Norwegian Embassy
Sarajevo

EGG
Economic Governance for Growth

UNDP

PRIRUČNIK ZA C++

C++ SADRŽAJ

1 OSNOVE PROGRAMIRANJA

1.1 UVOD

Programiranje je jedno od najatraktivnijih zanimanja u modernom svijetu. Ono što je dobro, potreba za programerima je stalno u porastu. Razlog tome je rapidan tehnološki razvoj svijeta.

Svijet se razvija u smjeru automatizacije svih proizvodnih procesa, počevši od proizvodnje, trgovine, ali i usluge. Svi ti automatizirani procesi ne mogu funkcionisati bez programa koji prave programeri.

U današnje vrijeme postoji puno programskih jezika i često to može predstavljati problem za početnike/ce koji žele da se počnu baviti programiranjem.

Koji jezik odabrati? Koji je najbolji? To su pitanja na koja svako ima svoj odgovor, zavisno od vlastite preferencije. Ali pravi odgovor je da ne postoji najbolji programski jezik, nego svaki programski jezik ima svoju specifičnu svrhu za koju je namijenjen.

Ono što je najbitnije za početnike/ce je da nije bitno koji će programski jezik odabrati kao prvi, jer u svojoj suštini temeljne osnove su iste za sve programske jezike. Razlika je samo u sintaksičkim pravilima.

Nakon što se savladaju osnove, mogu se lako prebaciti na bilo koji drugi programski jezik, bez potrebe da ih uče ispočetka.

C++ je jedan od najpopularnijih programskih jezika svih vremena i može se naći u današnjim operativnim sistemima, grafičkim korisničkim interfejsima i ugrađenim sistemima. Također se koristi za izradu videoigrica.

1.2 ALGORITMI

Riječ algoritam dolazi od latinskoga prijevoda imena arapskoga matematičara Abu Ja'far Muhammad ibn Musa Al-Khwarizmi (Muhamed, otac Jafarov, sin Muse iz Khwarizma). Rođen je 780. godine na području današnjeg Uzbekistana, a umro je 850. godine u Bagdadu. Smatra se ocem algebre - grane matematike koja izučava matematičke operacije i relacije te strukture i koncepte koji iz njih proizlaze.

Algoritam označava precizan i jednoznačan opis rješenja nekoga problema.

Sastoji se iz konačnoga skupa uputa koje potpuno i nedvosmisleno definišu korake koji se trebaju učiniti, kao i njihov redoslijed, s ciljem rješavanja problema. Svaki korak mora biti definisan dopuštenim (izvedivim) instrukcijama. Isti problem može imati više različitih rješenja, pa time i više različitih algoritama.

Od 20. vijeka, pojavom računara, pojam algoritam počinje se koristiti i u računarstvu, a zatim i u drugim područjima.
Algoritam (rješenje nekoga problema) može se opisati na sljedeće načine:

Algoritam (rješenje nekoga problema) može se opisati na sljedeće načine:

- tekstualno
- grafički
- pseudokodom
- kombinacijom grafičkog opisa i pseudokoda
- programskim jezikom

Tekstualni opis algoritma

U tekstualnom se opisu koraci algoritma prikazuju rečenicama govornoga jezika. Prednost je ovakvoga opisa u tome što ga može razumjeti širi krug ljudi. Veliki je nedostatak nepreciznost opisa koja proizlazi iz nepreciznosti samoga govornog jezika.

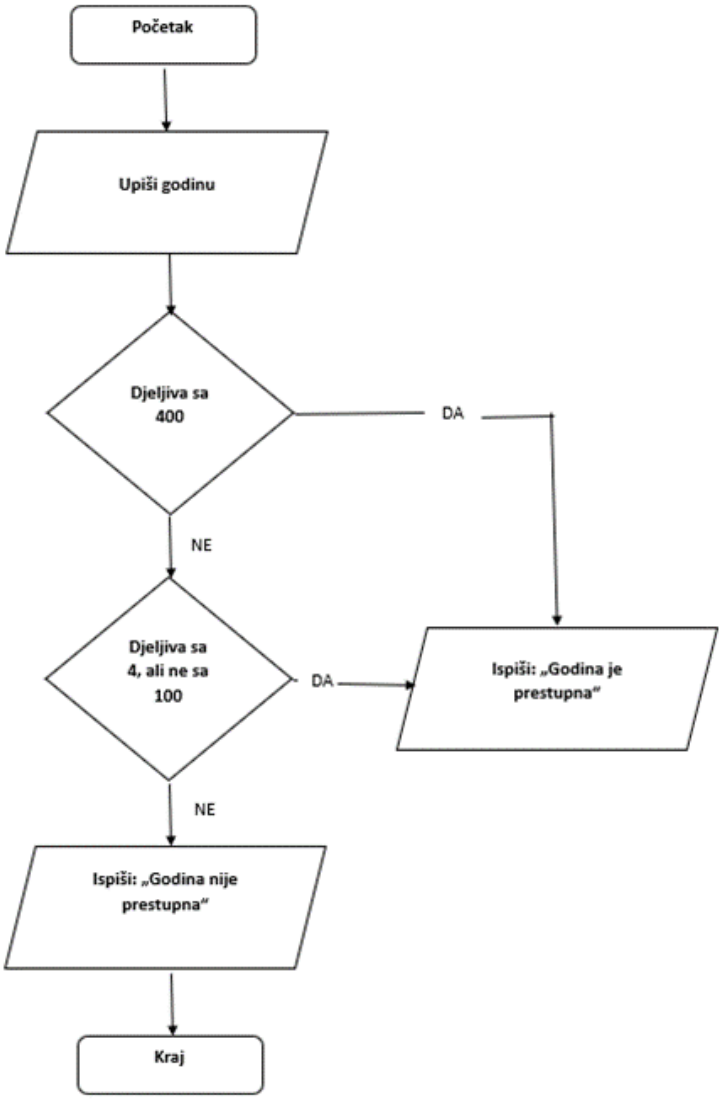
Primjer tekstualnog opisa algoritma - provjera je li godina prestupna:

RB	Koraci algoritma
1	Ako je godina djeljiva s 400 ona je prestupna godina.
2	Ako je godina djeljiva s 4 ali ne i sa 100 ona je prestupna godina.

Tabela 1: Primjer tekstualnog opisa algoritma - provjera je li godina prestupna

Grafički opis algoritma

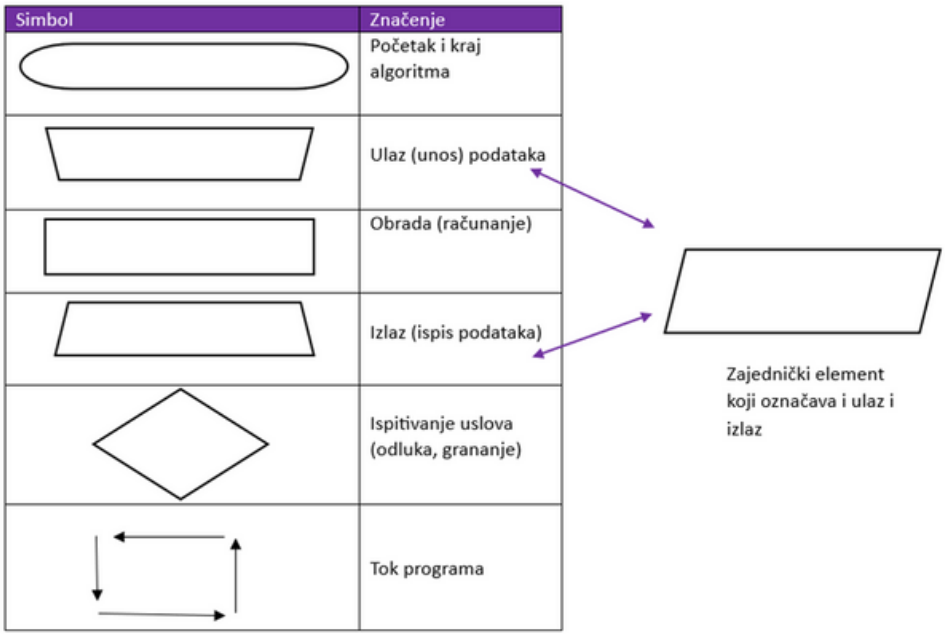
U grafičkom se opisu koraci algoritma predstavljaju grafičkim simbolima. Grafički su simboli međusobno povezani strelicama koje upućuju na redoslijed grafičkih simbola, odnosno redoslijed izvođenja koraka algoritma. Ovaj način opisa algoritma precizniji je od tekstualnoga opisa, ali je za njegovo shvaćanje potrebno poznavanje značenja grafičkih simbola iz kojih se sastoji opis, te je stoga razumljiv užemu krugu ljudi.



Slika 1: Primjer grafičkog opisa algoritma – provjera je li godina prestupna

Grafički opis algoritma (Slika 1) zove se dijagram toka ili blok dijagram.

Blok dijagram je vrsta grafičkog opisa algoritma primjenom unaprijed definisanih grafičkih simbola. Grafički simboli predstavljaju određeni algoritamski korak, a strelice koje ih povezuju, sljedeći algoritamski korak koji se treba izvesti. Blok dijagram se koristi tokom oblikovanja algoritma. Slika 2. prikazuje grafičke simbole koji se koriste pri izradi bloka dijagrama.



Slika 2: Grafički simboli za izradu bloka dijagrama

Algoritam opisan pseudokodom

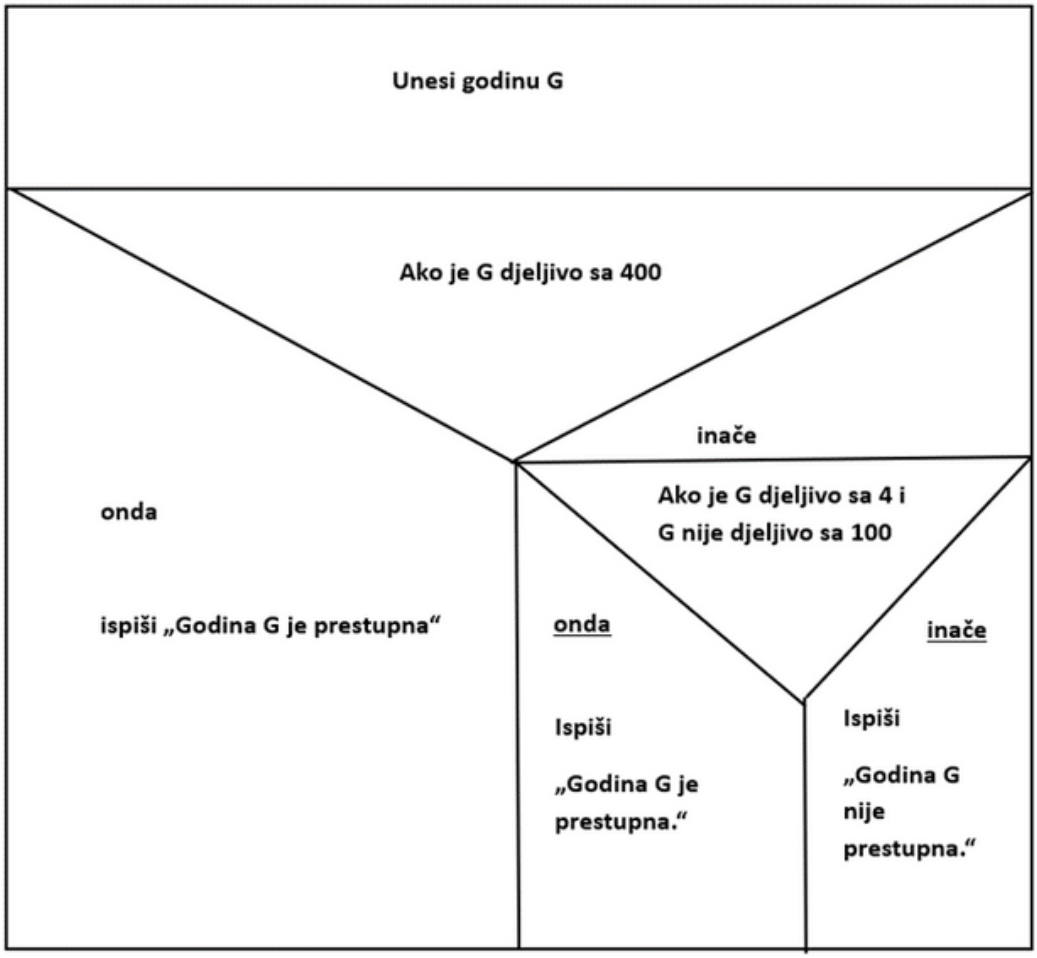
Pseudokod predstavlja formalizirani tekstualni opis algoritma kojim se postiže veća preciznost koja nedostaje klasičnom govornom jeziku. Obično se svaka linija pseudokoda označava rednim brojem. Zbog svog tekstualnog opisa, pseudokod je razumljiv širemu krugu ljudi. Pseudokod predstavlja detaljni opis algoritma primjenom formaliziranog prirodnog jezika. Zbog toga je on čitljiv ljudima, a ne računarima. Koristi se prilikom oblikovanja algoritma, a prije njegove same implementacije u nekome programskom jeziku koji je razumljiv računarima. Pseudokod bi trebao biti potpuno nezavisan od bilo kojeg programskog jezika, odnosno riječi koje se koriste u pseudokodu ne bi trebale biti vezane za neki specifični programski jezik.

