1. Osnove programiranja

1.1. Programiranje i programski jezici

1.1.1. Uvod

-pojmovi **program** i **programiranje** prisutni su danas na svakom koraku
-mada se značenje tih pojmova smatra razumljivim, **postupak programiranja** to nije
-danas smo okruženi **složenim elektroničkim uređajima** (računalima, mobitelima,…), za koje upotreba i upute za rad uključuju postupak **programiranja**
*-na primjer, nove perilice rublja imaju više **senzora** (npr. temperature vode, razine vode i sl.) pa omogućuju sve **učinkovitije**, a time i **štedljivije** pranje, ali potrebno ih je pravilno **programirati***
-jednostavan primjer programiranja obrađivali smo prošle godine u **Excelu** kada smo učili korištenje nekih od jednostavnijih **funkcija** (npr. min, max, average, if, sum,...)
-**Excel** je aplikacija koja se u nekim zanimanjima koristi intenzivno (**špediteri**, ekonomisti, **prometni tehničari**,...), pa znati **osnove programiranja** može biti vrlo korisno
-osnove programiranja korisno je znati za primjenu u tzv. **skriptnim jezicima** koji omogućuju **automatiziranu primjenu** nekih **programskih alata**
*-npr. u **Photoshopu** se obrada fotografija može automatizirati tako da se svakoj fotografiji podesi automatski kontrast i balans boja*
-kod programiranja **naše iskustvo i način razmišljanja** često **nisu usklađeni s načinom programiranja uređaja i naprava**
-zato se moramo kod programiranja **prilagoditi** traženom objektu (računalo, perilica, mobitel,…)
-slijedi nekoliko **definicija općih pojmova** vezanih uz programiranje (koje smo učili u 1. razredu)
-**programska oprema** – **skup svih programa instaliranih** na nekom računalu
-**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,…)
-**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** – postupak kojim se napisani program **priprema za rad**
-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**
-programiranje je jedan od **najstresnijih** poslova
-problem kod programiranja je postojanje **ogromnog broja programskih jezika** od kojih svaki traži dosta **dugo učenje i vježbanje**
-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 je svladavanje 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**
-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**
-to je **najniža razina** programiranja, 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**

b) **simbolički jezici niske razine (asembleri)**
-nastali su kako bi ljudima **olakšali programiranje**, jer ljudi lakše **pamte simbole**, nego binarne brojeve
-takvi jezici još uvijek su **prilagođeniji računalu**, nego programeru koji ih koristi
-**asembler** (engl. **assembler**) je simbolički jezik u kome je svaka **binarna naredba strojnog jezika predočena odgovarajućim simbolom** (najčešće **kombinacijom nekoliko slova**)
-primjer:
*ADD - zbroji (engl. **add**)*
*SUB - oduzmi (engl. **subtract**)*
*CMP - usporedi (engl. **compare**)*
-**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**
-primjer asemblerske naredbe:
ADD A, #100
-programi pisani u asembleru su **čitljiviji i lakši za razumijevanje** od binarnog zapisa, ali ih je još uvijek **vrlo teško pisati i ispravljati**
-oni **ovise o vrsti i unutarnjoj građi računala** (procesoru)
-imaju **veliku brzinu izvršavanja** (rade brže od istih programa pisanih u jezicima više razine)
-**rijetko** se koriste na računalima

