broj od 1 do 5) i njezinu vrijednost memorirajte pod imenom **ocjena**. Ukoliko je varijabla **ocjena** pozitivna, ispišite tekst „Ocjena je pozitivna.“, inače ispišite tekst „Unijeli ste nedovoljnu ocjenu“.

-rješenje:

```
int main(int argc, char *argv[])
{
   short int ocjena;
   cout<<"Unesite ocjenu->";
   cin>>ocjena;
   if (ocjena>1)
      cout<<endl<<"Ocjena je pozitivna."<<endl;
   else
      cout<<endl<<"Unijeli ste negativnu ocjenu."<<endl;
   system("PAUSE");
   return EXIT_SUCCESS;
}
```

-analiza primjera: vidljivo je da u primjeru provjeravamo da li je varijabla **x** veća od 1 (pozitivna ocjena), dok u suprotnom slučaju ispisujemo tekst o unosu negativne ocjene

-isti zadatak smo mogli riješiti na više načina, primjerice, mogli smo provjeriti da li je **x**=1, ili da li je **x**<2 i sl.

-primjer: Unesite broj **x**. Ako je **x** paran i djeljiv s 3, povećajte ga za 1 i ispišite mu vrijednost, inače ga umanjite za 1 i ispišite mu vrijednost.

-rješenje:

```
int main(int argc, char *argv[])
{
   short int x, a, b;
   cout<<"Unesite broj->";
   cin>>x;
   a=x%2; //ostatak cjelobrojnog dijeljenja s 2 - provjera djeljivosti s 2
   b=x%3; //ostatak cjelobrojnog dijeljenja s 3 - provjera djeljivosti s 3
   if ((a==0) && (b==0)) //broj je istovremeno djeljiv s 2 i 3 (I logička operacija nad uvjetima)
   {
    x++; //povećanje x za 1
    cout<<endl<<"Broj je "<<x<<endl;
```

```
    }
    else
    {
      x--; //umanjenje x za 1
      cout<<endl<<"Broj je "<<x<<endl;
    }
    system("PAUSE");
    return EXIT_SUCCESS;
}
```

-analiza primjera: provjeravamo da li je broj djeljiv s 2 i da li je djeljiv s 3 (pomoću ostatka dijeljenja)
-uvjete za djeljivost s 2 i s 3 povežemo **I** logičkom operacijom, jer tražimo da broj bude djeljiv s oba broja
-primjer: Unesite broj. Ako je paran i u opsegu od 23 do 456 (uključujući granice), ispišite mu vrijednost, inače ispišite poruku „Vrijednost broja ne zadovoljava.“
-rješenje:

```
int main(int argc, char *argv[])
{
   short int x, a;
   cout<<"Unesite broj->";
   cin>>x;
   a=x%2; //ostatak cjelobrojnog dijeljenja s 2 - provjera djeljivosti s 2
   if ((a==0) && (x>=23) && (x<=456)) //složeni uvjet
     cout<<endl<<"Broj je "<<x<<endl;
   else
     cout<<endl<<"Vrijednost broja ne zadovoljava."<<endl;
   system("PAUSE");
   return EXIT_SUCCESS;
}
```

-analiza rješenja: parnost provjeravamo tehnikom iz prijašnjih primjera (ostatak dijeljenja s 2), dok uvjet da je broj u zadanom intervalu ostvarujemo provjerom da li je broj veći ili jednak od donje granice i istovremeno (**I** operacija) manji ili jednak od gornje granice
-primjer: Unesite broj **x**. Ukoliko je paran i u opsegu od 32 do 56 (uključujući granice) ili ako je paran i u opsegu od 86 do 243 (bez granica), onda ga udvostručite, inače ga umanjite za 1. U oba slučaja ispišite novu vrijednost broja.
-rješenje:

```
int main(int argc, char *argv[])
{
   short int x, a;
   cout<<"Unesite broj->";
   cin>>x;
   a=x%2; //ostatak cjelobrojnog dijeljenja s 2 - provjera djeljivosti s 2
   if ((a==0) && (((x>=32) && (x<=56)) || ((x>86) && (x<243)))) //složeni uvjet
   {
     x=x*2;
     cout<<endl<<"Broj je "<<x<<endl;
   }
   else
   {
     x--; //x umanjujemo za 1
     cout<<endl<<"Broj je "<<x<<endl;
   }
   system("PAUSE");
   return EXIT_SUCCESS;
```

```
}
```
-analiza rješenja:
-u primjeru prvo provjeravamo da li je broj paran, a potom to povežemo **I** operacijom (&&) s provjerom da li je broj barem u jednom od zadanih intervala
-pripadnost barem jednom od intervala provjeravamo upotrebom **ILI** operacije (||) nad oba intervala
-samu pripadnost jednom intervalu provjeravamo pomoću **I** operacije nad uvjetom da je broj veći, odnosno veći ili jednak od donje i manji, odnosno manji ili jednak od gornje granice intervala

2.20. Ugnježđeni If i višestruko uvjetno grananje

-u prijašnjim lekcijama naučili smo osnovnu upotrebu if naredbe
-**if** naredba može **proširiti** područje svoje primjene uvođenjem:
a) **ugnježđivanja** (engl. **nesting**)
b) **višestrukog uvjetnog grananja** (engl. **multiple choice if**)

2.20.1. Ugnježđeni If

-**ugnježđena if** naredba (engl. **nested if**) sastoji se od **jedne ili više if naredbi unutar drugih if naredbi**
-**način pisanja ugnježđenih if** naredbi:

```
if (uvjet1)
{
   naredbe1;
   if (uvjet2)
      {
       naredbe2;
       .
       .
       .
           if (uvjetN)
               {
               naredbeN;
}      } .... }
```

-objašnjenje načina pisanja **ugnježđenih if** naredbi:
a) **uvjet1** do **uvjetN** predstavlja **jednostavne ili složene uvjete**
b) koliko je **vitičastih zagrada otvoreno**, toliko ih mora biti **zatvoreno**
c) svaki **if blok unutar drugog if bloka uvlačimo** za nekoliko znakova da povećamo **preglednost**
d) ovaj način pisanja ima smisla, ako **iza svake provjere uvjeta** izvodimo **bar neku drugu naredbu** prije nailaska na **iduću razinu if naredbe** (inače bismo mogli uvjete više takvih **if**-ova staviti u **jedan if**)
-primjer:

```
if (a<2)
  {
    if (a>=1)
       {
        a++;
        cout<<a<<endl;
       }
  }
```

-isti primjer možemo kraće riješiti (jer iza prve if naredbe nema nikakvih naredbi do druge if naredbe) ovako:

```
if ((a<2) && (a>=1))
  {
   a++;
   cout<<a<<endl;
  }
```

e) **izvođenje ugniježđenih if naredbi** odvija se na ovaj način:

1.) ukoliko je **uvjet** u nekoj **if** naredbi **istinit**, tada se **izvršavaju naredbe iza nje**, te dolazimo do **iduće if naredbe**

2.) ukoliko je **uvjet** neke **if** naredbe **neistinit**, onda je **kraj** cijelog bloka **idućih if naredbi**, **ne izvršavaju** se nikakve **naredbe**, već se prelazi na **prvu naredbu iza ugniježđenih if** naredbi

-primjer: Unesite cijeli broj **x**. Ako je **x** paran, ispišite tekst „Broj je paran.“, inače je kraj programa. U slučaju da je **x** djeljiv s 3, dodatno ispišite tekst „Broj je višekratnik od 6.“.

-rješenje:

```
int main(int argc, char *argv[])
{
   short int x, a, b;
   cout<<"Unesite broj->";
   cin>>x;
   a=x%2; //ostatak cjelobrojnog dijeljenja s 2 - provjera djeljivosti s 2
   b=x%3; //ostatak cjelobrojnog dijeljenja s 3 - provjera djeljivosti s 3
   if (a==0)
     {
      cout<<endl<<"Broj je paran."<<endl;
      if (b==0)
        {
         cout<<endl<<"Broj je visekratnik od 6."<<endl;
        }
     }
   system("PAUSE");
   return EXIT_SUCCESS;
}
```

-analiza rješenja: prvo odredimo ostatke dijeljenja s 2 i 3 i zapamtimo ih kao **a** i **b**

-potom u vanjskoj if naredbi provjeravamo parnost i ispišemo odgovarajući tekst

-ako je broj paran, dodatno provjeravamo da li je djeljiv s tri i ispišemo odgovarajuću poruku

-uočljivo je da za broj djeljiv s 2 i 3 dobijemo ispis obje poruke, a to je i trebalo

-primjer: Unesite dva cijela broja **x** i **y**. Ukoliko oba istodobno nisu nula ispišite tekst „Oba broja su različita od nule.“ Potom za slučaj da je **y** veći od **x** ispišite tekst „Razlika je „ i iznos razlike **y-x**.