RB	Koraci algoritma
1	Unesi godinu G
2	Ako je G djeljivo s 400 onda
3	Ispiši "Godina G je prestupna."
4	Inače ako je G djeljivo s 4 i G nije djeljivo sa 100 onda Ispiši "Godina G je prestupna."
5	Inače
6	Ispiši "Godina G nije prestupna."
7	Kraj Ako jeste

Tabela 2: Primjer algoritma opisanog pseudokodom - provjera je li godina prestupna

Algoritam opisan kombinacijom grafičkoga opisa i pseudokoda

Ovaj opis kombinuje karakteristike grafičkoga opisa i pseudokoda. Slično kao i u grafičkom opisu, postoje posebni grafički simboli kojima se predstavljaju osnovne algoritamske strukture. U te se simbole upisuje pseudokod kojim se detaljnije opisuju koraci algoritma. Time se dodatno povećava preciznost opisa algoritma, ali se sužuje krug ljudi koji ga razumiju, budući da je potrebno poznavati značenje pojedinog grafičkog simbola. Prikazani se opis algoritama naziva strukturogram.



Slika 3: Primjer algoritma opisanog kombinacijom grafičkoga opisa i pseudokoda – provjera je li godina prestupna

Algoritam opisan kombinacijom grafičkoga opisa i pseudokoda

Ovaj opis kombinuje karakteristike grafičkoga opisa i pseudokoda. Slično kao i u grafičkom opisu, postoje posebni grafički simboli kojima se predstavljaju osnovne algoritamske strukture. U te se simbole upisuje pseudokod kojim se detaljnije opisuju koraci algoritma. Time se dodatno povećava preciznost opisa algoritma, ali se sužuje krug ljudi koji ga razumiju, budući da je potrebno poznavati značenje pojedinog grafičkog simbola. Prikazani se opis algoritama naziva strukturogram.

RB	Koraci algoritma
1	#include <conio.h>
2	#include <stdio.h>
3	main(){
4	unsigned int G;
5	printf ("Unesite godinu: ");
6	scanf("%d",&G);
7	if (G%400==0)
8	printf("Godina %d je prestupna.", G);
9	else if (G%4==0 && G%100!=0)
10	printf("Godina %d je prestupna.", G);
11	else
12	printf("Godina %d nije prestupna.", G);
13	getch();}

Tabela 3: Primjer algoritma opisanog programskim jezikom C - provjera je li godina prestupna

Algoritam opisan u tabeli 3 napisan je programskim jezikom C.

1.3 POJMOVI - RAČUNARSKI PROGRAM, PROGRAMSKI JEZIK, PROGRAMIRANJE

Algoritam, dakle, predstavlja opis rješenja nekoga problema. Rješenje nekoga problema (algoritam) se može opisati tekstualno, grafički, pseudokodom, kombinacijom grafičkog opisa i pseudokoda te programskim jezikom. Opisom algoritma nekim programskim jezikom nastaje računarski program.

Računarski program predstavlja algoritam opisan nekim programskim jezikom i time prilagođen izvođenju na računaru. Računari ne mogu funkcionisati bez računarskog programa koji izvode.

Programski jezik je umjetno stvoren jezik koji je namijenjen davanju instrukcija računarima. Ovim se jezikom algoritam može precizno opisati kao niz instrukcija koje izvodi računar.

Programski jezik se sastoji iz dvije osnovne komponente:

- **sintakse** (forme) i
- **semantike** (značenja).

Sintaksa programskog jezika kaže kakva forma instrukcije mora biti, a kako bi je računar razumio (npr. u govornome jeziku je jasno da je riječ "Prgrm" pogrešna – greška sintakse – jer je ispravno "Program"). Spomenute se greške lako otkrivaju (sam računar ih otkriva) i lako ispravljaju.

Semantika je značenje onoga što se programskim jezikom opisalo. Primjera radi, programskim se jezikom opisao izračun prosječne ocjene iz pet predmeta (ocjene su k1, k2, k3, k4 i k5). U jednom se dijelu algoritma nalazi formula za izračun prosječne ocjene koja glasi Prosjek=(k1+k2+k3+k4+k5)/2. Računar će prema navedenoj formuli izračunati prosjek ocjena. Jasno je da formula nije ispravna (dijeli se s dva umjesto s pet). No, računar ovu semantičku pogrešku (pogrešno značenje) neće primijetiti te će dati pogrešan rezultat. Otkrivanje semantičkih pogrešaka u računarskom programu nije jednostavno i postoji posebni alat (engl. debugger) koji ovaj postupak olakšava. Nekim je programskim jezikom algoritam prilagođen izvođenju na računaru i dobiven je računarski program koji je u formi izvornoga koda, tj. u formi čitljivoj ljudima.
Postoje dva osnovna načina na koja računar izvodi računarski program:

- interpretiranjem instrukcija u računarskom programu i
- izvođenjem izvršne verzije računarskog programa.

Pri interpretiranju instrukcija, na računaru mora postojati posebni računarski program - interpreter.

Interpreter učitava računarski program, zatim čita instrukciju po instrukciju, tumači je i izvršava. Budući da tumačenje instrukcija troši vrijeme, interpreteri su spori. Računarski program nije u izvršnom obliku (mašinskom kodu). On može biti u izvornom obliku (izvornom kodu) ili u nekom posebnom obliku potrebnom interpreteru (npr. Bytecode kod JVM – Java Virtual Machine). Pri izvođenju izvršne verzije računarskog programa (ekstenzija EXE), on se treba iz izvornog koda preoblikovati u mašinski kod. Ovo preoblikovanje izvodi posebni računarski program - **kompajler**.

Kompajler učita cijeli računarski program te ga pretvori u poseban oblik koji računar može izravno izvesti, a da ne treba dodatni računarski program. Računarski program u mašinskom kodu nije čitljiv ljudima, već računarima.

Programiranje (računarsko programiranje) je proces oblikovanja, pisanja, testiranja, pronalaženja i uklanjanja greški te održavanja izvornog koda računarskog programa. Osoba koja stvara računarski program programiranjem naziva se programer. Programer polazi od zadanog problema, oblikuje algoritam koji problem rješava, te ga opisuje nekim programskim jezikom. Učenje programiranja uključuje učenje sintakse (forme) nekoga programskog jezika i sticanje osnovnih intuitivnih znanja glede algoritmizacije problema opisanog riječima.

Postoje četiri osnovna pristupa u programiranju:

- proceduralno programiranje
- objektno orijentisano programiranje
- funkcionalno programiranje
- logičko programiranje

Proceduralno programiranje se još naziva i algoritamsko programiranje jer se kod njega rješenje problema opisuje tako da se navode koraci (instrukcije) koje računar treba izvesti. Računarskim programom se daje odgovor na pitanje kako se dolazi do rješenja (imperativno programiranje). Ako je problem složen, onda se on razbija na manje probleme koji se zatim rješavaju. Iz rješenja manjih problema proizlaze procedure. Pozivajući procedure (dakle rješavajući manje probleme), računar rješava glavni problem. Koriste se proceduralni programski jezici kao što su C, Pascal, Basic itd.

Pri **objektno orijentisanom programiranju**, glavni se problem razbija u manje probleme čija su rješenja predstavljena objektima. Objekti imaju stanja i metode. Računar mijenja stanja objekata i poziva njegove metode te na taj način rješava glavni problem. Glavna poteškoća ovog pristupa je uočavanje objekata koji su potrebni za rješavanje glavnog problema. I ovdje se računarskim programom daje odgovor na pitanje kako se dolazi do rješenja (imperativno programiranje). Koriste se objektno orijentisani programski jezici kao što su C++, Java, C# itd.

Funkcionalno programiranje spada u skupinu deklarativnog programiranja u kojem se računaru kaže što treba napraviti (a ne kako, kao kod imperativnog programiranja). Za rješavanje problema izrađuju se funkcije. Koriste se funkcionalni programski jezici kao što su Lisp, Haskell, ML.

Logičko programiranje također spada u skupinu deklarativnog programiranja. Temelji se na matematičkoj logici, odnosno predikatnom računu (npr. ako je broj > 0, onda je broj pozitivan). Koriste se logički programski jezici (npr. Prolog).

1.4 OBJEKTNO ORIJENTISANO PROGRAMIRANJE

Kako je već i navedeno, objektno orijentisano programiranje predstavlja programiranje gdje se glavni problem razbija u manje probleme čija su rješenja predstavljena objektima.

C++ je objektno orijentisani programski jezik koji daje jasnu strukturu programima i omogućava ponovnu upotrebu koda, smanjujući troškove razvoja. C++ je prenosiv i može se koristiti za razvoj aplikacija koje se mogu prilagoditi višestrukim platformama. Stoga je jako popularan među programerima. C++ je blizak programskim jezicima C, C# i Javi, te programerima olakšava prelazak na C++ ili obrnuto.

1.5 PONAVLJANJE

Nakon završetka poglavlja, čitatelju bi trebali biti poznati sljedeći pojmovi:

- Algoritmi
- Tekstualno predstavljanje algoritama
- Grafičko predstavljanje algoritama
- Predstavljanje algoritama pseudokodom
- Predstavljanje algoritama kombinacijom grafičkog opisa i pseudokoda
- Predstavljanje algoritama programskim jezikom
- Računarski program
- Programski jezik
- Sintaksa
- Semantika
- Interpreter
- Kompajler
- Programiranje
- Proceduralno programiranje
- Objektno orijentisano programiranje
- Funkcionalno programiranje
- Logičko programiranje

2 OSNOVE C++ PROGRAMIRANJA

2.1 ALATI ZA RAZVOJ

Za pisanje C++ programa može koristiti CodeBlocks, te je stoga bitno da se poznaju barem osnove ovog razvojnog okruženja. Pod time se misli na kreiranje aplikacije komandne linije te organizaciju programskog koda u projektu.

CodeBlocks je besplatan, open-source cross-platform IDE koji podržava više kompajlera. Razvijen je u C++. IDE je skraćenica od Integrated Development Environment, a riječ je o softveru koji pruža mogućnosti razvoja aplikacija. IDE se obično sastoji od:

- dijela za uređivanje i pisanje koda,
- kompajliranje koda i
- testiranje (Debugging).

Neki popularni IDE alati imaju sva 3 modula u sebi, ali neki nemaju.