c) **viši programski jezici**
-da bi se **olakšalo programiranje** i da bi se isti program mogao izvršavati na **različitim računalima** (procesorima) stvoreni su **simbolički jezici visoke razine** (naredbe su **nalik govornom jeziku**, **lakše 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 **neovisni su o računalu** 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**
-mogu se koristiti **za bilo koje zadatke**
b) **jezici prilagođeni određenoj vrsti problema**
-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
-programski jezik C **nema mnogo ključnih** (rezerviranih) **riječi**, samo njih 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** koji mu je dao podršku za tzv. **objektno orijentirano programiranje** (engl. **object-oriented programming**) te ga nazvao programskim jezikom **C++**
-jezik **C++** danas je jedan od **najkorištenijih programskih jezika opće namjene**
-program pisan u **jeziku više razine** mora se prije pokretanja pomoću **jezičnog prevoditelja pretvoriti u oblik pogodan za određeni procesor**
-**jezične prevoditelje** dijelimo u dvije grupe:
a) **interpreteri** (engl. interpreter)
-naredbe višeg programskog jezika pretvaraju u naredbe strojnog jezika **tijekom izvršavanja** (takav je npr. **QBasic**)
b) **kompajleri** (engl. compiler)
-naredbe 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**

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

b) **objektno orijentirane** (engl. **object-oriented language**)
-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**
-da bi računalo riješilo zadatak, zadatak treba **pretvoriti** u oblik koji uključuje samo **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 potrebno **više pomoćnih postupaka**
-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**
-**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**
-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**
-**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 nastoji pronaći algoritam koji **najbrže, najučinkovitije i najsigurnije** dovodi do rezultata
-**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** izgleda kao program, 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
-pseudokôd treba biti napisan tako da programer može na temelju njega **napisati program u bilo kojem programskom jeziku**
-svaki programski jezik ima **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**)
-**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
-algoritam izvršavanjem od početka do kraja **uvijek za iste ulazne podatke mora dati isti rezultat**
-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
-**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 **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

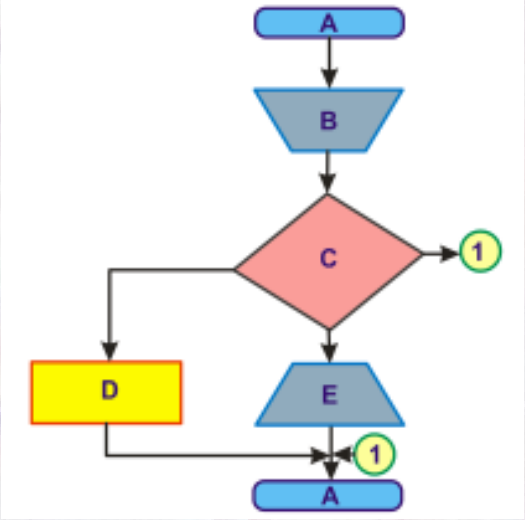
-ovakvim korištenjem varijabli program je postao dosta **nepregledniji**, ali uštedjeli smo na količini memorije potrebne za izvršavanje programa
4.) Napišite algoritam koji će odrediti manjeg od dva učitana broja x i y te ga ispisati na ekranu, a većeg umanjiti za 3 i taj iznos ispisati na ekranu. Ukoliko 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 **sporiji**, 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

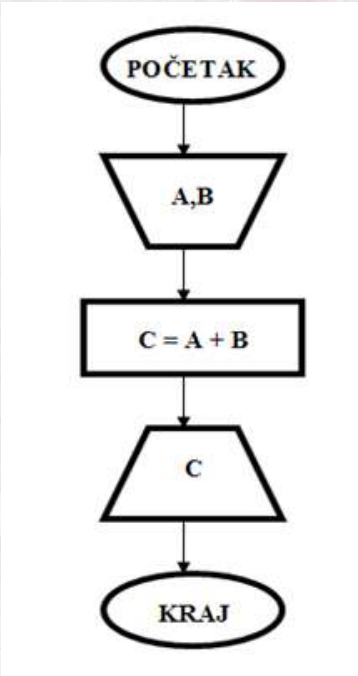
-on nam služi za **grafičko predočavanje algoritama**, pri čemu je prikazom pomoću **jednostavnih grafičkih simbola spojenih usmjerenim crtama** (strelicama) bitno **lakše pratiti** način funkcioniranja algoritma
-**usmjerene crte** (**strelice**) pokazuju **tijek rješavanja zadatka**
-dijagram tijeka **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**
-na idućoj slici su slovima i brojkom označeni **najčešće korišteni simboli u dijagramu tijeka**