-rješenje:

```
int main(int argc, char *argv[])
{
   short int x, y, a;
   cout<<"Unesite prvi broj->";
   cin>>x;
   cout<<"Unesite drugi broj->";
   cin>>y;
   a=y-x;
   if ((x!=0)&& (y!=0)) //za slučaj da su oba broja različita od 0
     {
      cout<<endl<<"Oba broja su razlicita od nule."<<endl;
      if (y>x)
        {
         cout<<endl<<"Razlika je "<<a<<endl;
        }
     }
   system("PAUSE");
   return EXIT_SUCCESS;
}
```

2.20.1. Višestruko uvjetno grananje

-**višestruko uvjetno grananje if** naredbom (engl. **multiple choice if**) sastoji se od **jedne if naredbe** na koju se **nakon svake ključne riječi else nadovezuje novi if** s nekim uvjetom (osim iza završnog **else**-a)

-**način pisanja višestrukog grananja if** naredbom:

```
if (uvjet1)
  {
    naredbe1;
  }
   else if (uvjet2)
      {
        naredbe2;
      }
      else if (uvjet3)
      {
        naredbe3;
      }
      .
      .
      .
           else if (uvjetN)
              {
                naredbeN;
              }
else
  {
    zadnji blok naredbi;
  }
```

-objašnjenje načina **funkcioniranja višestrukog grananja if** naredbom:

a) ako je **istinit uvjet u prvoj if naredbi**, izvršavaju se **naredbe iz prvog bloka** naredbi i višestruko grananje **završava**

b) ukoliko je **uvjet u prvom if-u neistinit**, **preskaču** se naredbe u **prvom bloku**, a nakon toga se provjerava da li je **istinit drugi uvjet**
   -ako je **istinit**, izvršava se **drugi blok** naredbi i višestruko grananje **završava**
   -ako je **neistinit** i drugi uvjet provjerava se **treći** i tako **do prvog uvjeta koji je točan**
   -ukratko, **vrše se nove provjere uvjeta tako dugo dok se ne dođe do nekog istinitog uvjeta, čime ova naredba završava**
   -**zaključak**: uvijek se izvršava **samo naredbe iza jedne provjere uvjeta** i to one koja je **prva istinita**
   -čak i da ima **više ispunjenih uvjeta** (što nije smisleno kod programiranja), izvršit će se samo one **nakon prvog ispunjenog uvjeta** pa je **besmisleno pisati ostale ispunjene uvjete**, jer se nikada **neće izvršiti**
   -ako **nije ispunjen** nijedan **uvjet iza nekog od if-ova**, izvršavaju se **naredbe iza zadnjeg else-a**

c) na osnovu prijašnjeg izlaganja, može se zaključiti da **višestruko grananje if naredbom** ima smisla, ako su **pokriveni svi mogući ishodi u nekom zadatku** (uključujući i onaj kada **nijedan uvjet nije ispunjen**)
   -dakle, **redom unosimo uvjete koje provjeravamo**, a **iza zadnjeg else pišemo naredbe** koje se izvrše, ako **nijedan uvjet nije ispunjen**
   -pritom može biti bitan **poredak uvjeta** koji se navode za provjeru

-primjer: Unesite koeficijente kvadratne jednadžbe (**a**, **b** i **c**) i odredi njezinu diskriminantu **d**. Za **d**=0 ispišite tekst „x1=x2“, za **d**>0 tekst „realni x1, x2“, a za **d**<0 tekst „kompleksni x1, x2“.

```
int main(int argc, char *argv[])
```

```
{
   double a, b, c, d;
   cout<<"Unesite a->";
   cin>>a;
   cout<<"Unesite b->";
   cin>>b;
   cout<<"Unesite c->";
   cin>>c;
   d=b*b - 4*a*c;
   if (d==0)
     {
      cout<<endl<<"x1=x2"<<endl;
     }
      else if (d>0)
        {
         cout<<endl<<"realni x1, x2"<<endl;
        }
        else //tu nema provjere uvjeta
          {
           cout<<endl<<"kompleksni x1, x2"<<endl;
          }
   system("PAUSE");
   return EXIT_SUCCESS;
}
```

-analiza primjera: nakon izračunane diskriminante prvo se provjerava da li je ona 0, ako nije da li je veća od 0, a jedini mogući preostali slučaj je da je diskriminanta manja od 0 (to je slučaj iza zadnjeg else-a kada se ne piše if)

2.21. Ugnježđeni If i višestruko uvjetno grananje - vježbe

-na nekoliko primjera **uvježavamo** upotrebu prijašnjih naredbi

-primjer: Riješite kvadratnu jednadžbu oblika $y=ax^2+bx+c$ (unesite **a**, **b** i **c**). Ispišite rješenja i tekst o vrsti rješenja (dvostruko, realno, kompleksno). Upotrijebite **višestruko grananje if naredbom**.

-rješenje:

```
#include <cstdlib>
#include <iostream>
#include <cmath>

using namespace std;

 int main(int argc, char *argv[])
{
   double a, b, c, d, x1, x2;
   cout<<"Unesite a->";
   cin>>a;
   cout<<"Unesite b->";
   cin>>b;
   cout<<"Unesite c->";
   cin>>c;
   d=b*b - 4*a*c; //diskriminanta
   if (d==0) //isto dvostruko rješenje
     {
      x1=(-b/(2*a)); //x1 ili x2, jer su isti
      cout<<endl<<"x1=x2="<<x1<<endl;
```

```
        }
        else if (d>0) //realna rješenja
          {
           cout<<endl<<"realni x1, x2"<<endl;
           x1=(-b+sqrt(d))/(2*a);
           cout<<endl<<"x1="<<x1<<endl;
           x2=(-b-sqrt(d))/(2*a);
           cout<<endl<<"x2="<<x2<<endl;
          }
          else //tu nema provjere uvjeta (kompleksna rješenja)
            {
             cout<<endl<<"kompleksni x1, x2"<<endl;
             x1=(-b/(2*a));//realni dio rješenja za x1
             cout<<endl<<"x1="<<x1;
             x2=sqrt(abs(d))/(2*a);//kompleksni dio rješenja za x1
             cout<<"+("<<x2<<"i)"<<endl;
             //realni dio rješenja za x2 je isti kao za x1
             cout<<endl<<"x2="<<x1;
             //kompleksni dio rješenja za x2 je isti kao za x1
             cout<<"-("<<x2<<"i)"<<endl;
            }
    system("PAUSE");
    return EXIT_SUCCESS;
 }
```

-analiza rješenja: vidljivo je da izračunavamo diskriminantu, najprije tražimo dvostruka rješenja i ispisujemo ih, a potom dva realna i njih ispišemo

-na kraju tražimo kompleksna rješenja čiji realni (označeno s **x1**) i imaginarni dio (označeno s **x2**) izračunamo samo jednom, a ispis vršimo uz promjenu predznaka ispred imaginarnog dijela

-time smo uštedili na računanju, ali takvo označavanje je **zbunjujuće** pa je bilo bolje uvesti zasebne varijable za realni i imaginarni dio

-primjer: U banci je novac u prostoriji sa sefom. U njoj je alarmni sustav koji (ako je uključen) provjerava stanja tri senzora: da li su vrata prostorije otvorena, je li upaljeno svjetlo u prostoriji i da li je otvoren sef. Ako je alarmni sustav uključen, a otvorena su vrata, alarm policiji šalje poruku "Vrata prostorije sa sefom su otvorena.". Ukoliko je povrh toga upaljeno svjetlo, šalje dodatnu poruku "Svjetlo je upaljeno." U slučaju da su još i vrata sefa otvorena, alarm šalje i poruku "U banci je pljačka!". U programu upotrijebite četiri cjelobrojne varijable za unos stanja senzora i uključenosti alarma: **alarm**, **vrata**, **svjetlo** i **sef**. Nula predstavlja neaktivirani senzor ili neuključeni alarm, bilo što drugo aktivirani senzor, odnosno, uključeni alarm. Za realizaciju programa upotrijebite **ugnježđeni if**.

-rješenje:

```
#include <cstdlib>
#include <iostream>

using namespace std;

 int main(int argc, char *argv[])
{
   int alarm, vrata, svjetlo, sef;
   cout<<"Unesite stanje senzora alarma->";
   cin>>alarm;
   cout<<"Unesite stanje senzora vrata->";
   cin>>vrata;
   cout<<"Unesite stanje senzora svjetla->";
   cin>>svjetlo;
```

```
   cout<<"Unesite stanje senzora sefa->";
   cin>>sef;
   if (alarm!=0) //ako je alarm uključen
     {
      if (vrata!=0) //ako su vrata otvorena
        {
         cout<<endl<<"Vrata prostorije sa sefom su otvorena."<<endl;
         if (svjetlo!=0) //ako je svjetlo upaljeno
           {
            cout<<endl<<"Svjetlo je upaljeno."<<endl;
            if (sef!=0) //ako je sef otvoren
              {
               cout<<endl<<"U banci je pljacka!"<<endl;
              }
           }
        }
     }
   system("PAUSE");
   return EXIT_SUCCESS;
}
```

-analiza rješenja: vidljivo je da se u svakom if-u provjerava aktivno stanje (kada nije 0) te se ispiše odgovarajuća poruka i ide na idući ugnježđeni if

2.22. Grananje naredbom switch-case

-do sada smo upoznali različite načine upotrebe if naredbe i njezinih varijanti (ugnježđeni if, višestruko grananje if-om)

-vidjeli smo da **višestruko grananje** daje prilično **duge programe** s dosta **if** i **else** ključnih riječi koje mogu zbuniti programera

-zato je uvedena posebna naredba koja je **zamjena za višestruko grananje if naredbom** ukoliko je grananje izvedeno na temelju **stanja vrijednosti cijelog broja**

-ta naredba zove se **switch** (ili **switch-case**)

-njezina **sintaksa** je:

```
switch (cjelobrojna vrijednost)
{
  case konstanta1:
     naredbe1;
  break;

  case konstanta2:
     naredbe2;
  break;

  .
  .
  .

  case konstantaN:
     naredbeN;
  break;

  default:
     naredbe;
```

}

-objašnjenje **pisanja** naredbe:

a) obavezno se pišu **switch**, okrugle i vitičaste zagrade - **()** i **{}** i **cjelobrojna vrijednost** u okruglim zagradama (sve označeno **sivom** bojom)

-primjer **obaveznog** dijela switch-case naredbe:

```
switch (dan)
 {
 }
```

b) **cjelobrojna vrijednost** u okruglim zagradama može biti:

1.) **izraz** koji nakon izračunavanja daje **cjelobrojnu** vrijednost

-primjer:

```
switch (2*x+4)
 {
 }
```

2.) ime **varijable** koja ima zadanu **cjelobrojnu** vrijednost

-primjer:

```
switch (broj)
 {
 }
```

3.) **cjelobrojna konstanta** - može se koristiti, ali je tada naredba praktički **besmislena**, jer se svodi na if-else naredbu

-primjer:

```
switch (5)
 {
 }
```

c) koristi se **proizvoljan broj case** ključnih riječi (**svaka** mora **završiti** ključnom riječju **break** i znakom **;**)