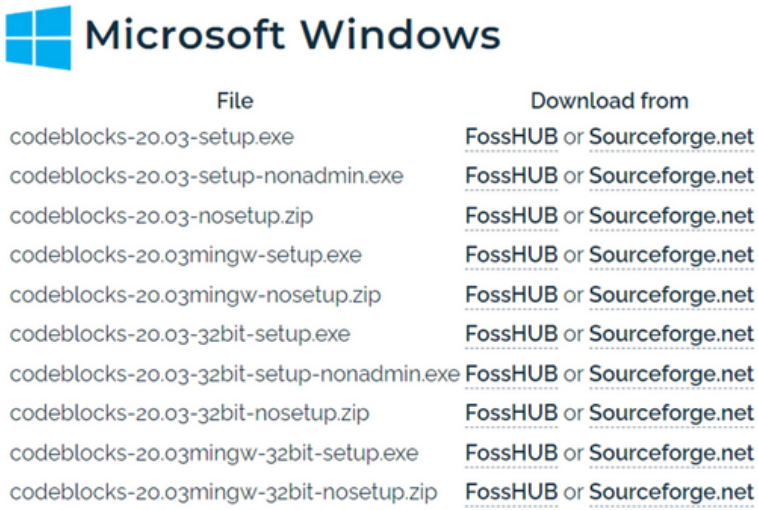
Osnovne komponente alata su:

- Kompajler
- Uređivač koda
- Debugger
- GUI dizajner
- Migracija korisnika/ca
- Projektne datoteke i izgradnja sistema.

2.2 INSTALACIJA PROGRAMA

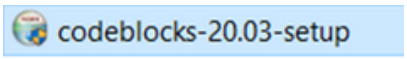
Otiđite na adresu http://www.codeblocks.org/downloads/binaries/.

Otvoriće vam se prozor kao s naredne slike:



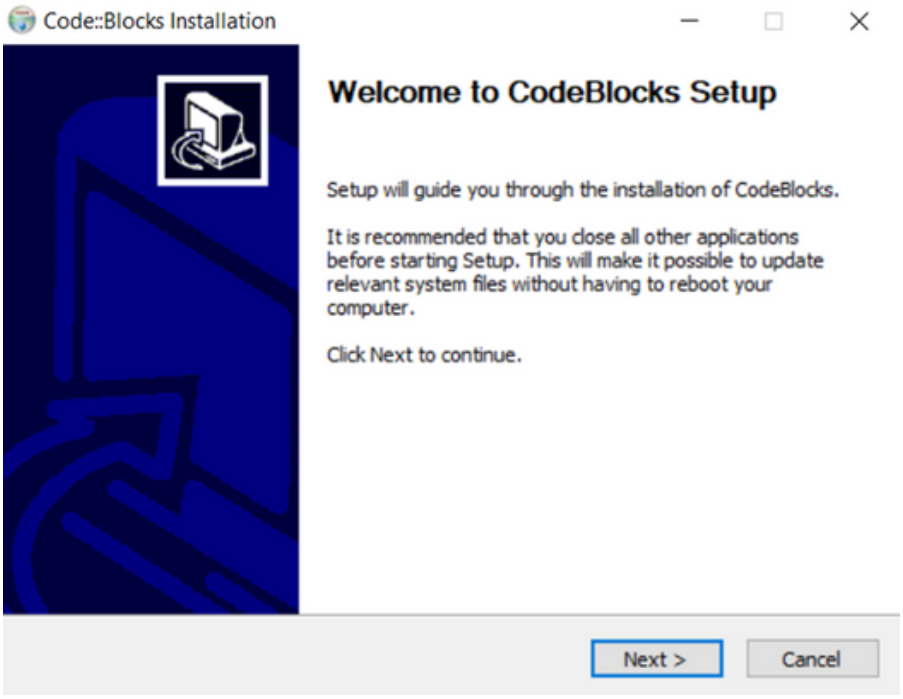
Slika 4: Download CodeBlocks

Kliknite bilo koju od navedene dvije opcije (lakši je download sa FossHUB). Skinuće vam se fajl kao s naredne slike:



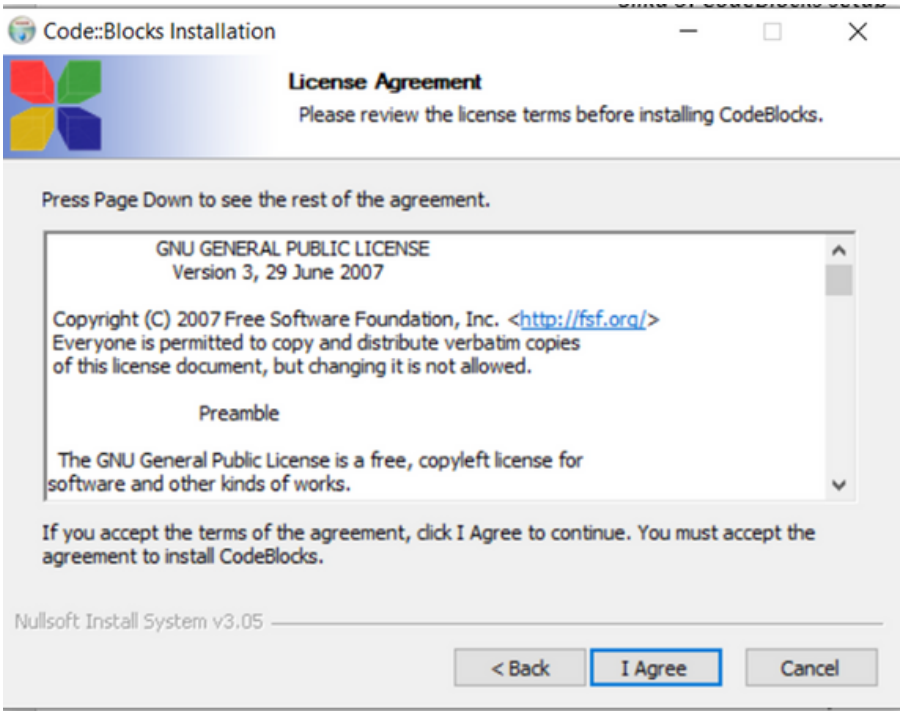
Slika 5: CodeBlocks setup

Pokrenite instalaciju duplim klikom na setup i na sljedećem koraku kliknite *Next*



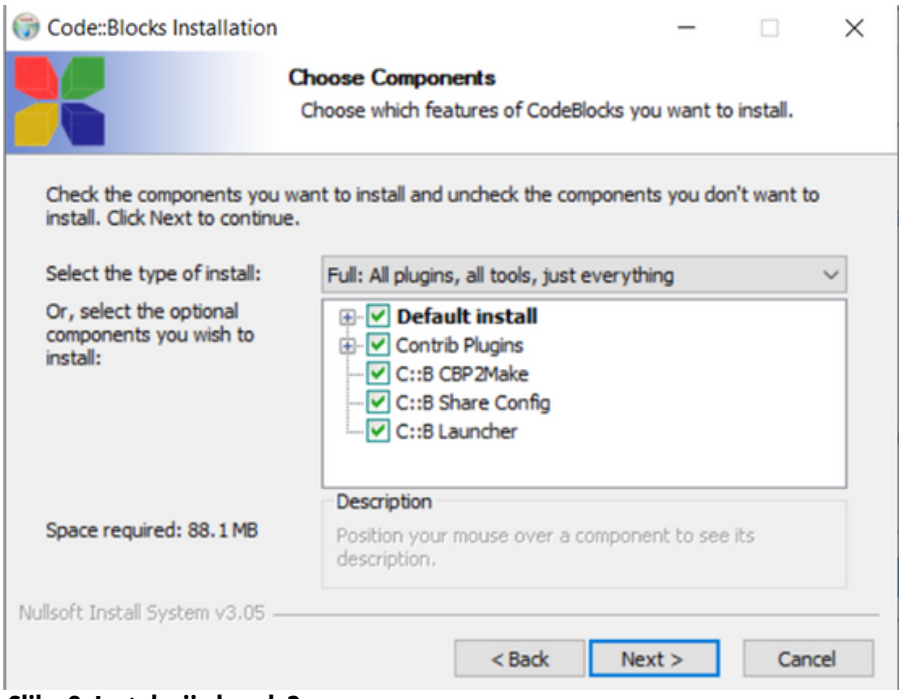
Slika 6: Instalacija korak 1

Zatim kliknite I Agree

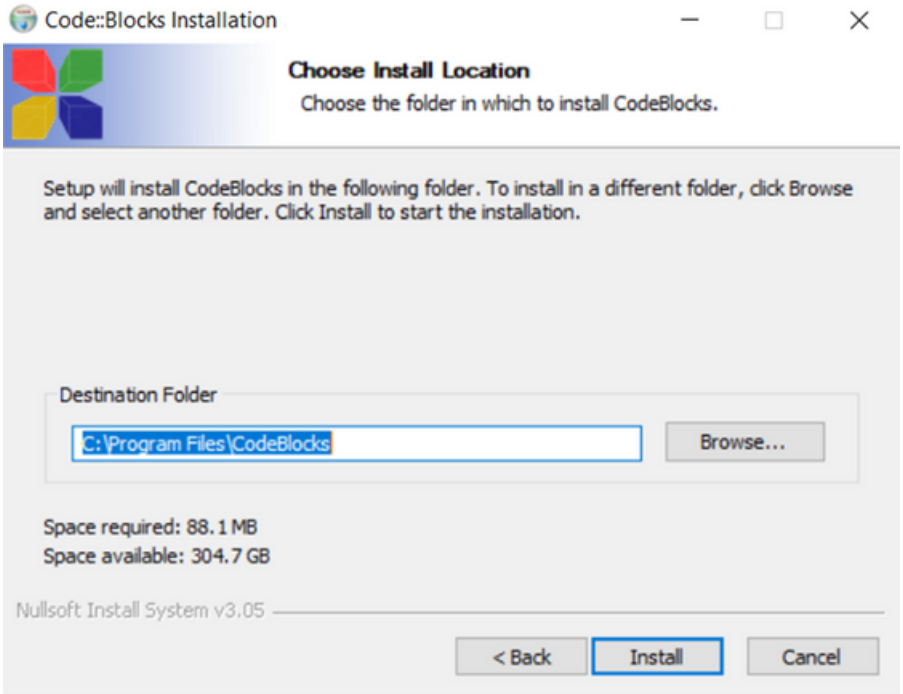


Slika 7: Instalacija korak 2

Na sljedeća dva ekrana ostavite sve opcije kako su i predložene – klikovi na Next i Install.

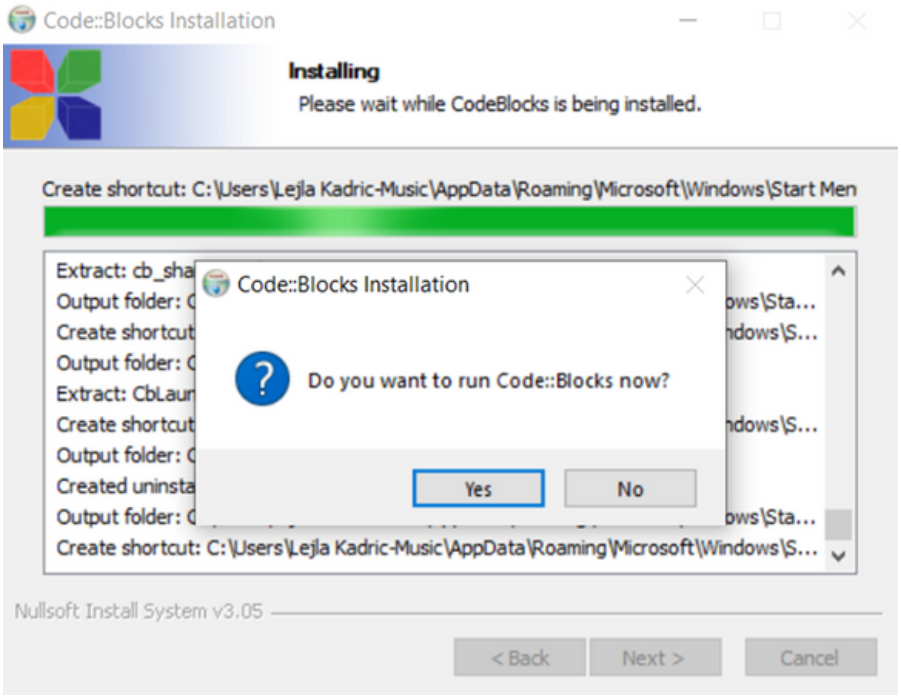


Slika 8: Instalacija korak 3



Slika 9: Instalacija korak 4

Na sljedeća dva ekrana ostavite sve opcije kako su i predložene – klikovi na Next i Install.