-**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**) 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**)
-**uvjet** 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**)

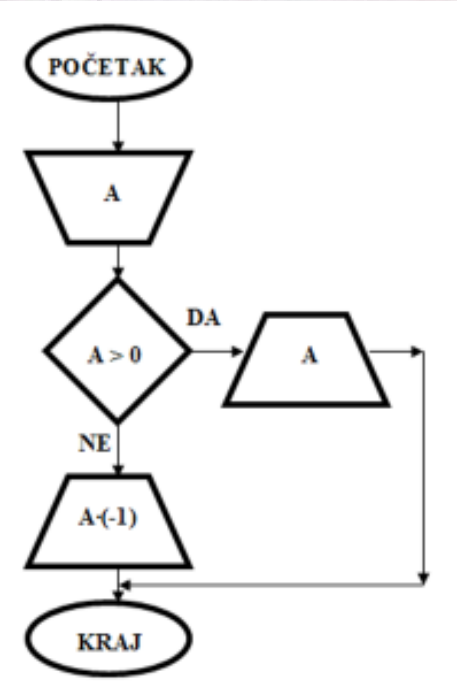
-ukoliko **iz romba izlaze više od dvije strelice**, tada se radi o tzv. **višestrukom odabiru** (engl. **multiple choice**)
-tu se **pitanje u rombu** svodi na to da li je **neki podatak jednak nekoj unaprijed zadanoj cijeloj vrijednosti** (npr. 1, 3, 23, -89, 8990,...), pri čemu se **vrijednost** za 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**
-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

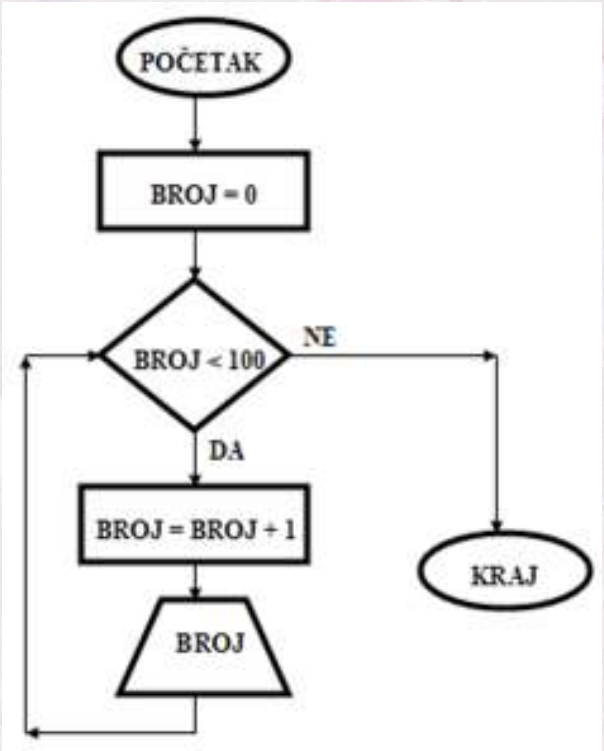


-**primjer 2**: Ispiši apsolutnu vrijednost unešenog broja A.
-algoritam:

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**



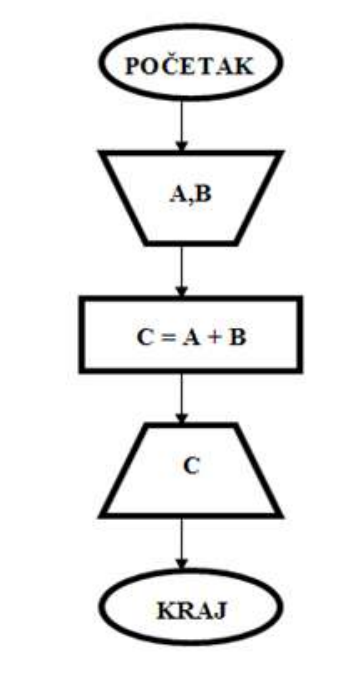
1.5. Programske strukture

-**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

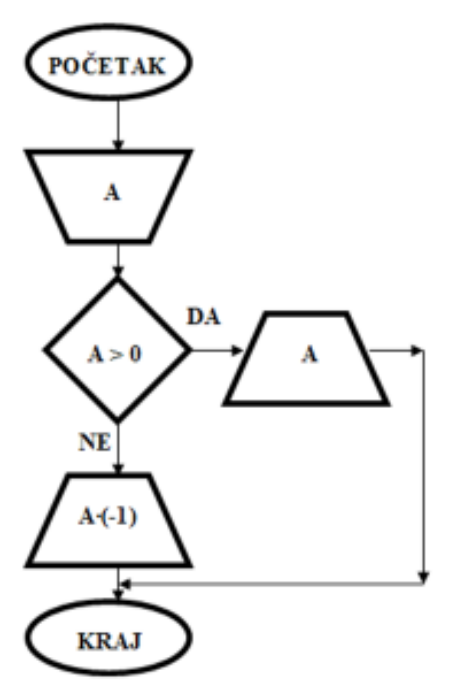
početak
  upiši brojeve A i B
  C=A+B
  ispiši C
kraj



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**)
-**uvjet** 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,...)

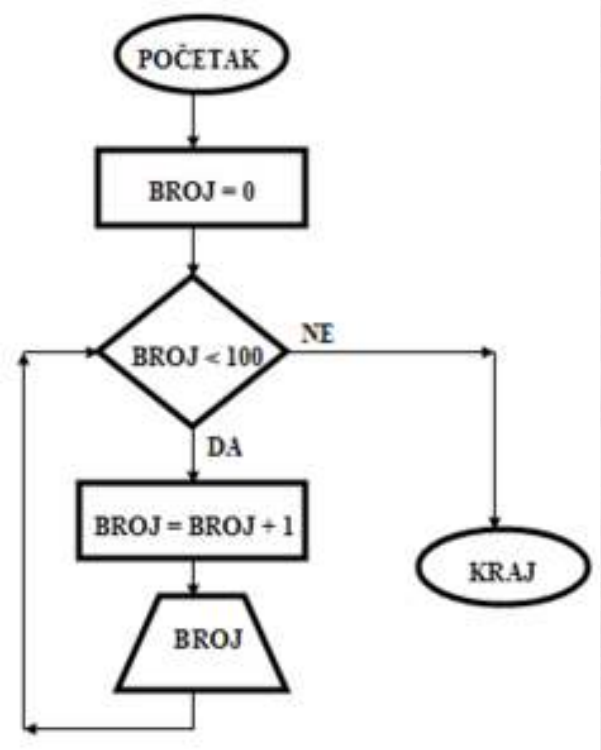
-obično se za slučaj **višestrukog odabira** pitanje u algoritmu zadaje u obliku **ako je** pa se zatim **ispod redom navode vrijednosti varijable** i radnje koje se poduzimaju
-primjer:
početak
upiši A
ako je
A=1
ispiši A
A=234
*A=A*2*
ispiši A
A=-23
A=A+2
ispiši A
kraj
-slijedi prikaz primjera **grananja** 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**) 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 **pregledniji 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**
-**načela pisanja** pseudokoda odnose se 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**
-varijabli se **vrijednost pridružuje** pomoću **operatora pridruživanja** koji se označava **kombinacijom znaka dvotočke i znaka jednakosti** (:=)

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 napisan pomoću **uvlačenja** dijelova koda puno je preglednij:

```
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**

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
```

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**
-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	!