-primjer upotrebe samo **jedne case-break** kombinacije :

```
switch (5)
 {
   case 1:
   break;
 }
```

d) ključna riječ **default nije obavezna**, ali ako se upotrijebi, iza nje se piše znak **:**

-primjer:

```
switch (dan)
 {
   case 1:
   break;
   default:
 }
```

-bitna napomena: iza **default** i iza **brojeva** u case piše se **:**, a iza **break** i ostalih naredbi **;**

e) **iza** svake riječi **case** navodi se **cijeli broj** (uobičajeno **pozitivan**, ali ne mora biti) praćen znakom **:**

-primjer:

```
switch (mjesec)
 {
   case 6:
   break;
 }
```

f) cijeli **brojevi** iza riječi **case** mogu biti **bilo koji**, ali logično je da budu **različiti** (nije nužno)

-primjer:

```
switch (broj)
```

```
{
  case -6:
  break;
  case 2:
  break;
  case 2: //može biti isti broj kao u prijašnjem case-u
  break;
}
```

g) oznaka **naredbe** ili **naredbeN** (N je prirodan broj) označava da se radi o **grupi naredbi** pri čemu **nije** potrebno koristiti vitičaste zagrade (**{}**) za njihovo grupiranje, već one završavaju uobičajeno (najčešće znakom **;**)

-primjer:

```
switch (dan)
 {
  case 1:
  cout<<"Dan u tjednu je ponedjeljak."<<endl;
    a=a*2;
  break;
  default:
    cout<<"Unijeli ste krivi broj."<<endl;
    cout<<"Ponovite unos broja->";
    cin>>dan;
 }
```

-objašnjenje **funkcioniranja** naredbe:

a) naredba radi tako da **redom** kako su napisane **case** naredbe **uspoređuje konstante iza case** s vrijednošću u **zagradi** iza switch

-kada je **konstanta** iza nekog case-a **jednaka** iznosu u **zagradi** iza switch-a, izvršavaju se **sve naredbe** do **prvog break**-a i time naredba switch-case **završava**

-primjer:

```
switch (1)
 {
 case 1: //za broj 1 u zagradi ispisuje se tekst o ponedjeljku
    cout<<"Danas je ponedjeljak."<<endl;
    cout<<"Naredba switch-case je završila";
 break;
 case 4:
    cout<<"Danas je četvrtak."<<endl;
    cout<<"Naredba switch-case je završila";
 break;
 }
```

b) ukoliko se **nijedan broj** iza **case** ne poklapa s **brojem u zagradi** iza switch, izvršavaju se naredbe **iza default:** (**default** u računalstvu označava **podrazumijevanu vrijednost**, dakle ako ona **nije drukčije zadana**)

-primjer:

```
switch (7)
 {
  case 1:
    cout<<"Odabrali ste broj 1."<<endl;
  break;
  case 4:
    cout<<"Odabrali ste broj 4."<<endl;
```

```
 break;
 default:
   cout<<"Unijeli ste pogrešan broj."<<endl; //izvršava se ova naredba, jer ne postoji case s brojem 7
 }
```

-ako u prijašnjem slučaju **default:** ne postoji, **switch-case** naredba **završava** kao da je nije ni bilo

-primjer:

```
switch (7)
 {
  case 1:
    cout<<"Odabrali ste broj 1."<<endl;
  break;
  case 4:
    cout<<"Odabrali ste broj 4."<<endl;
  break;
} //ova naredba ne radi ništa
```

c) naredba switch-case **korisna** je kada se koristi operacija izračunavanja **ostatka cjelobrojnog dijeljenja** (**%**) za izračun vrijednosti u zagradi iza ključne riječi **switch** ili kada se neki cijeli broj **unosi tipkovnicom**

-primjer:

```
dan=broj%7; //ostatak cjelobrojnog djeljenja je od 0 od 6 (npr. 0 označava ponedjeljak, 1 utorak itd.)
```

-tako napisana naredba koristi se pri radu s **danima u tjednu**, **mjesecima** i slično ili se pomoću nje može birati što želimo postići u programu (izrada **menija s naredbama**)

-primjer: Varijabla **ocjena** (tipa **int**) sadrži broj od 1 do 5 koji je unijet tipkovnicom. Za svaku ocjenu ispišite njezin naziv.

-rješenje:

```
#include <cstdlib>
#include <iostream>

using namespace std;

 int main(int argc, char *argv[])
{
 int ocjena;
 cout<<"Upisite ocjenu od 1 do 5->";
 cin>>ocjena;

 switch (ocjena)
 {
  case 1:
    cout<<endl<<"jedan"<<endl;
  break;
  case 2:
    cout<<endl<<"dva"<<endl;
  break;
  case 3:
    cout<<endl<<"tri"<<endl;
  break;
  case 4:
    cout<<endl<<"cetiri"<<endl;
  break;
  case 5:
```

```
   cout<<endl<<"pet"<<endl;
 break;
 }
 system("PAUSE");
 return EXIT_SUCCESS;
}
```

-analiza rješenja: vidi se da se ovisno o **iznosu varijable ocjena** izvrše naredbe iz nekog od **case**-ova
-uočite da se za **pogrešnu** vrijednost varijable **ne izvršavaju** bilo koje naredbe
-primjer: Varijabla **dan** (tipa **int**) sadrži broj od 1 do 7 koji je unijet tipkovnicom. Za svaki iznos ispišite naziv dana (1 je ponedjeljak itd.). Ukoliko broj **nije** u opsegu od 1 do 7, napišite tekst "Pogrešan broj.".
-rješenje:

```
#include <cstdlib>
#include <iostream>

using namespace std;

 int main(int argc, char *argv[])
{
 int dan;
 cout<<"Upisite broj od 1 do 7->";
 cin>>dan;

 switch (dan)
 {
  case 1:
    cout<<endl<<"ponedjeljak"<<endl;
  break;
  case 2:
    cout<<endl<<"utorak"<<endl;
  break;
  case 3:
    cout<<endl<<"srijeda"<<endl;
  break;
  case 4:
    cout<<endl<<"cetvrtak"<<endl;
  break;
  case 5:
    cout<<endl<<"petak"<<endl;
  break;
  case 6:
    cout<<endl<<"subota"<<endl;
  break;
  case 7:
    cout<<endl<<"nedjelja"<<endl;
  break;
  default:
    cout<<endl<<"Pogresan broj!"<<endl;
  break;
 }
 system("PAUSE");
 return EXIT_SUCCESS;
}
```

-analiza rješenja: uočljivo je da u odnosu na prvi primjer imamo dodatnu ključnu riječ **default** iza koje se nalazi naredba koja se izvršava, ako broj nije u opsegu od 1 do 7

2.23. Grananje naredbom switch-case (uvježbavanje)

-upotrebu **switch-case** naredbe uvježbavat ćemo na **složenijim** primjerima

-primjer: Napravite meni za matematičke operacije nad dva realna broja **a** i **b** (tip **double**). Unesite broj od 1 do 4 u varijablu **operacija** i pridružite mu ova značenja:
1 - zbrajanje
2 - oduzimanje
3 - množenje
4 - djeljenje
Ispišite rezultat operacije oblika: **a** operacija **b** (npr. **a/b**). Ako je broj **b** jednak 0, a izabrana je operacija djeljenja, ispišite umjesto rezultata poruku "Djeljenje nulom nije definirano!". U slučaju da je unesena kriva oznaka operacije, ispišite tekst "Unijeli ste pogrešnu oznaku operacije!".

-rješenje:

```
#include <cstdlib>
#include <iostream>

using namespace std;

int main(int argc, char *argv[])
{
  int operacija;
  double rezultat, a, b;
  cout<<"Unesite prvi broj->";
  cin>>a;
  cout<<"Unesite drugi broj->";
  cin>>b;
  cout<<"Unesite oznaku operacije (1=+ 2=- 3=* 4=/)->";
  cin>>operacija;

  switch (operacija)
  {
   case 1:
     rezultat=a+b;
     cout<<endl<<"Rezultat je "<<rezultat<<endl;
   break;
   case 2:
     rezultat=a-b;
     cout<<endl<<"Rezultat je "<<rezultat<<endl;
   break;
   case 3:
     rezultat=a*b;
     cout<<endl<<"Rezultat je "<<rezultat<<endl;
   break;
   case 4:
     if (b==0)
        {
          cout<<endl<<"Djeljenje nulom nije definirano!"<<endl;
        }
     else
        {
          rezultat=a/b;
```

```
        cout<<endl<<"Rezultat je "<<rezultat<<endl;
       }
  break;
  default:
  cout<<endl<<"Unijeli ste pogrešnu oznaku operacije!"<<endl;
 }
 system("PAUSE");
 return EXIT_SUCCESS;
}
```

-analiza primjera: vidimo da unosom broja od 1 do 4 biramo **jedan** slučaj u **switch** naredbi za koji se izračunava odgovarajuća **operacija** i ispisuje rezultat

-ukoliko smo unijeli **pogrešnu oznaku** operacije, u **if - else** naredbi to utvrđujemo i ispisujemo odgovarajuću **poruku**, inače izračunamo **a/b** (za unešen broj 4)

-primjer: Napišite program koji upotrebom **switch-case** naredbe određuje koji je dan u tjednu bio na neki unešeni datum zadan u obliku: **dd mm gggg** (d=dan, m=mjesec, g=godina).