Slika 10: Instalacija korak 5

Trebalo bi da se pojavi navedena ikona na desktopu kao sa slike 11.

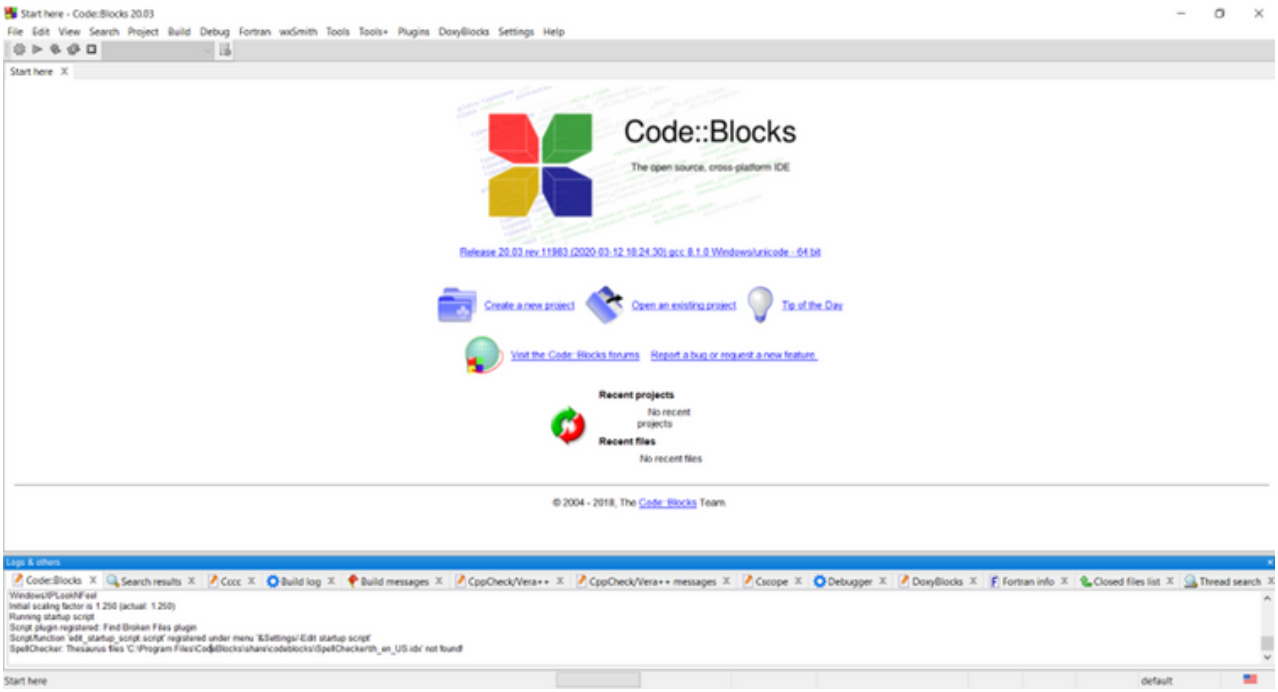
Slika 9: Instalacija korak 4

2.2.1 Prilagodba CodeBlocksa za programiranje

U izborniku View:

- kliknite na Manager i tako maknite kvačicu s njega,
- iz View –>Toolbars maknite kvačice sa svega, osim Compiler.

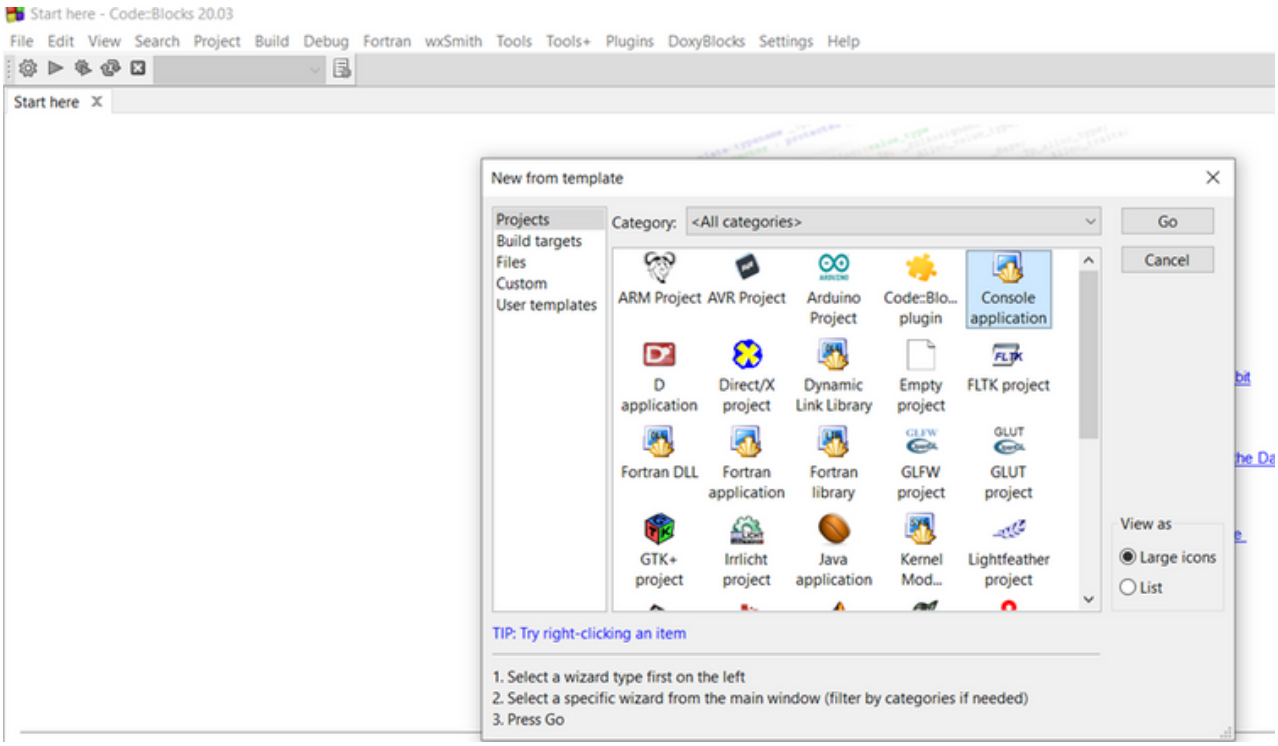
Sada bi vam View –>Toolbars trebao izgledati ovako:



Slika 12: Prilagodba okruženja

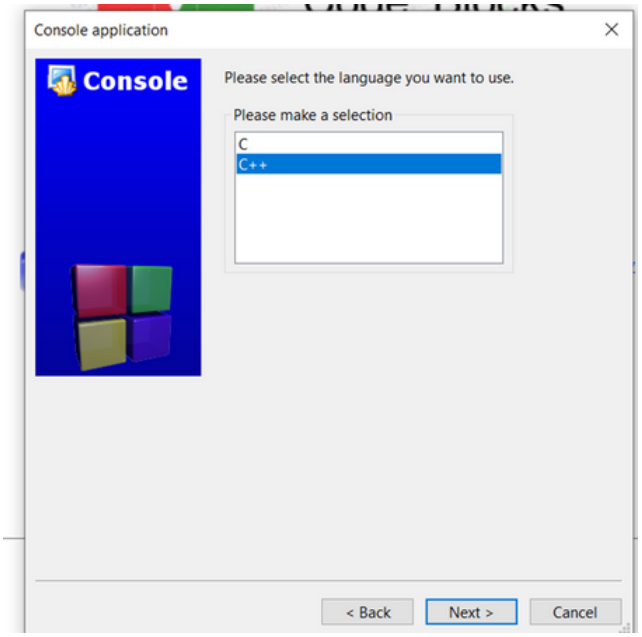
2.3 KREIRANJE PROJEKTA

Projekat kreiramo tako što iz menija otvorimo File –> New –> Project i kliknemo Console Application –> Go



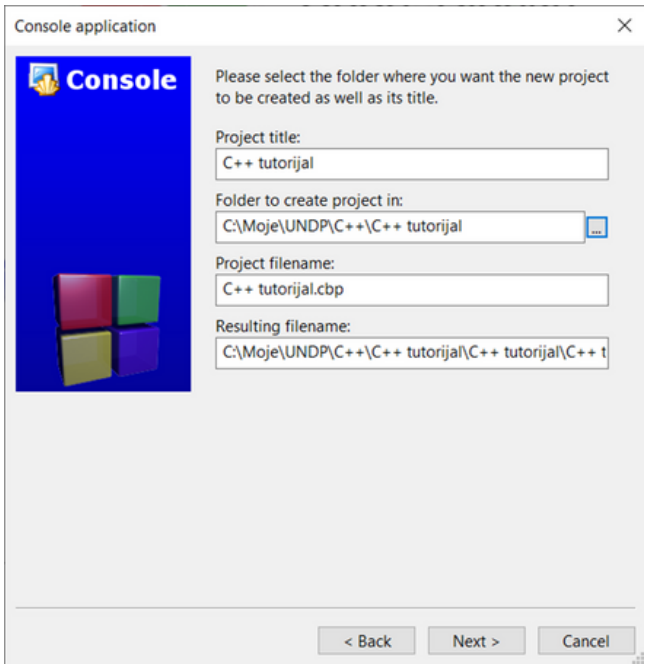
Slika 13: Kreiranje novog projekta

Dalje biramo Next i izaberemo C++ kao s naredne slike.



Slika 14: Odabir C++ projekta

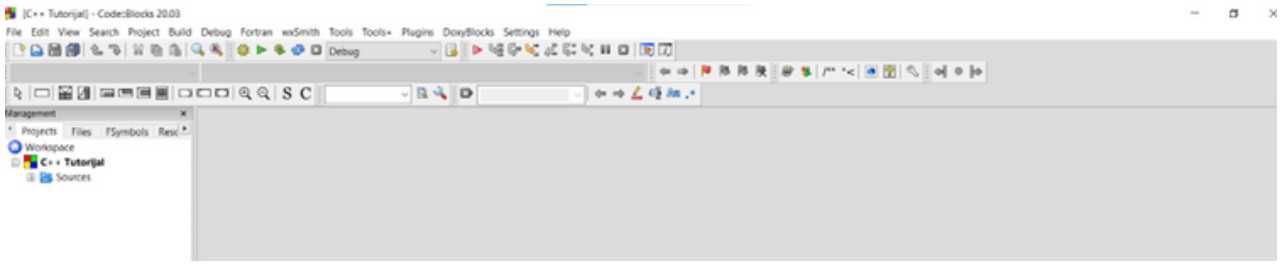
Dalje unesemo ime projekta i izaberemo folder gdje ćemo projekat kreirati. Prethodno kreirajte folder gdje ćete smještati projekat.