-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

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**

2. Izrada programa u programskom jeziku C++

2.1. Osnovni pojmovi i nastanak programa

-jezik **C++** je jezik **opće namjene**
-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**)
-**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
-**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**
-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**
-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)
-**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**
-posao programera znatno **olakšan objedinjavanjem svih datoteka** vezanih za jedan program u **projekt** (engl. **project**)
-**projekt** je **skup međusobno povezanih datoteka**
-**projekt** se **stvara** aktiviranjem naredbe **Datoteka->Nova->Projekt** (simbol **->** označava da se radi o naredbama u meniju)
-**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**
-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**)
-**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.

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

-radi lakšeg pamćenja uvodi se označavanje varijabli **simboličkim imenima**, tj. pomoću njezina naziva ili kraće - pomoću **imena varijable**
-ime varijable često se puta naziva i **identifikatorom**
-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**)
b) ime mora **početi slovom ili znakom potcrtavanja** (_), tj. **ne smije početi znamenkom**

-**pravila za određivanje simboličkog imena**:

a) **ne smije** se rabiti **razmak** kao dio imena varijable
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**
-od svih ključnih riječi programskog jezika C++ samo riječi **auto**, **do** i **operator** koriste se u hrvatskom jeziku te njih **ne smijemo koristiti**
-primjer:
do (pogrešno), *do_sada* (dobro), *auto* (pogrešno), *auto1* (dobro), *operator* (pogrešno), *operator1* (dobro)

d) program **razlikuje velika i mala slova**
-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
-kod programiranja trebamo paziti da **ne koristimo preduga opisna imena**
-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

-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;
-**deklariranoj varijabli** se može **pridružiti vrijednost operatorom pridruživanja** (znak **jednako**, =)
-ukoliko se **zadaje početna vrijednost** varijable tada to nazivamo **inicijalizacijom**
-**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
-preporuka je 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
-**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 $\mathbf{\pm 3.4*10^{38}}$ **do** $\mathbf{\pm 1.17*10^{-38}}$
-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**
-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**
-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**
-**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

-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**

-**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**

-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**

-**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; */
```

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**

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**)

2.4. Funkcije

-**funkcije** zamjenjuju **blokove naredbi**, a u program ih ubacujemo **navođenje**m njihova **imena**

-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**

-**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**

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**

-**upotrebom funkcija dobijamo**:

a) **podjelu programa na manje dijelove**

-time program postaje **modularan**

b) program koji je **čitljiviji**

c) **razumljivije ponašanje** programa

-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**

-dozvoljeno je **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);
```

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

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()***
-**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**
-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**

-**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 uneseni broj tipa float upisuje se u varijablu broj
```

-uneseni **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**

-primjer:

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

-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** 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**
-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**)
-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
13

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 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

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**
-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
```

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**

-**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

-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
```

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

-primjer:

```
a=23;
b=17;
```

```
c=a + b; //c je 40
```
-u istom redu mi možemo obaviti **zbrajanje više operanada**
-primjer:
```
a=20;
b=12;
c=6;
d=31;
e=a + b + c + d; //e je 20+12+6+31=69
```

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

-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
```

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

-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
```

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

-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
```

-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
```

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 **%**
-**način pisanja**:

operand1 % operand2

-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
```

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;
```

-primjer:

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

-**zaključak**: ako nismo sigurni u **prioritet** izraza, upotrijebimo **zagrade** i time definiramo **vlastite prioritete** u izračunima

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**

-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;
```

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

-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

-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**)
-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

-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

-ukoliko je broj s **lijeve** strane operatora **veći od ili jednak** broju s **desne** strane, tada je rezultat **true**, inače je **false**
-operator je kombinacija znakova **>=** (**bez razmaka**)
-primjer:

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

2.11.5. Operator jednako

-ukoliko su podaci **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
```

2.11.6. Operator različito od

-ukoliko su podaci **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**
-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)%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)%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)

2.15. Pismena provjera znanja

-