-rješenje:

```
#include <cstdlib>
#include <iostream>

using namespace std;

int main(int argc, char *argv[])
{
 int dan, mjesec;
 long int godina, datum, pomoc;

 cout<<"Unesite datum u obliku dd mm gggg ->";
 cin>>dan>>mjesec>>godina; //unos tri broja odvojena razmakom
 pomoc=365*godina+dan+31*(mjesec-1); //dio koji se ponavlja u obje formule
 if (mjesec<3)
   {
    datum=pomoc+(godina-1)/4-3*((godina-1)/100+1)/4;
   }
 else
   {
    datum=pomoc-static_cast<int>(0.4*mjesec+2.3)+godina/4-3*(godina/100+1)/4;
    //static_cast<int> odbacuje decimale i realni broj u zagradi pretvara u cijeli
   }
 cout<<endl<<dan<<". "<<mjesec<<". "<<godina<<". je "; //zajednicki dio ispisa
 switch (datum%7)
 {
  case 0:
    cout<<"subota."<<endl;
  break;
  case 1:
    cout<<"nedelja."<<endl;
  break;
  case 2:
    cout<<"ponedjeljak."<<endl;
  break;
  case 3:
    cout<<"utorak."<<endl;
  break;
```

```
 case 4:
   cout<<"srijeda."<<endl;
 break;
 case 5:
   cout<<"cetvrtak."<<endl;
 break;
 default://tu smo mogli staviti i case 6:, ali je ovo krace
   cout<<"petak."<<endl;
 }
 system("PAUSE");
 return EXIT_SUCCESS;
}
```

-analiza primjera: nakon **unosa datuma** (u **tri varijable** pomoću **jedne cin** naredbe) izračunava se **zajednički** dio formule i potom, ovisno o **mjesecu**, izračunava **ostatak** formule
-upotrijebili smo novu naredbu (**static_cast<int>**) koja **odbacuje decimale i realni broj u zagradi iza nje pretvara u cijeli**
-nakon izračuna formula ispisuje se **zajednički tekst** i potom se određuje **ostatak cjelobrojnog djeljenja** sa 7 u **switch** naredbi
-ovisno o **ostatku** ispisuje se odgovarajući **dan** u poruci
-vidljivo je da za **ostatak** jednak **6** ne koristimo **case 6:**, već **default:** (daje **isti** rezultat), jer je to **kraće** napisano

2.25. Matematičke funkcije - ocjenjivanje vježbe

-zadatak ima **pet** elemenata ocjenjivanja: **četiri** dijela za izračun podizraza (**a** do **d**) i završno uklapanje ta četiri dijela u **konačnu** formulu
-svaki element boduje se **jednim** bodom, a ukupni **broj bodova** jednak je **ocjeni** vježbe (osim za 0 bodova)
-zadatak grupe **A**: Izračunajte upotrebom matematičkih funkcija slijedeći izraz:

$y = \frac{\sqrt[3]{a^2 b^4}}{\sqrt{c}d}$ (1 bod)

-pritom su **a**, **b**, **c** i **d** zadani ovim izrazima:
1.) $a = |x^3 sinx - 1|$ (1 bod)
2.) $b = 2\cos(2x+1) - 1$ (1 bod)
3.) $c = 4tg^2 x^2 + \sqrt{|x|}$ (1 bod)
4.) $d = 3x^4 + 2x^3 + x^2$ (1 bod)
-varijable **a**, **b**, **c**, **d**, **x** i **y** su tipa double

-zadatak grupe **B**: Izračunajte upotrebom matematičkih funkcija slijedeći izraz:

$y = \frac{\sqrt[4]{a^3 b^2}}{\sqrt{c}d}$ (1 bod)

-pritom su **a**, **b**, **c** i **d** zadani ovim izrazima:
1.) $a = |x^2 cosx + 1|$ (1 bod)
2.) $b = 3\sin(4x-1) + 3$ (1 bod)
3.) $c = 2tg^2 x^3 + \sqrt{|x|}$ (1 bod)
4.) $d = 4x^5 + 3x^3 + x^2$ (1 bod)
-varijable **a**, **b**, **c**, **d**, **x** i **y** su tipa double

2.25. Grananje if naredbom - ocjenjivanje vježbe

-zadatak ima **pet** elemenata ocjenjivanja
-svaki element boduje se **jednim** bodom, a ukupni **broj bodova** jednak je **ocjeni** vježbe (osim za 0 bodova)

-u zadatku se koristi **bilo koji** oblik **if** naredbi

-zadatak grupe **A**: Upotrebom **if** naredbi napišite program u kojem se uz pripadajući tekst (npr. "Unesi a") unose vrijednosti cjelobrojnih pozitivnih varijabli **a** i **b**. Za njih napravite ove radnje:

a) ako je **a** veće od **b** i **b** veće od 2, ispišite vrijednost varijable **b** (1 bod)
b) ako je **a** parno, udvostručite ga (1 bod)
c) ako je **a** u opsegu od 2 do 32 (s uključenim granicama), povećajte ga za 1 i ispišite (1 bod), inače ga umanjite za 2 i ispišite (1 bod)
d) ako je **b** veći ili jednak 25 i djeljiv sa 7, ispišite ostatak djeljenja **b** s 8

-zadatak grupe **B**: Upotrebom **if** naredbi napišite program u kojem se uz pripadajući tekst (npr. "Unesi a") unose vrijednosti cjelobrojnih pozitivnih varijabli **a** i **b**. Za njih napravite ove radnje:

a) ako je **b** veće od **a** i **b** manje od 22, ispišite vrijednost varijable **a** (1 bod)
b) ako je **b** parno, utrostručite ga (1 bod)
c) ako je **b** u opsegu od 12 do 47 (bez granica), umanjite ga za 3 i ispišite (1 bod), inače ga povećajte za 4 i ispišite (1 bod)
d) ako je **a** manji ili jednak 34 i djeljiv sa 6, ispišite ostatak djeljenja **b** s 3

2.26. Grananje switch naredbom - ocjenjivanje vježbe

-zadatak ima **deset** elemenata ocjenjivanja
-svaki element boduje se **jednim** bodom, a kriterij **ocjenjivanja** vježbe je slijedeći:

0 - 4 boda	**1**
5 bodova	**2**
6 bodova	**3**
7 - 8 bodova	**4**
9 - 10 bodova	**5**

-zadatak grupe **A**: Upotrebom **switch** naredbe napišite program u kojem se uz pripadajući tekst (npr. "Unesi a") unose vrijednosti cjelobrojnih pozitivnih varijabli **a** i **b**. Za njih napravite ove radnje:

a) ako je **a** veće od **b** upotrijebite **switch** naredbu koja obrađuje ove vrijednosti varijable **b**:
    1.) za **b** jednako 2 ispišite tekst "Dva." (1 bod)
    2.) za **b** jednako 4 udvostručite **a** i ispišite ga (1 bod)
    3.) za **b** jednako 6 odredite ostatak djeljenja **a** sa 7 i ispišite ga (1 bod)
    4.) za **b** jednako 10 povećajte **a** za 2 i ispišite ga (1 bod)
    5.) za bilo koju drugu vrijednost ispišite tekst "Pogrešno."
b) ako je **a** manji ili jednak **b** upotrijebite **switch** naredbu koja obrađuje ove vrijednosti varijable **a**:
    1.) za **a** jednako 3 ispišite tekst "Tri." (1 bod)
    2.) za **a** jednako 5 utrostručite **b** i ispišite ga (1 bod)
    3.) za **a** jednako 7 odredite ostatak djeljenja **b** sa 5 i ispišite ga (1 bod)
    4.) za **a** jednako 8 povećajte **b** za 5 i ispišite ga (1 bod)
    5.) za bilo koju drugu vrijednost ispišite tekst "Nepravilno."

-zadatak grupe **B**: Upotrebom **switch** naredbe napišite program u kojem se uz pripadajući tekst (npr. "Unesi a") unose vrijednosti cjelobrojnih pozitivnih varijabli **a** i **b**. Za njih napravite ove radnje:

a) ako je **a** manje od **b** upotrijebite **switch** naredbu koja obrađuje ove vrijednosti varijable **b**:
    1.) za **b** jednako 3 ispišite tekst "Tri." (1 bod)
    2.) za **b** jednako 5 udvostručite **a** i ispišite ga (1 bod)
    3.) za **b** jednako 7 odredite ostatak djeljenja **a** sa 6 i ispišite ga (1 bod)
    4.) za **b** jednako 11 povećajte **a** za 3 i ispišite ga (1 bod)
    5.) za bilo koju drugu vrijednost ispišite tekst "Pogrešno."
b) ako je **a** veći ili jednak **b** upotrijebite **switch** naredbu koja obrađuje ove vrijednosti varijable **a**:
    1.) za **a** jednako 4 ispišite tekst "Četiri." (1 bod)

2.) za **a** jednako 6 utrostručite **b** i ispišite ga (1 bod)
3.) za **a** jednako 8 odredite ostatak djeljenja **b** sa 5 i ispišite ga (1 bod)
4.) za **a** jednako 7 povećajte **b** za 5 i ispišite ga (1 bod)
5.) za bilo koju drugu vrijednost ispišite tekst "Nepravilno."