Slika 15: Naziv i putanja do projekta

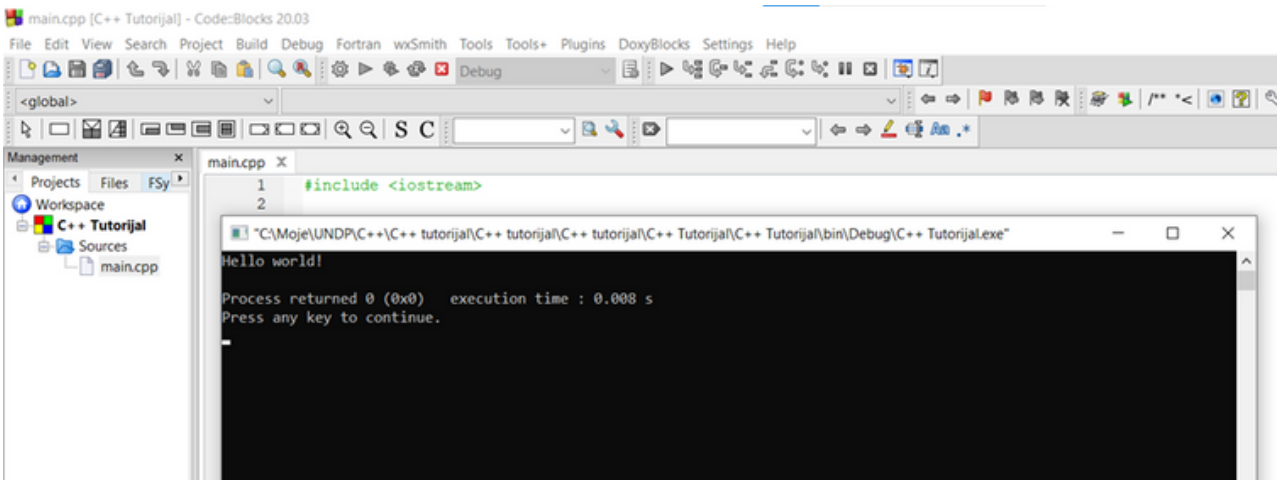
Dalje se klikne Next, ostavite sljedeći prozor kakav i jeste i kliknete Finish.

U lijevom uglu pojavi se novi projekat kao na slici 16.



Slika 16: C++ Tutorijal projekat

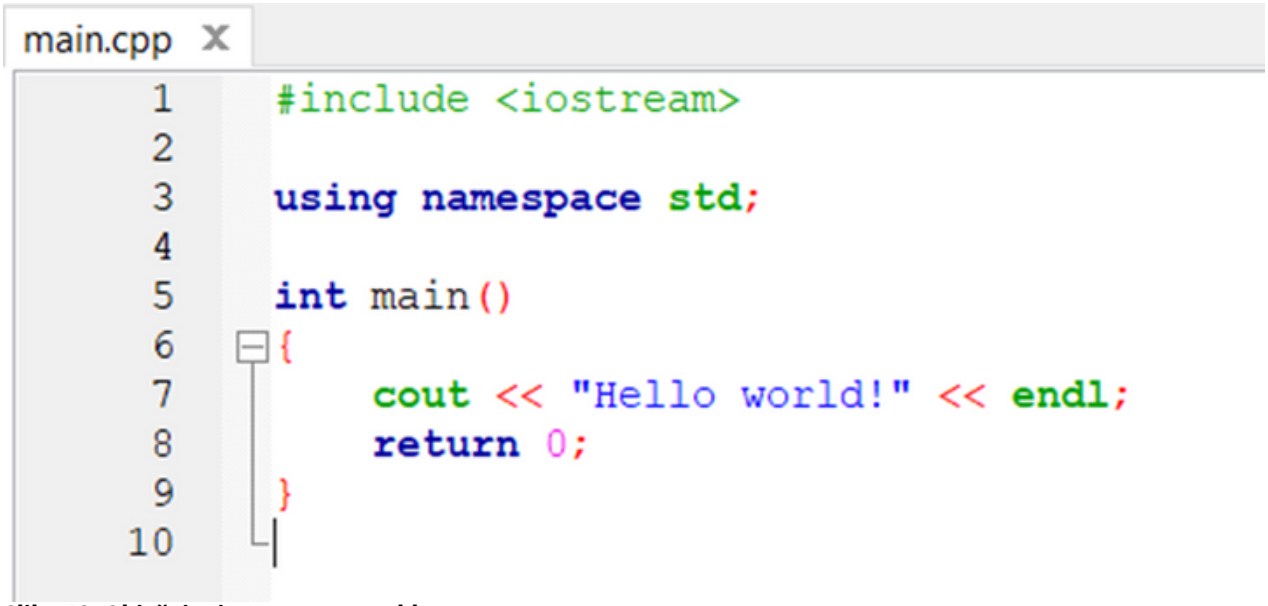
Kako bismo provjerili da sve radi ispravno, otvorite projekat s lijeve strane duplim klikom na main.cpp. Otiđite u Meniju na opciju Bild –> Run. Program ispiše poruku „Hello world!“ kao na slici 17.



Slika 17: Hello world!

2.4 OBJAŠNJENJE OSNOVOG PROBLEMA

U nastavku slijedi objašnjenje koda iz prethodnog primjera.



Slika 18: Objašnjenje osnovnog problema

1. Linija koda #include <iostream> je tzv. predprocesorska direktiva. To podrazumijeva dodavanje biblioteke (engl. library) koja se zove iostream. Dalji kod se dakle ne može izvršavati ako se prethodno ne učitaju sve funkcije (kao i sav ostali kod) iz navedene biblioteke koja je već napisana. Dakle, ovo je preduslov za izvršavanje bilo kojeg daljnjeg koda. Više o funkcijama pročitaćete u nastavku priručnika.
2. Linija koda using namespace std; Naredbe „#include“ i „using namespace std;” su jedne od mnogih predefinisanih naredbi unutar C++ jezika koje smanjuju vrijeme potrebno da programer obavi neke osnovne funkcije. Naredba „using namespace std;” nam omogućuje korištenje unaprijed definisanih riječi, funkcija, klasa, objekata a time ujedno skraćuje dužinu koda. Namespace je možemo reći „prostor” imena kojima imenujemo sve promjenljive, funkcije i klase u svojim programima. Svrha postojanja ovih odvojenih „prostora” je da ne dođe do kolizije ukoliko program postane suviše velik, a inspiracija za imenovanje novih promjenljivih zataji.
Rječica „using” govori kompajleru da se koristi „prostor”, koji se navede u namespace liniji.
Std predstavlja standard, što znači da će se za sve buduće naredbe koristiti standardni namespace. U programskom jeziku C++ prilikom završavanja naredbe koristi se simbol „;” (tačka-zarez). Tim načinom govorimo programu prevoditelju (kompajler) gdje je kraj naredbe. U jednoj komandnoj liniji moguće je imati i više naredbi poput cout<<111<<endl; cout<<333<<endl; Iako se ovaj kod sastoji od dvije naredbe, zahvaljujući razdjelniku tačka-zarez, moguće ih je upisati u istoj komandnoj liniji što se često koristi kako bi sam kod programa bio čitljiviji.
3. Linija kodaint main() predstavlja početak svakog programa.
4. Otvorena i zatvorena vitičasta zagrada (plave boje na narednoj slici) predstavljaju tijelo funkcije

```
6   {
7       cout << "Hello world!" << endl;
8       return 0;
9   }
10
```

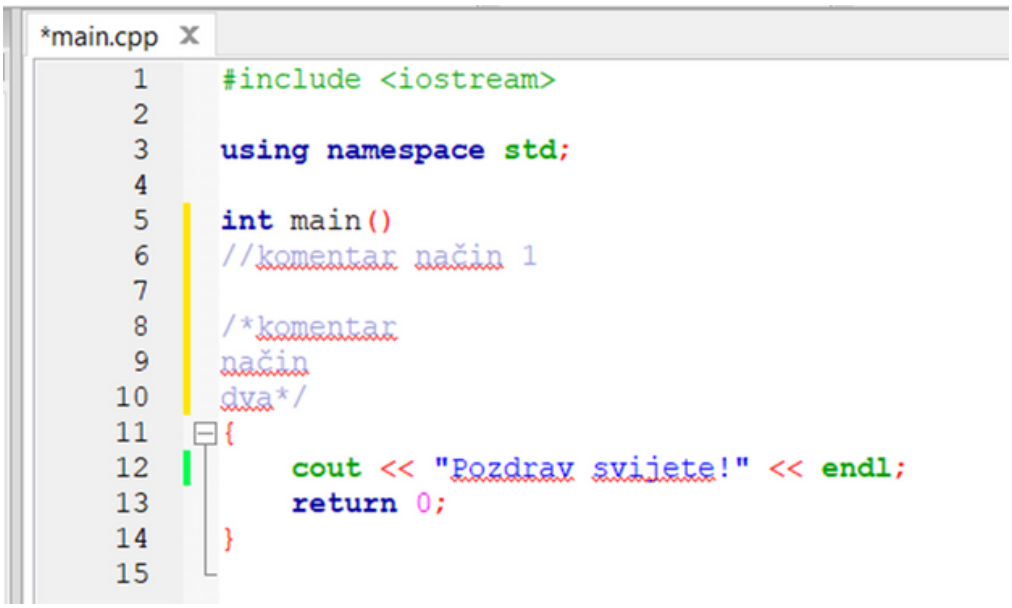
Slika 19: Tijelo funkcije

5. Linija koda cout << "Hello world!" << endl; predstavlja ispisivanje streama Hello world! i predstavlja pisanje u konzolu (Console OUT). Dakle, funkcija cout (čitaj c out) ispisuje navedeni stream koji joj je proslijeđen. Dakle, iz iostream biblioteke uzeta je funkcija cout i njom ispisan navedeni tekst koji se nalazi između dvostrukih navodnika. Zapamtite – sve što se nalazi ispod dvostrukih navodnika se tretira kao tekst. Komanda endl end line predstavlja prelazak u novi red. Znak << je tzv. overloaded operator tj. operator izlaza (output operator ili insertion operator), koji ukazuje kompajleru da sve što slijedi nakon simbola treba biti odštampano na ekranu/konzoli.
6. Linija kodareturn 0; vraća nulu. Dakle, ako program završi dobro, vratiće nulu, a ako se dogodi neka greška, neće vratiti nulu.

2.5 KOMENTARI

Jedna od najvažnijih stvari prilikom pisanja koda je komentarisanje. Iako je mnogima u početku teško se priviknuti na ovu praksu, s vremenom se programeri nauče obavezno komentarisati svoj kod kako bi se i oni sami, ali i drugi programeri koji će proučavat njihov kod, mogli snaći i lakše shvatiti „što je pjesnik htio reći”. Druga svrha komentara je kada programeru određeni dio koda ne treba, a ne želi ga izbrisati i kao brzo rješenje se nudi opcija da se kod zakomentariše. Postoje dva načina komentarisanja. Ukoliko želimo zakomentarisati samo jednu liniju, koristit ćemo „//”: dva operatora dijeljenja odnosno dva slash znaka. Sve što se nalazi u toj jednoj liniji koda iza dva slash znaka će se smatrati komentarom.