2.27. Petlje

-često je u programima potrebno **ponoviti** dio naredbi
-to nam omogućuju programske konstrukcije nazvane **petlje** (engl. **loops**)
-bez ponavljanja naredbi teško je napisati program koji radi nešto korisno
-ponavljanje u programima zamjenjuje **pisanje većeg broja identičnih naredbi**
-**ponavljanjem** dobivamo:
a) **smanjenje dužine** programa (**brže** se izvodi i zauzima **manje memorije**)
b) **olakšano analiziranje** programa
c) **brže i jednostavnije modificiranje** (**mijenjanje**) programa
d) puno lakšu upotrebu **memorijskih struktura** (npr. polja brojeva)
-petlje se dijele u dvije grupe obzirom na **broj ponavljanja**:
a) petlje s **određenim** (**zadanim**) **brojem ponavljanja**
-takvu petlju u C++ jeziku zovemo **for petlja** (engl. **for loop**)
b) petlje **bez zadanog broja ponavljanja**
-takve petlje u C++ jeziku su **while** (engl. **while loop**) i **do-while** (engl. **do-while loop**)
-da bismo znali **koliko puta ili dokle ponavljamo**, u petljama se koristi **cjelobrojna** (**int** tip) varijabla koja nam pomaže kod **brojanja**
-nju najčešće zovemo **brojač** (engl. **counter**) u petlji, a njezino **ime** može biti bilo koje mada se uobičajeno koriste **jednoslovna** imena (npr. i, j, k, l, m, n, a, b, c) ili riječ koja upućuje na **namjenu** varijable (npr. brojac, brojac1, brojac2,...)
-**vrijednost brojača** u petlji se **mijenja** tako da on:
a) **raste**
-takav način promjene je **češći**
-za **rastući brojač** u petlji kažemo da broji **uzlazno** (engl. **ascending counting**)
b) **pada**
-takav način promjene je **rjeđe** u upotrebi
-za **padajući brojač** u petlji kažemo da broji **silazno** (engl. **descending counting**)
-za brojač se najčešće koristi **pozitivne cjelobrojne vrijednosti** (npr. brojanje od 1 do 10 ili od 10 do 1), rjeđe **negativne** (npr. od -30 do -20 ili od -20 do -50), a najrjeđe **mješovite** cjelobrojne vrijednosti (npr. od -3 do 2 ili od 15 do -10)
-ipak, treba naglasiti da mi uvijek **početnu i završnu vrijednost brojača** kao i način njegove **promjene** podređujemo čim **efikasnijem** (**bržem, kraćem, razumljivijem**) **izvođenju** ponavljanja
-primjer: ukoliko trebamo ispisati redom cijele brojeve od -10 do 20, mi možemo brojati od 0 do 30 i nakon svakog ponavljanja oduzeti 10 i ispisati traženi broj
-brži način je brojati direktno od -10 do 20 pa oduzimanje otpada
-mada se za **promjenu** vrijednosti **brojača** može koristiti **bilo koji** način koji je dopušten nad **cjelobrojnim** tipom (npr. aritmetičke, logičke i relacijske operacije), mi se uobičajeno ograničavamo na samo dvije:
a) **zbrajanje** (**uobičajeno**)
-najčešće se vrijednost brojača **povećava za 1** (**inkrementira**)
b) **oduzimanje** (**rjeđe**)
-najčešće se vrijednost brojača **smanjuje za 1** (**dekrementira**)

2.27.1. Petlja for

-petlja **for** služi nam kada moramo dio programa **ponoviti određeni broj puta**
-**način pisanja for petlje**:

```
for (početni_izraz;uvjet_izvođenja;promjena_izraza)
   {
     blok_naredbi;
   }
```

-**objašnjenja** dijelova **for** petlje:

a) **početni_izraz**

-to je izraz koji **nakon_izračunavanja** definira **početno_stanje_varijable** (**brojača**) koju koristimo za brojanje

-uobičajeno je to tek **naredba pridruživanja početne vrijednosti brojača**

-primjer pridruživanja početne vrijednosti brojača: a=1; ili i=-2;....

b) **uvjet_izvođenja**

-to je **izraz** koji **nakon_izračunavanja** mora dati rezultat **logičke** vrijednosti (**bool**) ili **cjelobrojne** vrijednosti koja se tumači kao **logička** vrijednost (**0** je **false**, **ostalo** je **true**)

-za dobivanje uvjeta izvođenja uobičajeno se koriste **operacije usporedbe** koje imaju u sebi znak **manje** ili **veće** (tj. **<**, **>**, **<=** i **>=**), a **ne** i operacije čiste **jednakosti ili nejednakosti** (**==** i **!=**)

-primjer uvjeta izvođenja: i>10;
a>=-10;
m<3;
j<=100;

c) **promjena_izraza**

-to je izraz koji definira **način promjene vrijednosti brojača**

-najčešće se radi o **inkrementiranju** (**++**) ili **dekrementiranju** (**--**) **brojača**

-primjer: **i**++
j--
i=i+2
a=**a**-23

-**napomena**: iza promjene izraza **ne piše se** znak **;**

d) **zagrade** u for petlji (**obične** i **vitičaste**) su **obavezne**

-u **vitičastim** se pišu **bilo koje naredbe koje ponavljamo**

-u petlji možemo **ponavljati** neku drugu **petlju** pa takve petlje zovemo **ugnježđenim** (engl. **nested loops**)

e) **u zagradi obavezno** koristimo dva znaka **;**, dok **sve ostalo u zagradni nije obavezno** pisati, ali je **uobičajeno**

-objašnjenje **funkcioniranja for** petlje:

a) najprije se **izračuna početni_izraz** koji daje **početnu vrijednost brojača**

-kao što je već rečeno, uobičajeno je to pridruživanje **podesne početne vrijednosti** brojaču (npr. i=1;)

b) izračuna se **uvjet_izvođenja**

-ukoliko je on **true**, **naredbe** u petlji se **izvedu**, ako **nije true**, for pelja je **završila** i program se nastavlja prvom naredbom iza nje

-primjer: a<=10;

-**napomena**: **početni_izraz**, **uvjet_izvođenja** i **promjena_izraza** odnose se na **istu varijablu** koja služi za **brojanje** u petlji

-primjer: ako je početni_izraz zadan kao **a**=1; tada može uvjet_izvođenja biti npr. **a**<23; (dakle, radi se o **istoj varijabli** za brojanje)

c) odredi se **promjena_izraza**

-ukoliko **pelja_nije_završila**, već su se naredbe u njoj **ponovile**, potrebno je **prije ponovne provjere uvjeta izvođenja**, utjecati na **promjenu vrijednosti brojača**

-to se radi u dijelu for petlje koji smo označili kao **promjena_izraza**

-**uobičajena** promjena je pomoću **zbrajanja** ili **oduzimanja**

-primjer: i++

```
a--
a=a-3
b=b+4
```

-primjer pisanja **cijele** petlje:

```
for (i=3;i<=10;i++)
  {
  }
```

d) petlja se tako dugo **ponavlja** dok je **uvjet_izvođenja true**

-kada on postane jednak **false**, **petlja je završila**

e) **početni_izraz** određuje se **samo jednom**, **uvjet_izvođenja** izračunava se dok ne **postane false**, a **promjena_izraza** određuje se **nakon** što je izračunavanje **uvjeta izvođenja** dalo **true**

-vrlo **je bitno naglasiti** da se najprije **odredi uvjet_izvođenja**, potom (ako je on bio **true**) se izvedu sve **naredbe koje ponavljamo** u petlji, a tek zatim se **odredi promjena_izraza**

-primjer for petlje:

```
for (i=1;i<=5;i++)
{
 cout<<i<<endl;
}
```

-analiza primjera:

-najprije se provede dodjela početne vrijednosti brojaču **i** (**i**=1)

-potom se provjerava da li je 1<=5

-pošto je to istina, izvede se naredba u petlji (ispiše se broj 1 i prebaci ispis u novi red)

-u idućem koraku **i** se poveća za 1 i postane jednak 2

-provjeravamo uvjet ponavljanja da li je 2>=5

-to je istina pa ispišemo broj 2 u novom redu i povećamo **i** za 1 (**i** postaje jednak 3)

-ponovo provjeravamo da li je novi **i** takav da zadovoljava uvjet da je **i**<=5

-pošto je 3<=5, petlja se izvodi, na ekranu se ispiše 3 i prebaci ispis u novi red, a **i** postaje jednak 4

-kako vrijedi 4<=5, ispišemo i 4, dok **i** postaje jednak 5

-još uvijek vrijedi da je 5<=5, te ispisujemo 5, dok **i** postaje jednak 6

-ali više ne vrijedi da je 6<=5 pa petlja završava

-rezultat izvođenja petlje je ovaj ispis na ekranu:

```
1
2
3
4
5
```

-bitna napomena: **nije_poželjno_u_petlji** na neki drukčiji način **mijenjati_brojač** koji koristimo za **brojanje ponavljanja**

-dakle, nije zabranjeno mijenjati brojač i na neki drugi način, ali to utječe na na broj ponavljanja u petlji te moramo dobro **razmisliti** kakav je taj utjecaj i dokle se petlja izvodi

-primjer:

```
for (i=1; i<=5;i++)
  {
    i=i+2;
  }
```

-analiza primjera:

-na početku je i=1;

-pošto je 1<=5, izvodi se naredbu i=1+2; te je sada i=3

-nakon toga se dodatno brojač **i** poveća za 1 (i++) te postaje jednak 4

-vrijedi da je 4<=5 i petlja se ponavlja

-izvodi se i=i+2; te je i=6, a potom se izvede prirast brojača (i++) i **i** je jednak 7

-ne vrijedi da je 7<=5 i petlja je završila

-dakle, petlja se izvela samo dva puta, dok bi se inače (bez utjecaja na **i** u dijelu petlje koji ponavljamo) petlja izvela pet puta
-da smo ispisivali brojeve na ekranu nakon naredbe i=i+2; dobili bismo ispis:

3
6

-bez naredbe i=i+2; u petlji dobili bismo ispis:

1
2
3
4
5

-dakle, vidimo da naredbe koje **mijenjaju brojač** u dijelu petlje koji **ponavljamo**, **bitno utječu na rezultat**
-ukoliko imamo pelju u kojoj se **brojač smanjuje ili povećava za 1**, **broj ponavljanja** određujemo po formuli: $|G - D| \pm 1$
-u njoj je **G gornja granica brojanja** (**viša** granica), a **D donja granica brojanja** (**niža** granica)
-pritom kod dobivanja **obje granice** u for petlji mora pisati = (inače je **granica za 1 manja**)
-oba broja uzimaju se **sa svojim predznacima**, za **prirast** brojača pišemo **+1**, a za **umanjenje** brojača **-1**
-primjer: Odredite broj ponavljanja u slijedećoj petlji:

```
for (i=-2;i<=10;i++)
   {
    cout<<i<<endl;
    }
```