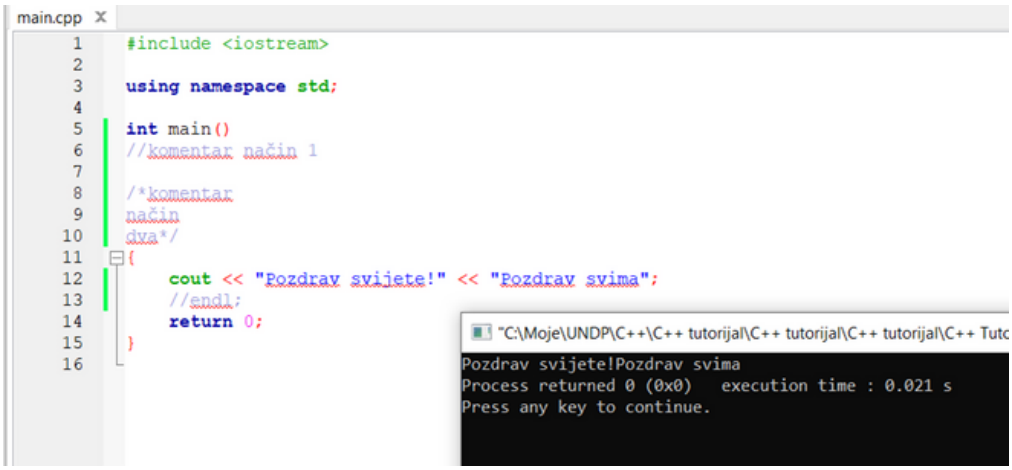
Ukoliko imamo veći dio koda za zakomentarisati (zamislite 50 linija koda), tada ne bi imalo smisla ići liniju po liniju i svuda stavljati po dva slash simbola i kasnije svaki od njih ručno i uklanjati. Naravno da postoji lakši način a to je korištenjem „/*" i „*/" simbola. Početak (odakle želimo početi komentarisanje) se obilježava slash simbolom i zvjezdicom a kraj (dokle da se kod zakomentariše) obilježavamo prvo simbolom zvjezdice a zatim slash simbolom. Sve što je zakomentarisano se neće izvoditi prilikom prevođenja programa. Ukoliko imamo zakomentarisano više linija, moguće je taj dio koda minimizirati pritiskom na simbol „-" (minus odnosno povlaka) koji se nalazi uz početnu liniju komentara. Na isti način, samo pritiskom na simbol „+" (plus) će se sav zakomentarisani tekst prikazati.



Slika 20: Komentari

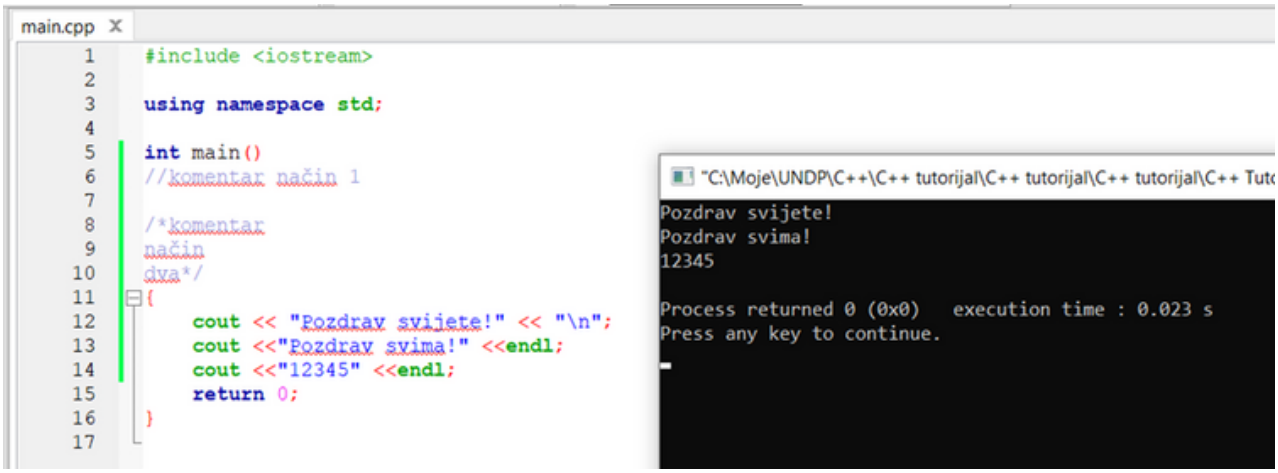
2.6 ISPISIVANJE TEKSTA

U odnosu na prethodni primjer gdje se ispisivao tekst „Pozdrav svijete" kao na slici 20, sada ispisujemo dvije rečenice kao na slici 21.

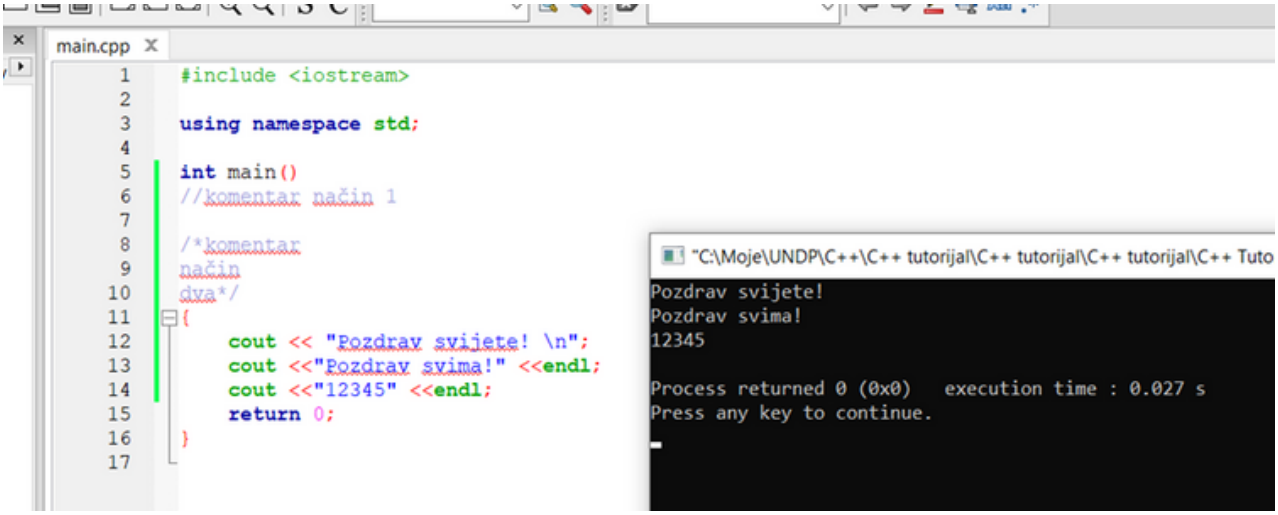


Slika 21: Ispis teksta u jednom redu

Primjećuje se da se u gornjem primjeru sav tekst na ekranu „lijepi" jedan za drugi. Kada želimo to da izbjegnemo i dobijemo pregledniji ispis tako da poslije nekog ispisa na ekran kursor prijeđe u novi red, treba koristiti konstantu endl (eng. end of line) iz biblioteke iostream, kao u primjeru koji slijedi i koji je samo izmijenjena verzija prethodnog primjera. Za prelazak u novi red može i da se pošalje „\n" na standardni izlaz s istim efektom kao da smo upotrijebili konstantu endl.



Slika 22: Ispis teksta u više redova koristeći i „\n"



Slika 23: Ispis teksta u više redova koristeći i „\n"

2.7 UVOD U VARIJABLE

2.7.1 Šta su varijable

Pojam varijable ste zasigurno već čuli.

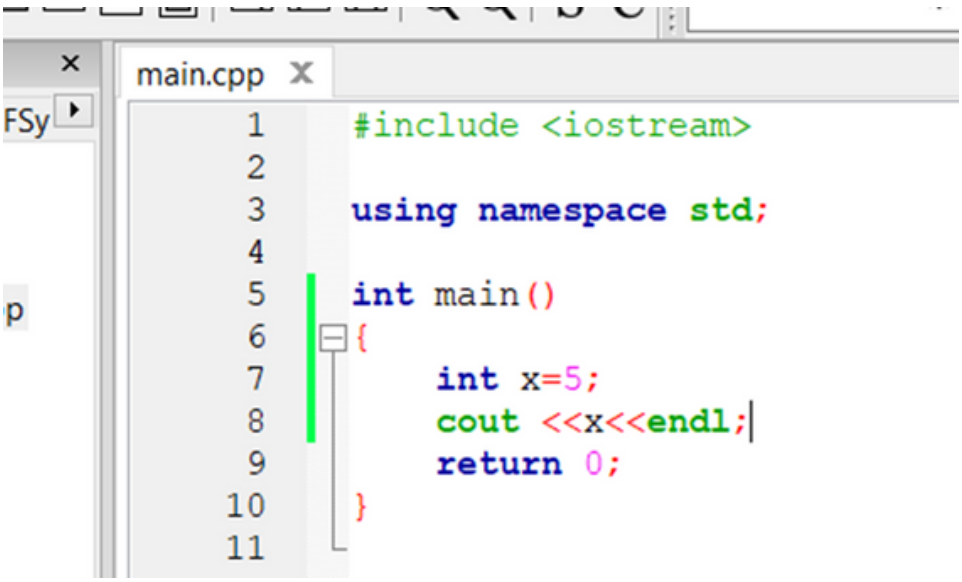
Osnovni zadaci poput X=5, Y=5, Z=X+Y, pronađi „Z", u sebi sadrže tri različite varijable. U ovom primjeru to su varijable X, Y i Z. Varijable neće uvijek poprimiti unaprijed definisanu vrijednost i tu će se vidjeti moć varijabli. Svaku varijablu je potrebno definisati a logično je da ćemo je htjeti i inicijalizirati.

2.7.2 Deklaracija varijabli

Deklaracija varijabli je zapravo govorenje našem programu da ćemo koristiti varijablu nekog imena. Primjerice, naredba int x; će našem programu reći da stvaramo varijablu koja je tipa integer (skraćenica je int a predstavlja cijele brojeve) i koja će se zvati „x". Značenje riječi integer i ostalih tipova podataka slijedi u nastavku priručnika.

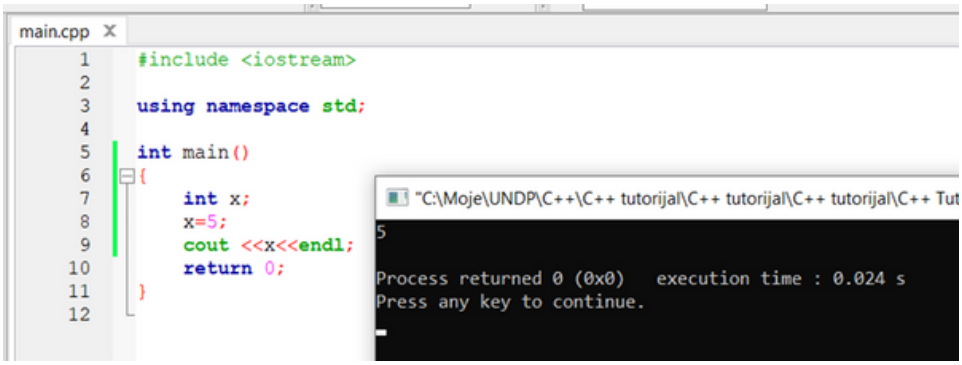
2.7.3 Inicijalizacije varijable

Svaku varijablu, nakon što je stvorimo (u gornjem primjeru je to bilo int x;), poželjno je i postaviti na neku vrijednost odnosno pridružiti joj neku vrijednost. Postupak dodjeljivanja neke vrijednosti nekoj varijabli zovemo inicijalizacija. Primjer inicijalizacije bi bio da smo varijabli „x" dodijelili neku cjelobrojnu vrijednost, poput recimo 5 (pet). Naš kod bi tada izgledao ovako: int x = 5;



Slika 24: Varijabla X tipa integer vrijednosti 5

Može se i kroz dva zasebna koraka obaviti postupak, prvo stvoriti varijablu a zatim joj dodijeliti vrijednost. Tada bi kod ovako izgledao:

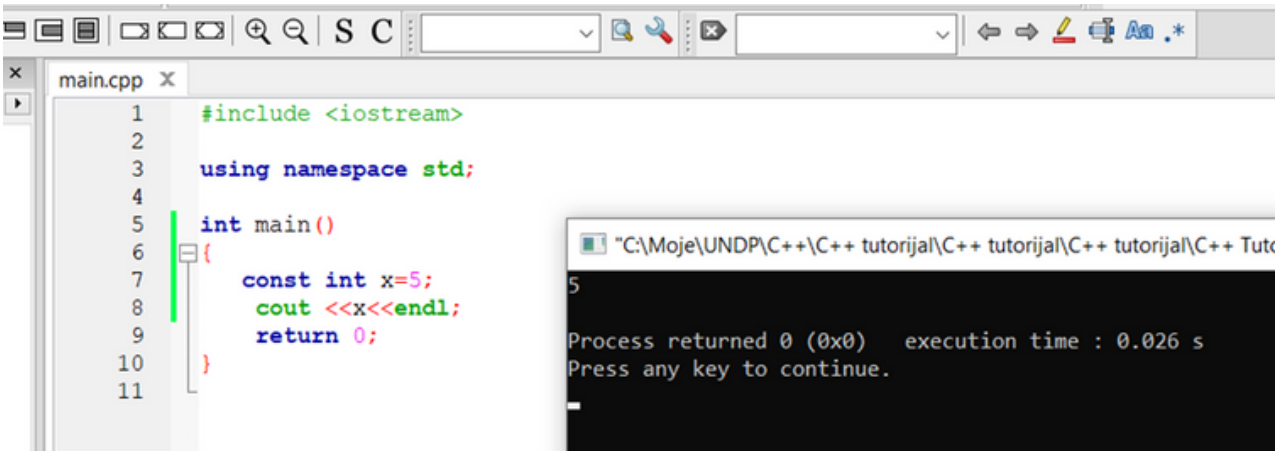


Slika 25: Varijabla X tipa integer vrijednosti 5 u dva koraka

Varijabla je dakle simbolično ime za neku vrijednost kako bismo se lakše snašli u našem programu.

2.7.4 Konstantne varijable

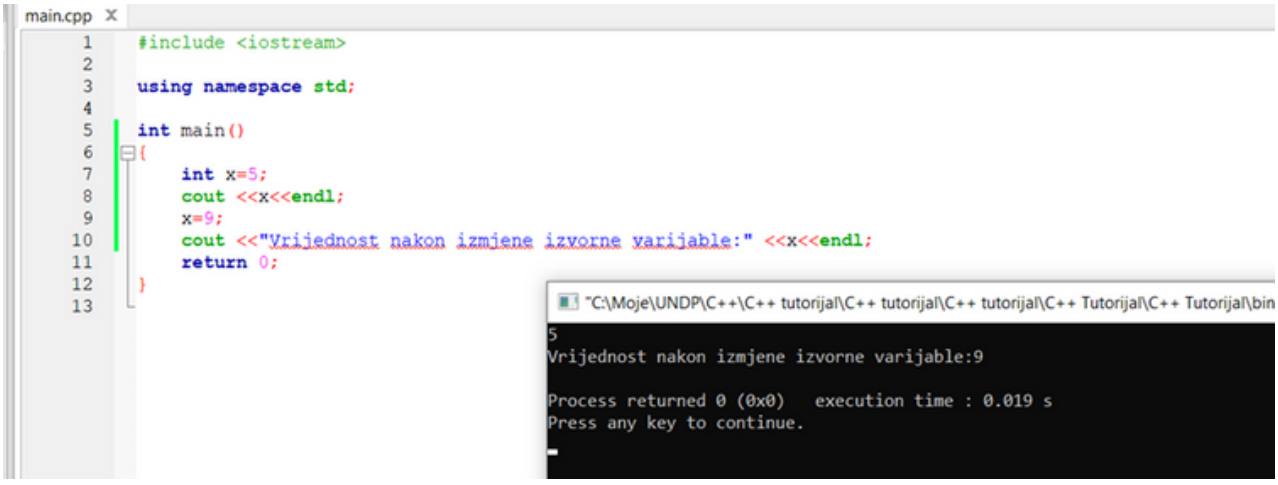
Poseban tip varijabli su konstante. Ovu vrstu varijabli možemo samo jednom postaviti na neku vrijednost (inicijalizirati). Koriste se kada želimo zaštititi vrijednost neke varijable u našem programu da je drugi programeri ne mogu naknadno mijenjati u svom kodu. Sintaksa je jednostavna, samo se doda ključna riječ„const" ispred postavljanja tipa podatka te varijable. Primjer možete vidjeti na slici 26.



Slika 26: Konstantna varijabla X tipa integer vrijednosti 5

2.8 MEMORIJSKI KONCEPT VARIJABLI

Prilikom deklaracije varijabla ima neku svoju početnu vrijednost, ali se u toku programa može mijenjati.
Na slici 27 deklarisali smo varijablu X tipa integer kojoj je vrijednost 5. Nakon toga smo linijom koda x=9; promijenili vrijednost iste te varijable. Tip podatka nije potrebno navoditi svaki put, dovoljno je samo prilikom inicijalizacije. Pokretanjem programa vidimo da je varijabla X promijenila vrijednost.



Slika 27: Izmjena vrijednosti varijable

2.9 PONAVLJANJE

Nakon završetka poglavlja, čitatelju bi trebali biti poznati sljedeći pojmovi:

- IDE - Integrated Development Environment
- Kreiranje projekta
- Objašnjenje koda osnovnog problema
- Komentari
- Varijable

3 OPERATORI, VARIJABLE I NAREDBE

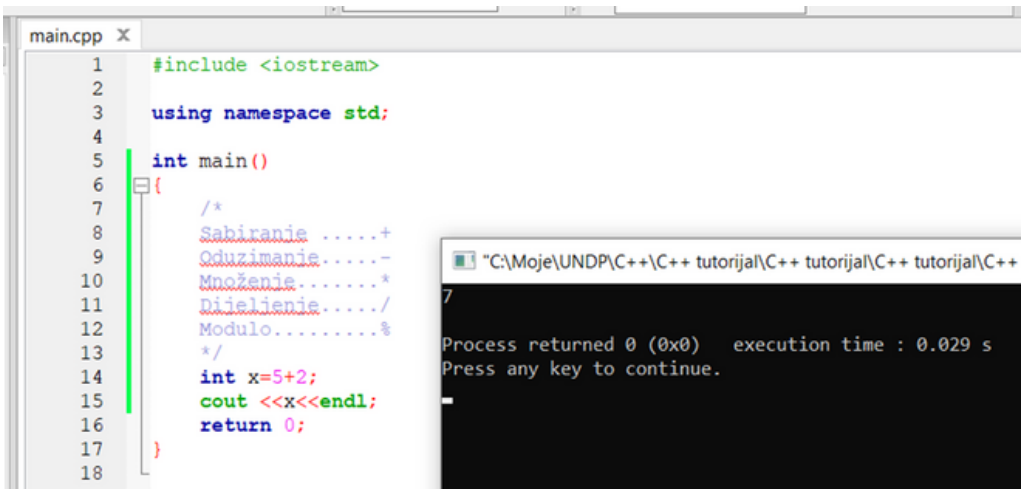
3.1 ARITMETIČKI OPERATORI

U narednoj tabeli nalaze se svi aritmetički operatori u C++ programiranju.

Funkcija	Operator
Sabiranje	+
Oduzimanje	-
Množenje	*
Dijeljenje	/
Modulus (rezultat je cjelobrojni ostatak dijeljenja dva cijela broja)	%

Tabela 4: Aritmetički operatori

Aritmetički operatori uključuju sabiranje (+), oduzimanje (-), množenje (*) i dijeljenje (/). Mogu se koristiti na vrijednostima bilo koje numeričke vrste: byte, short, int, long, float ili double. Kad računar izvršava jednu od ovih operacija, vrijednosti koje uzima u račun moraju biti iste vrste. Rezultat množenja dviju cjelobrojnih varijabli je cjelobrojna varijabla, rezultat množenja double varijabli je opet double. Oprez je potreban kod operatora dijeljenja. Rezultat dijeljenja dviju cjelobrojnih varijabli je cijeli broj, a eventualni ostatak kod dijeljenja se odbacuje. Operator koji računa ostatak cjelobrojnog dijeljenja označava se znakom % i naziva modulo.



Slika 28: Primjer sabiranja

main.cpp

```
1  #include <iostream>
2
3  using namespace std;
4
5  int main()
6  {
7      /*
8      Sabiranje .....+
9      Oduzimanje.....-
10     Množenje.......*
11     Dijeljenje.....//
12     Modulo.........%
13     */
14     int x=5-2;
15     cout <<x<<endl;
16     return 0;
17 }
```

```
"C:\Moje\UNDP\C++\C++ tutorijal\C++ tutorijal\C++ tutorijal\C++ Tutorijal
3

Process returned 0 (0x0)   execution time : 0.030 s
Press any key to continue.
```

Slika 29: Primjer oduzimanja

main.cpp

```
1  #include <iostream>
2
3  using namespace std;
4
5  int main()
6  {
7      /*
8      Sabiranje .....+
9      Oduzimanje.....-
10     Množenje.......*
11     Dijeljenje.....//
12     Modulo.........%
13     */
14     int x=5-2;
15     cout <<x<<endl;
16     return 0;
17 }
```

```
"C:\Moje\UNDP\C++\C++ tutorijal\C++ tutorijal\C++ tutorijal\C++ Tutorijal
3

Process returned 0 (0x0)   execution time : 0.030 s
Press any key to continue.
```

Slika 30: Primjer množenja

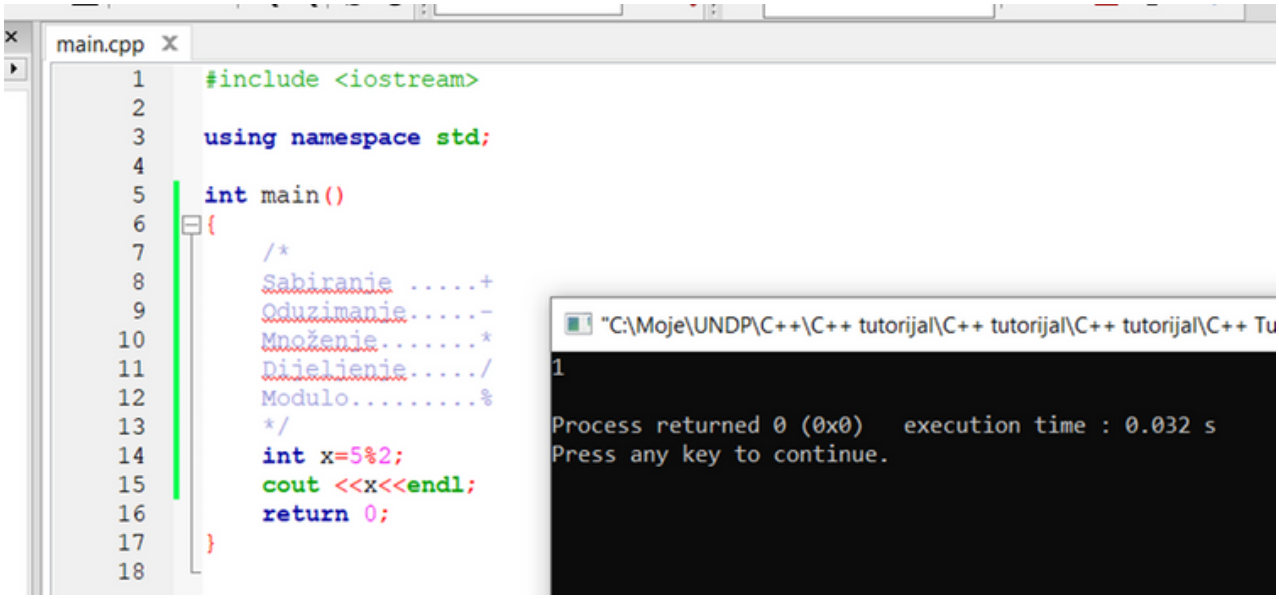
main.cpp

```
1  #include <iostream>
2
3  using namespace std;
4
5  int main()
6  {
7      /*
8      Sabiranje .....+
9      Oduzimanje.....-
10     Množenje.......*
11     Dijeljenje.....//
12     Modulo.........%
13     */
14     int x=5/2;
15     cout <<x<<endl;
16     return 0;
17 }
18
```

```
"C:\Moje\UNDP\C++\C++ tutorijal\C++ tutorijal\C++ tutorijal\C++ T
2

Process returned 0 (0x0)   execution time : 0.028 s
Press any key to continue.
```

Slika 31: Primjer dijeljenja



Slika 32: Primjer modulusa (ostatak od dijeljenja)

3.2 IDENTIFIKATORI

Svaka varijabla ili funkcija mora imati jedinstveno ime kako bi se mogli referencirati na njih. Imenovanja su proizvoljna no ipak postoje neka ograničenja:

- Svaki identifikator može sadržavati velika i mala slova engleske abecede, znamenke i simbol podcrte (_) no uz uslov da prvi znak imena odnosno identifikatora mora biti ili slovo ili podcrta.
- Identifikator ne smije biti jednak nekoj od ključnih riječi jezika, ali je može sadržavati. To znači da naš identifikator ne može biti riječ „int" jer je ta riječ predodređena u C++ jeziku za cjelobrojni tip podataka, ali smije se zvati npr. „pinta". Dobra je praksa (nije striktno zabranjeno, ali se ne preporučuje) izbjegavati stavljanje dviju ili više podcrta jednu do druge npr. „moj__program" jer lako može doći do krivog imenovanja pri pozivanju takvih imena u programu. Također se ne preporučuje započinjati ime s dvije podcrte i velikim slovom jer su neke riječi koje su striktno rezervirane za C++ pisane po takvom predlošku (npr. „__FILE__").