-rješenje: G=10, D=-2, broj ponavljanja je abs(10-(-2))+1=abs(12)+1=12+1=13
-**napomena**: broj **1** se **dodaje ili oduzima** u formuli za broj ponavljanja zbog toga što ponavljanje vršimo i kod nailaska **na gornju i donju granicu** (zbog = u oba dijela upravljačke strukture for petlje)
-kada bi ista petlja bila napisana **bez** = u dijelu za uvjet izvođenja (dakle, for (i=-2;i<10;i++)), dobili bismo **za 1 manji broj izvođenja** (12, ne 13)
-primjer: Napišite program koji na ekranu ispisuje sve brojeve od 0 do 5.
-rješenje:

```
#include <cstdlib>
#include <iostream>

using namespace std;

int main(int argc, char *argv[])
{
   short int i;
   for (i=0;i<=5;i++)
      {
       cout<<i<<endl;
       }
   system("PAUSE");
   return EXIT_SUCCESS;
}
```

-analiza rješenja: početna vrijednost brojača je 0, a pošto trebamo ispisati sve brojeve, povećavamo ga za 1 sve dok nije jednak gornjoj granici (5)
-rezultat na ekranu je:

0
1
2
3

4
5

-primjer: Napišite program koji na ekranu ispisuje neparne brojeve od 10 do 1.
-rješenje:

```
#include <cstdlib>
#include <iostream>

using namespace std;

int main(int argc, char *argv[])
{
   short int i;
   for (i=9;i>=1;i=i-2) //i se smanjuje za 2
   {
    cout<<i<<endl;
   }
   system("PAUSE");
   return EXIT_SUCCESS;
}
```

-analiza rješenja: početna vrijednost brojača je 9 (prvi neparan broj koji trebamo ispisati), a pošto trebamo ispisati neparne brojeve, umanjujemo ga za 2, a ne za 1
-donja granica je 1, a pritom smo morali staviti uvjet da je i>=1 (dakle, ponavljaj sve dok je **i** veći ili jednak od 1)
-na ekranu dobivamo ovaj ispis:

```
9
7
5
3
1
```

-analizirajte ovu netipično napisanu petlju:

```
for (;;)
   {
   }
```

-analiza: **uvjet** u petlji se smatra **uvijek točnim** te dobivamo tzv. **beskonačnu petlju** (engl. **endless loop**)
-takva petlja ponavlja se sve **dok ne ugasimo program** pomoću **Upravitelja zadataka** (engl. **task manager**)

2.27.2. Petlja for – uvježbavanje

-cilj današnjeg predavanja je uvježbati upotrebu **for petlje na složenijim primjerima**
-primjer: Unesite cijeli broj (varijabla **broj** tipa int). Ukoliko je pozitivan (≥0), ispišite sve parne brojeve od 100 do 0. U suprotnom slučaju ispišite sve neparne brojeve u opsegu od -100 do 1000.
-rješenje:

```
#include <cstdlib>
#include <iostream>

using namespace std;

int main(int argc, char *argv[])
{
   int broj, i;
   cout<<"Unesite broj-->";
```

```
   cin>>broj;
   if (broj>=0)//povjera da li je broj pozitivan
     {
      for (i=100;i>=0;i=i-2)//ispis parnih brojeva od 100 do 0
        {
         cout<<i<<endl;
        }
     }
   else
     {
      for (i=-99;i<1000;i=i+2)//ispis neparnih brojeva od -99 do 999
        {
         cout<<i<<endl;
        }
     }
   system("PAUSE");
   return EXIT_SUCCESS;
}
```

-analiza rješenja: nakon unosa varijable **broj** vršimo provjeru (**if-else**) da li je **pozitivna** ili **negativna**
-za pozitivnu varijablu **broj** ispišemo prvom **for** petljom **parne** brojeve od 100 do 0 (obje **granične vrijednosti** brojača petlje su **parne** pa su uključene u ispis)
-za **negativnu** varijablu **broj** ispišemo drugom **for** petljom **neparne** brojeve od -99 do 999 (obje zadane vrijednosti opsega bile su parne pa su promjenjene na način da se ispišu **samo neparni** brojevi)
-prva vrijednost zato postaje -99 (granica je bila -100), dok druga ostaje 1000, ali se **uvjet** završetka petlje **mijenja** (i<1000 umjesto i<=1000) te je rezultat **promjena gornje granice** na 999
-primjer: Unesite dva različita cijela broja (varijable **broj1** i **broj2** tipa int). Manji od njih predstavlja **donju**, a veći **gornju** granicu for petlje u kojoj se ispisuje brojeve od donje do gornje granice s **korakom** 5. Ukoliko je donja granica manja od 50, postavite ju da bude 50. Slično tome, ako je gornja granica veća od 255, fiksirajte ju na 255.
-rješenje:

```
#include <cstdlib>
#include <iostream>

using namespace std;

int main(int argc, char *argv[])
{
   int broj1, broj2, pomoc, i;
   cout<<"Unesite prvi broj-->";
   cin>>broj1;
   cout<<endl;
   cout<<"Unesite drugi broj-->";
   cin>>broj2;
   cout<<endl;
   if (broj2<=broj1)//povjera da li je donja granica veća od gornje
     {
      pomoc=broj1;
      broj1=broj2;
      broj2=pomoc;//zamjena sadržaja varijabli broj1 i broj2
     }
   if (broj1<50)
     {
      broj1=50;//default vrijednost granice, ako je manja od 50
```

```
    }
   if (broj2>255)
     {
      broj2=255;//default vrijednost granice, ako je veća od 255
     }
   for (i=broj1;i<=broj2;i=i+5)//ispis s korakom petlje 5
        {
         cout<<i<<endl;
        }
   system("PAUSE");
   return EXIT_SUCCESS;
}
```

-analiza rješenja: nakon **unosa** brojeva u prvom **if-u** vršimo **provjeru** (i po potrebi **zamjenu**) tako da varijabla broj1 bude **niža** (donja), a broj2 **viša** (gornja) granica petlje
-idućim **if-ovima** štitimo se od **neželjenih vrijednosti granica** i u tom slučaju granice poprimaju **unaprijed zadanu vrijednost** (**default** vrijednost)
-na kraju, u običnoj **for** petlji ispisujemo brojeve u opsegu od broj1 do broj2 s povećanjem za 5 (korak petlje je 5)
-primjer: Ispišite tablicu množenja od 1 do 10 pomoću **ugnježđenih for petlji**.
-rješenje:

```
#include <cstdlib>
#include <iostream>
#include <iomanip> //potrebno za formatirani ispis broja

using namespace std;

int main(int argc, char *argv[])
{
   short int redak, stupac;
   for (redak=1;redak<=10;redak++)//redovi tablice od 1 do 10
        {
         for (stupac=1;stupac<=10;stupac++) //prelaz na ispis jednog retka
            {
             cout<<setw(5)<<redak*stupac; //ispis jednog broja u retku tablice
            }
           cout<<endl; //pomak u novi redak
        }

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

-analiza rješenja: za ispis koristimo **ugnježđene for petlje** (engl. **nested for loops**)
-**vanjska for petlja** definira koji se redak tablice ispisuje, a **unutrašnja** koji se broj u retku ispisuje
-prelaz u **novi redak** (cout<<endl;) vrši se tek na kraju ispisa retka, stoga koristimo **dvije cout** naredbe, od kojih prva piše u istom retku, a **van prve for petlje** vrši se prelaz u novi redak
-uočite da se **unutrašnja** petlja izvodi **puno više puta od vanjske**, jer za svaki **prolaz kroz vanjsku** petlju napravi se **kompletna unutrašnja petlja**
-**napomena**: zbog potrebe za **pravilnim poravnanjem brojeva u tablici** oni su **formatirani** na ispis duljine do 5 znakova (najviše 3 znamenke, te dva razmaka)
-za takav formatirani ispis koristili smo se **manipulatorom** (**rukovateljem**) **ispisa** **setw(5)** koji definira **ispis broja** cout naredbom tako da je **širine** 5 znamenki
-zbog upotrebe manipulatora **setw()**, morali smo uključiti njegovu **definiciju** u odgovarajućoj zaglavnoj datoteci (**#include <iomanip>**)

-želite li vidjeti učinak manipulatora setw(5), probajte ispis s njim i bez njega

2.27.3. Petlje while i do-while

-za razliku od **for** petlje kod koje je **definiran broj ponavljanja**, kod petlji **while** i **do-while** **ne znamo unaprijed broj ponavljanja**, već on ovisi o **zadovoljenju uvjeta** koje smo definirali

2.27.3.1. Petlja while

-u strukturi ove petlje koristi se engleska riječ **while** koju možemo (za potrebe petlje) prevesti kao: **dok je istina** (zato ovu petlju upotrebljavamo u zadacima koji koriste ovu ili sličnu jezičnu interpretaciju)
-**sintaksa while** petlje je slijedeća:

```
while (uvjet)
    {
      naredbe;
    }
```

-**objašnjenje sintakse**: obavezno se pišu **while**, okrugle **()** i vitičaste **{}** zagrade
a) **uvjet** može biti bilo što (kao kod **if-a**), s time da se može tumačiti kao **logička vrijednost** (**true** ili **false**)
-dakle, **uvjet** može biti:
1.) **konstanta** (korisno samo za dobivanje **beskonačne petlje**)
2.) **ime varijable** (**cjelobrojna** ili **logička** varijabla čija se vrijednost u programu **mora mijenjati**, jer inače dobivamo **beskonačnu petlju**)
3.) **izraz** koji daje **logičku** ili **cjelobrojnu** vrijednost (**0** se tumači kao **false**, **ostalo** kao **true**)
-uobičajeno se koriste **operatori usporedbe** (npr. <, >, <=, >=)
b) unutar **vitičastih** zagrada možemo pisati **bilo koje naredbe** (većina ih završava znakom ; (ovisno o **sintaksi** tih naredbi))
-objašnjenje **funkcioniranja naredbe**:
a) na **početku while petlje** provjerava se da li je **zadovoljen uvjet**
b) ako je **uvjet ispunjen**, petlja se **izvršava**, tj. izvršavaju se **naredbe u vitičastim zagradama**
c) nakon toga ponovo se **vraćamo** na **početak petlje** i **provjeravamo uvjet**
d) ukoliko je uvjet **neispunjen**, while petlja **završava**
e) iz objašnjenja je očito da se naredbe u petlji **ne moraju nijednom izvršiti**, ako je **uvjet neispunjen** pri nailasku na petlju
-to je stoga što je **provjera uvjeta na početku petlje**
-**while petlja** često se koristi za **kontrolu unosa podataka tipkovnicom** (npr. program tako dugo traži ponavljanje unosa, dok nije zadovoljen zadani uvjet), kod rada s **datotekama**, pri **ispisu** i sl.
-pomoću **while** petlje može se realizirati **for petlja** (određeni broj ponavljanja), ali to se **rijetko** koristi
-primjer: U programu unesite pozitivni cijeli broj i ispišite ga na ekranu. Ukoliko je broj negativan, ponavljajte upis. Upotrijebite **while** petlju.
-rješenje:

```
#include <cstdlib>
#include <iostream>

using namespace std;

int main(int argc, char *argv[])
{
   int broj;
   broj=-1; //inicijalizacija varijable
   while (broj<0)
         {
          cout<<"Unesite pozitivan cijeli broj-->";
```

```
        cin>>broj;
        cout<<endl;
       }
   cout<<"Unijeli ste broj "<<broj<<endl;
   system("PAUSE");
   return EXIT_SUCCESS;
}
```

-analiza rješenja: prije nailaska na **while** petlju napravili smo **inicijalizaciju** varijable **broj** iznosom za koji ćemo **ući** u **while** petlju (bilo koji **negativni** cijeli broj, ovdje je to -1)
-**uvjet petlje** postavljen je tako da se **ponavlja za krivi unos**, ovdje je to za **negativni** unešeni broj
-pri prvom unosu **nenegativnog broja** (≥0), **uvjet** na početku **while** petlje **nije zadovoljen** i **ponavljanje prestaje**, a to je i traženo
-primjetite da ovdje **nije** bilo potrebno upotrijebiti **if** naredbu za **provjeru** da li je unešeni broj **pozitivan**, jer se to provjerava u **uvjetu** na početku **while** petlje
-primjer: Upotrebom **while** petlje realizirajte beskonačnu petlju.
-rješenje:

```
#include <cstdlib>
#include <iostream>

using namespace std;

int main(int argc, char *argv[])

{
   while (1)
        {
        }
   system("PAUSE");
   return EXIT_SUCCESS;
}
```

-analiza rješenja: vidljivo je da je upotrijebljen **jednostavan uvjet** zadan vrijednošću **konstante** (1) koji se tumači kao **logička istina**
-umjesto njega mogli smo upotrijebiti logičku konstantu **true** ili neki drugi cijeli broj
-**napomena**: želimo li uštedjeti na **veličini programa**, poželjno je da umjesto broja stavimo **true** (troši **manju količinu memorije** od tipa short int ili int)
-**umjesto konstante** mogli smo staviti neki **uvjet** koji **uvijek vrijedi** (npr. 2>1), ali time **trošimo memoriju** za unos dvije konstante i operatora, a osim toga program je **sporiji** i zbog izračunavanja izraza

2.27. 3.2. Petlja do-while

-mada se u ovoj petlji koristi ključna riječ **while** kao i kod prijašnje petlje, **nemojte brkati** njihovu **sintaksu** i način **funkcioniranja**
-**sintaksa do-while** (prevedeno: **ponavljaj sve dok**) petlje je slijedeća:

```
do
  {
    naredbe;
  }
while (uvjet);
```

-**objašnjenje sintakse**: obavezno se pišu **do**, vitičaste **{}** zagrade, **while**, okrugle **()** zagrade i znak **;**
-uvjet može biti bilo što (kao kod if-a), s time da se može tumačiti kao logička vrijednost (true false)
-dakle, uvjet može biti:

a) **uvjet** može biti bilo što (kao kod **if-a**), s time da se može tumačiti kao **logička vrijednost** (**true** ili **false**)

-dakle, **uvjet** može biti:
1.) **konstanta** (korisno samo za dobivanje **beskonačne petlje**)
2.) **ime varijable** (**cjelobrojna** ili **logička** varijabla čija se vrijednost u programu **mora mijenjati**, jer inače dobivamo **beskonačnu petlju**)
3.) **izraz** koji daje **logičku** ili **cjelobrojnu** vrijednost (**0** se tumači kao **false**, **ostalo** kao **true**)
-uobičajeno se koriste **operatori usporedbe** (npr. <, >, <=, >=)

b) unutar **vitičastih** zagrada možemo pisati **bilo koje naredbe** (većina ih završava znakom **;** (ovisno o **sintaksi** tih naredbi))

-objašnjenje funkcioniranja naredbe:

a) **prvo** se izvrše sve **naredbe** u **vitičastim** zagradama, a **na kraju** do-while petlje **provjerava** se da li je **uvjet ispunjen**
b) ako je **uvjet ispunjen**, petlja se **ponavlja**, tj. izvršavaju se naredbe u vitičastim zagradama
c) nakon toga ponovo se **vraćamo** na **kraj** petlje i **provjeravamo uvjet**
d) ukoliko je **uvjet neispunjen**, do-while petlja **završava**
e) iz objašnjenja je očito da se **naredbe** u petlji moraju **barem jednom izvršiti** (čak i ako je **uvjet neispunjen** pri prvom prolasku kroz petlju)
-to je stoga što je **provjera uvjeta na kraju petlje**

-valja uočiti **suštinske razlike kod while i do-while petlje**:

a) **while** petlja **prvo provjerava uvjet**
b) **do-while** petlja **prvo izvršava naredbe** u tijelu petlje
c) **while** petlja **ne mora** nijednom **izvršiti** naredbe u tijelu petlje
d) **do-while** petlja mora **barem jednom izvršiti** naredbe u tijelu petlje
e) **while** petlju zovemo **petljom s provjerom na vrhu** (engl. **loop with top check**), a **do-while** petlju **petljom s provjerom na dnu** (engl. **loop with bottom check**)

-**do-while** petlja često se koristi za **kontrolu unosa** podataka **tipkovnicom** (npr. program tako dugo traži ponavljanje unosa, dok nije zadovoljen zadani uvjet), kod rada s **datotekama**, pri **ispisu** i sl.

-pomoću **do-while** petlje može se realizirati **for** petlja (određeni broj ponavljanja), ali to se **rijetko** koristi

-primjer: U programu unesite **pozitivni cijeli** broj i ispišite ga na ekranu. Ukoliko je broj **negativan**, ponavljajte upis. Upotrijebite **do-while** petlju.

-rješenje:

```
#include <cstdlib>
#include <iostream>

using namespace std;

int main(int argc, char *argv[])
{
   int broj;
   do
      {
      cout<<"Unesite pozitivan cijeli broj-->";
      cin>>broj;
      cout<<endl;
      }
   while (broj<0);
   cout<<"Unijeli ste broj "<<broj<<endl;
   system("PAUSE");
   return EXIT_SUCCESS;
}
```

-analiza rješenja:

-**uvjet petlje** postavljen je tako da se **ponavlja za krivi unos**, ovdje je to za **negativni** unešeni broj

-pri prvom unosu **nenegativnog** broja (≥0), uvjet na kraju do-while petlje **nije zadovoljen** i **ponavljanje prestaje**, a to je i traženo
-primjetite da ovdje nije bilo potrebno upotrijebiti **if** naredbu za provjeru da li je unešeni broj pozitivan, jer se to provjerava u samoj **strukturi** petlje
-uočite da je u istom zadatku **do-while** petlja bila **brža i time efikasnija** od **while** petlje (**nije** bila **potrebna inicijalizacija** varijable)
-primjer: Upotrebom **do-while** petlje realizirajte **beskonačnu** petlju.
-rješenje:

```
#include <cstdlib>
#include <iostream>

using namespace std;

int main(int argc, char *argv[])

{
   do
      {
      }
   while (1);
   system("PAUSE");
   return EXIT_SUCCESS;
}
```

-analiza rješenja: vidljivo je da je upotrijebljen **jednostavan uvjet** zadan vrijednošću **konstante** (1) koji se tumači kao **logička istina**
-umjesto njega mogli smo upotrijebiti logičku konstantu **true** ili neki drugi cijeli broj
-**napomena**: želimo li uštedjeti na **veličini programa**, poželjno je da umjesto broja stavimo **true** (troši **manju količinu memorije** od tipa short int ili int)
-**umjesto konstante** mogli smo staviti neki **uvjet** koji **uvijek vrijedi** (npr. 2>1), ali time **trošimo memoriju** za unos dvije konstante i operatora, a osim toga program je **sporiji** i zbog izračunavanja izraza

2.27.4. Petlje while i do-while - uvježbavanje

-cilj ovog predavanja je **uvježbati** upotrebu **while** i **do-while** petlji
-primjer: Pomoću **while** petlje realizirajte petlju s **pet** ponavljanja. U petlji ispisujte broj prolaska kroz nju.
-rješenje:

```
#include <cstdlib>
#include <iostream>

using namespace std;

int main(int argc, char *argv[])

{
   int brojac;
   brojac=1;//početno stanje ujedno je i početni broj prolaska kroz petlju
   while (brojac<6)
         {
           cout<<"Prolazim "<<brojac<<". put kroz petlju."<<endl;
           brojac++;//povećavamo brojač za 1
         }
   system("PAUSE");
```

```
   return EXIT_SUCCESS;
}
```
-analiza rješenja: uočavamo da smo za **početnu** vrijednost varijable **brojac** odabrali **1**, jer je to ujedno i **broj prolaska** kroz petlju koji moramo ispisati
-**uvijek** nastojimo odabrati **početne vrijednosti** varijabli tako da nam je **program čim jednostavniji, brži i/ili kraći**
-**uvjet** petlje postavljen je tako da se **kraće** zapiše (brojac<6), mada potpuno iste osobine programa dobivamo i uvjetom brojac<=5, ali tipkamo jedan znak više
-mada, moglo bi se reći da je pristup s **<=** **intuitivniji**, jer on automatski **definira gornju granicu** („**vidimo je**“), dok u slučaju s **<** ipak trebamo uložiti neki dodatan minimalni intelektualni napor
-u petlji **povećavamo** brojac za 1 (slično kao u for petlji)
-vidimo da je **while** petlja **univerzalnija** od **for** petlje, jer pokriva funkcioniranje i for petlje, dok for petlja ne može funkcionirati kao while petlja
-primjer: Upotrebom **while** petlje osigurajte unos **pozitivnog** (cijelog) broja **većeg** of 1. Potom od broja 1 do unešenog broja ispišite sve **neparne** brojeve (pomoću **while** petlje).
-rješenje:
```
#include <cstdlib>
#include <iostream>

using namespace std;

int main(int argc, char *argv[])

{
   int brojac, broj;
   broj=0;
   while (broj<2)
         {
          cout<<"Unesite cijeli broj veći od 1-->";
          cin>>broj;
         }
   brojac=1;
   while (brojac<=broj)
        {
         cout<<brojac<<endl;
         brojac=brojac+2;//povećavamo brojač za 2 (ispis neparnih brojeva)
        }
   system("PAUSE");
   return EXIT_SUCCESS;
}
```
-analiza rješenja: **prva while** petlja čeka **unos** broja **većeg** od 1, dok u **drugoj** tražimo da provjerava da li smo **dosegli** tu granicu (počevši od 1), uz **povećanje brojača** za 2 i **ispis** vrijednosti
-primjer: Upotrebom **Euklidova algoritma** odredite **najveći zajednički djelitelj** dva unešena **pozitivna cijela** broja **a** i **b**.
-**napomena**: **Euklidov algoritam** zadan je relacijom:

M(a,b)=M(b,r) koja se ponavlja dok r nije 0.

-tu je **M()** oznaka za **najveći zajednički djelitelj** (to je **najveći cijeli broj kojim su djeljiva oba broja**)

-**primjer** upotrebe **Euclidova algoritma** za brojeve 120 i 36:

M(120, 36) = ?

120 : 36 = 3 i ostatak **12**

M (120, 36) = M (36, **12**)

36 : 12 = 3 i ostatak 0. Kada je ostatak 0, prekidamo postupak (**rješenje je zadnji ostatak dijeljenja različit od 0**), pa je zato:

M (120, 36) = M (36, 12) = 12

-rješenje:

```
#include <cstdlib>
#include <iostream>

using namespace std;

int main(int argc, char *argv[])
{

//Euklidov algoritam zadan je s:
   //a) najveći zajednički djelitelj a i 0 je a;
   //b) najveći zajednički djelitelj a i b je isti kao
       //najveći zajednički djelitelj b i a mod b
  int a, b, pomoc;
  cout << "Unesi dva pozitivna cijela broja-->";
  cin >> a >> b;
  while (b != 0)
      {
       pomoc = a % b;
       a = b;
       b = pomoc;
      }
   cout<<"Najveci zajednicki djelitelj je "<<a<<endl;
   system("PAUSE");
   return EXIT_SUCCESS;
}
```

-analiza rješenja: vidljivo je da se petlja **while ponavlja** sve dok **b nije 0** (**b!=0**)
-u petlji se (po algoritmu) izračunava **ostatak dijeljenja a s b**, s time da se nakon toga **zamijeni a** s **b**, a **b** s ostatkom dijeljenja **a** s **b**
-rješenje je **zadnji ostatak dijeljenja**, a to je **a**

2.27.5. Naredbe break i continue u petljama, naredba goto

-do sada **opisano ponašanje** svih triju **petlji** možemo **promijeniti** upotrebom naredbi **break** i **continue**

2.27.5.1. Naredba break

-naredbu **break** do sada smo upoznali u **switch** naredbi
-u **svim petljama** naredba **break prekida izvođenje petlje u kojoj se nalazi**
-naredba se **piše** u obliku:

break;

-upotreba naredbe **break** uglavnom se nastoji **izbjeći** drukčijim **struktuiranjem programa**, jer ona **ruši preglednost** (i **struktuiranost**) programa
-naredba **break** uglavnom se koristi za reagiranje na **posebne slučajeve** odvijanja programa
-primjer: U **beskonačnoj** petlji unosite **pozitivni cijeli** broj i odredite te ispišite **ostatak** njegova dijeljenja s 32. Kada je prvi put taj ostatak jednak 29, prekinite petlju i završite program.
-rješenje:

```
#include <cstdlib>
#include <iostream>

using namespace std;

int main(int argc, char *argv[])
{
 int broj, ostatak;
 while (1)
      {
       cout << "Unesite pozitivan cijeli broj-->";
       cin >> broj;
       cout<<endl;
       ostatak=broj % 32;
       if (ostatak==29)
        {
         break;
        }
       else
        {
         cout<<ostatak<<endl;
        }
      }
   system("PAUSE");
   return EXIT_SUCCESS;
}
```

-analiza rješenja: u **beskonačnoj while** petlji unosimo broj, izračunavamo **ostatak** njegova dijeljenja s 32 te ako je on **različit** od 29, ispisujemo ga, inače **prekidamo** petlju naredbom **break** i završavamo program

2.27.5.2. Naredba continue

-naredba **continue** uzrokuje **skok u petlji na njezin kraj** (**prekače naredbe između continue i kraja petlje**), ali se **petlja potom dalje nastavlja**
-slično kao i kod naredbe break, naredba continue trebala bi se **čim manje koristiti** zbog istih razloga
-naredbu **continue** u potpunosti možemo **izbjeći naredbom if** i **drukčijim struktuiranjem programa**
-primjer: U **beskonačnoj** petlji unosite **cijele pozitivne** brojeve. Ispisujte sve one koji su djeljivi sa 7, a ostale ne ispisujte.
-rješenje:

```
#include <cstdlib>
#include <iostream>

using namespace std;

int main(int argc, char *argv[])
{
 int broj, ostatak;
 while (1)
      {
```

```
        cout << "Unesite pozitivan cijeli broj-->";
        cin >> broj;
        cout<<endl;
        ostatak=broj % 7;
        if (ostatak!=0) //broj nije djeljiv sa 7
         {
          continue;
         }
          cout<<broj<<endl;
       }
   system("PAUSE");
   return EXIT_SUCCESS;
}
```

-analiza rješenja: u **beskonačnoj while** petlji unose se **cijeli pozitivni** brojevi i računa njihov **ostatak** dijeljenja sa 7
-u petlji pomoću **if** naredbe provjeravamo da li **broj nije djeljiv** sa 7 (ostatak!=0) i u tom slučaju izvršava se naredba **continue** koja **preskače ispis** i ide na **kraj petlje**
-kada je broj djeljiv sa 7, on se ispisuje
-isti primjer smo mogli napisati preglednije i logičnije **bez upotrebe naredbe continue** na ovaj način:

```
 int broj, ostatak;
 while (1)
       {
        cout << "Unesite pozitivan cijeli broj-->";
        cin >> broj;
        cout<<endl;
        ostatak=broj % 7;
        if (ostatak==0) //broj je djeljiv sa 7
          {
            cout<<broj<<endl;
          }
       }
```

-vidimo da smo **uvjet u if** naredbi promijenili u **suprotan**, te je sada **logičniji rad programa**

2.27.5.3. Naredba goto

-naredba **goto** nije nužno vezana uz programske petlje, ali upravlja **odvijanjem** programa
-ona **definira skok programa** na mjesto na kojem je **odgovarajuća oznaka** (**labela**) **skoka** (engl. **label**)
-**labela** ima **odgovarajuće ime** (**različito** od ostalih imena varijabli, konstanti i sl. u programu) praćeno znakom **:** (**bez razmaka**, npr. skok: ili prvo:)
-**sintaksa goto** naredbe:

```
goto labela;
naredbe;
labela: naredbe;
```

-naredba **goto preskače** sve naredbe **između riječi goto i labele** koja je **navedena iza goto**
-naredba **goto** nastoji se **potpuno izbjeći**, jer **ruši dobru struktuiranost programa**
-**izbjeći** ju možemo npr. **upotrebom funkcija ili if naredbi**
-primjer: Upotrebom **goto** naredbe napišite program koji učitava **pozitivne cijele** brojeve, te ako je unešeni broj veći od 10, prekida program, a inače ispisuje unešeni broj.
-rješenje:

```
#include <cstdlib>
#include <iostream>
```

```
using namespace std;

int main(int argc, char *argv[])
{
  int broj;
  while (1)
        {
         cout << "Unesite pozitivan cijeli broj-->";
         cin >> broj;
         cout<<endl;
         if (broj > 10)
          {
           goto kraj;
          }
           cout<<broj<<endl;
        }
    kraj:    //labela skoka
    system("PAUSE");
    return EXIT_SUCCESS;
}
```

-analiza rješenja: u **beskonačnoj while** petlji unosimo brojeve, **if** naredbom provjeravamo da li je unešeni broj **veći** od 10, te tada pomoću **goto** naredbe **prekidamo petlju**

-isti primjer može se bolje riješiti **bez goto** naredbe:

```
#include <cstdlib>
#include <iostream>

using namespace std;

int main(int argc, char *argv[])
{
  int broj, i=1;
  while (i==1)
        {
         cout << "Unesite pozitivan cijeli broj-->";
         cin >> broj;
         cout<<endl;
         if (broj > 10)
          {
           i=2; //neka vrijednost koja nije 1 te prekida while petlju
          }
          else
          {
           cout<<broj<<endl;
          }
        }
    system("PAUSE");
    return EXIT_SUCCESS;
}
```

-analiza rješenja: **petlja** je **beskonačna**, ali **uvjet** je dan izrazom u kojem imamo varijablu **i** kojoj smo dodijelili **početnu vrijednost** koja **zadovoljava ulazak u petlju**

-pomoću **if-else** naredbe mijenamo **i** u **bilo koju vrijednost različitu od 1** kako bismo **prekinuli** beskonačnu petlju

2.28. Polja

-