Sugestija je da ne štedite na broju znakova u identifikatorima jer je generalno pravilo da treba koristiti deskriptivna imena varijabli i funkcija. Primjerice, ako nazovemo varijablu „brojPi" i dodijelimo joj vrijednost „3.14" i nakon nekoliko mjeseci ćemo znati na što se odnosi ta varijabla, no nazovemo li je samo „x" postoji izuzetno velika šansa kako se svrhe te varijable nećemo odmah moći prisjetiti.

3.3 TIPOVI PODATAKA

Slično poput ljudi, i računari razlikuju tipove podataka. Ljudi razlikuju znamenke od slova ili simbola, brojeve od riječi i unutar tih osnovnih podjela možemo napraviti još manje podjele. Slova možemo podijeliti na velika i mala a brojeve na cjelobrojne (1, 2, 3, ...) i na realne odnosno s pomičnim zarezom (3.141592653589793238, 6.543, 1.61803, ...).

U programerskom svijetu osnovne (primitivne) tipove podataka dijelimo na:

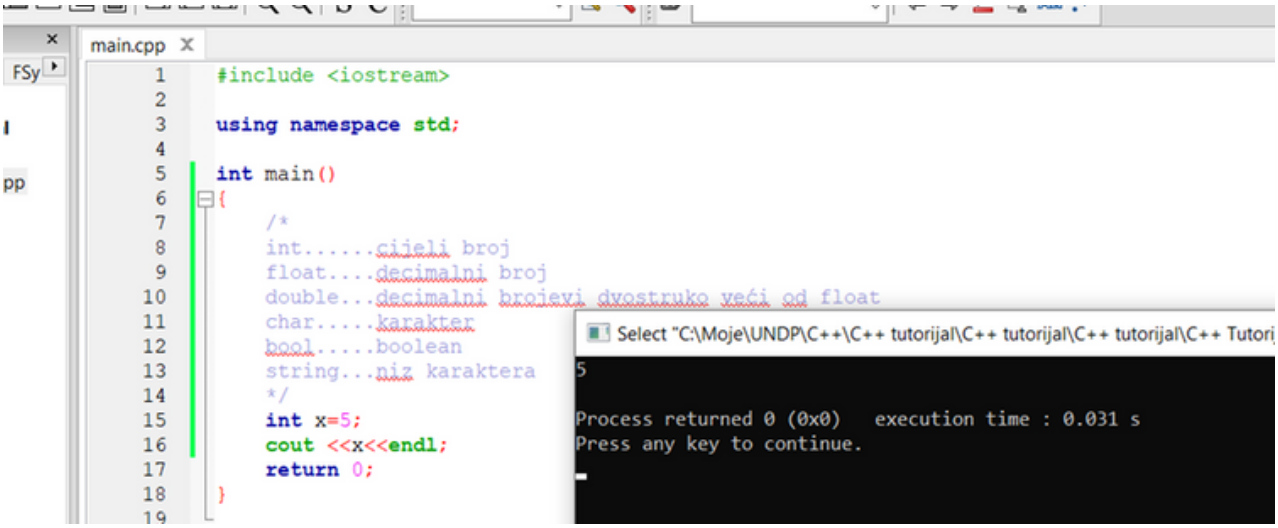
- integer,
- float,
- double,
- boolean,
- character.

Ime	Opis	Veličina	Raspon
char	Znak	1byte	S predznakom: -128 do 127 Bez predznaka: 0 do 255
short int(short)	Kratki cijeli broj	2byte	S predznakom: -32768 do 32767 Bez predznaka: 0 do 65535
int	Cijeli broj	4byte	S predznakom: -2147483648 do 2147483647 Bez predznaka: 0 do 4294967295
long int (long)	Dugi cijeli broj	4byte	S predznakom: -2147483648 do 2147483647 Bez predznaka: 0 do 4294967295
bool	Boolean	1byte	true ili false
float	Decimalni brojevi	4byte	+/- 3.4e +/- 38 (~7 cifara)
double	Decimalni brojevi dvostruke preciznosti	4byte	+/- 3.4e +/- 38 (~7 cifara)
long double	Dugi decimalni broj dvostruke preciznosti	8byte	+/- 3.4e +/- 38 (~7 cifara)
wchar_t	Široki znak	2byte ili 4byte	1 široki znak

Tabela 5: Tipovi podataka

3.3.1 Integer

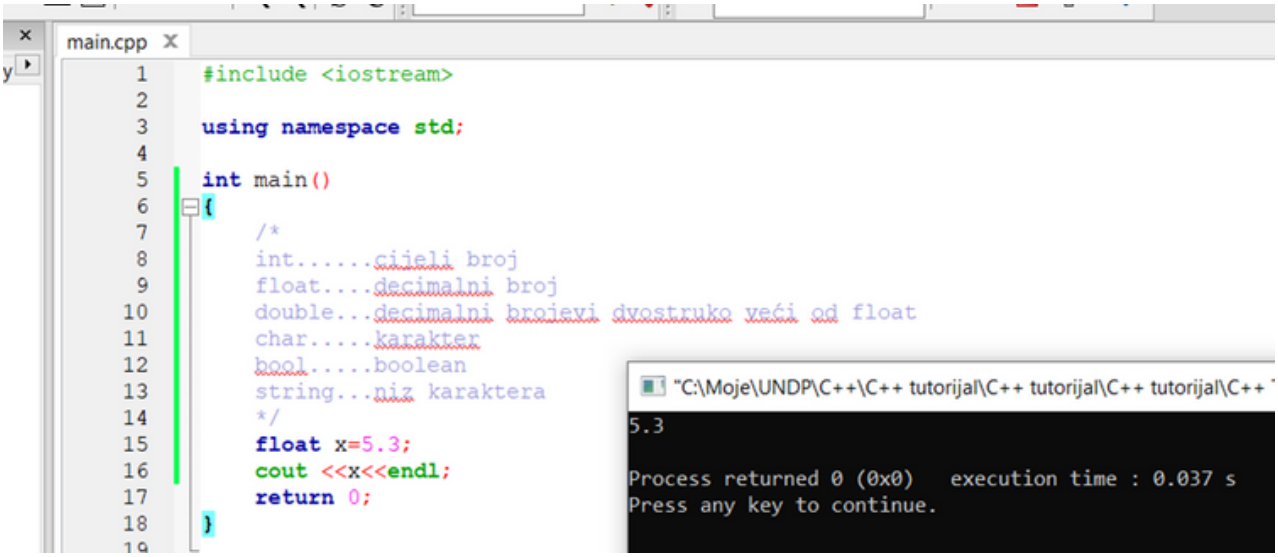
Integer (skraćenica je int) označava cijele brojeve. Tipu integer možemo pridodati i „podtipove" poput „short", „long" i „unsigned". Tip podatka short int će nam omogućiti manji raspon brojeva, a long int veći raspon brojeva, ali u skladu s tim i manje odnosno veće zauzeće memorije. Ukoliko se ukaže potreba za korištenjem striktno pozitivnih brojeva, tada možemo koristiti unsigned int.



Slika 33: int

3.3.2 Float

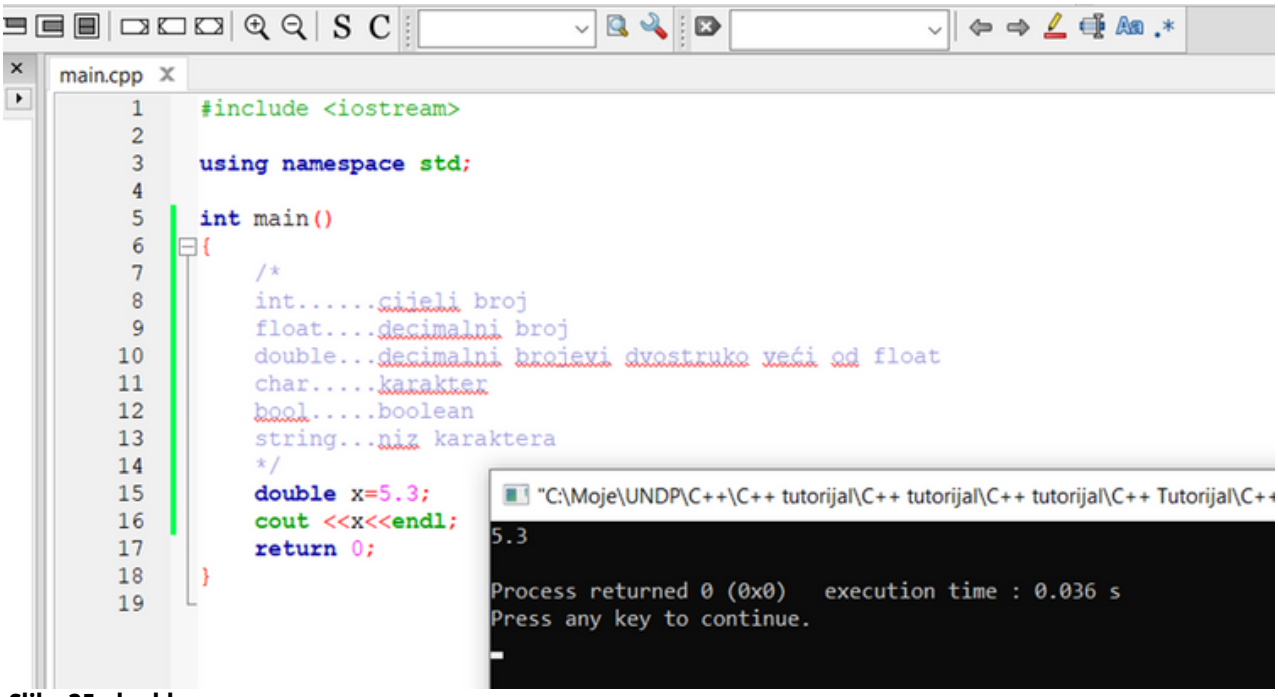
Ovaj tip podatka nam omogućuje veći raspon brojeva odnosno radi se o brojevima s pomičnim zarezom (tzv. decimalni brojevi). Takav tip podataka nam naravno daje puno veću preciznost. Decimalni zarez se ne zapisuje simbolom zareza (,) već simbolom tačka (.).



Slika 34: float

3.3.3 Double

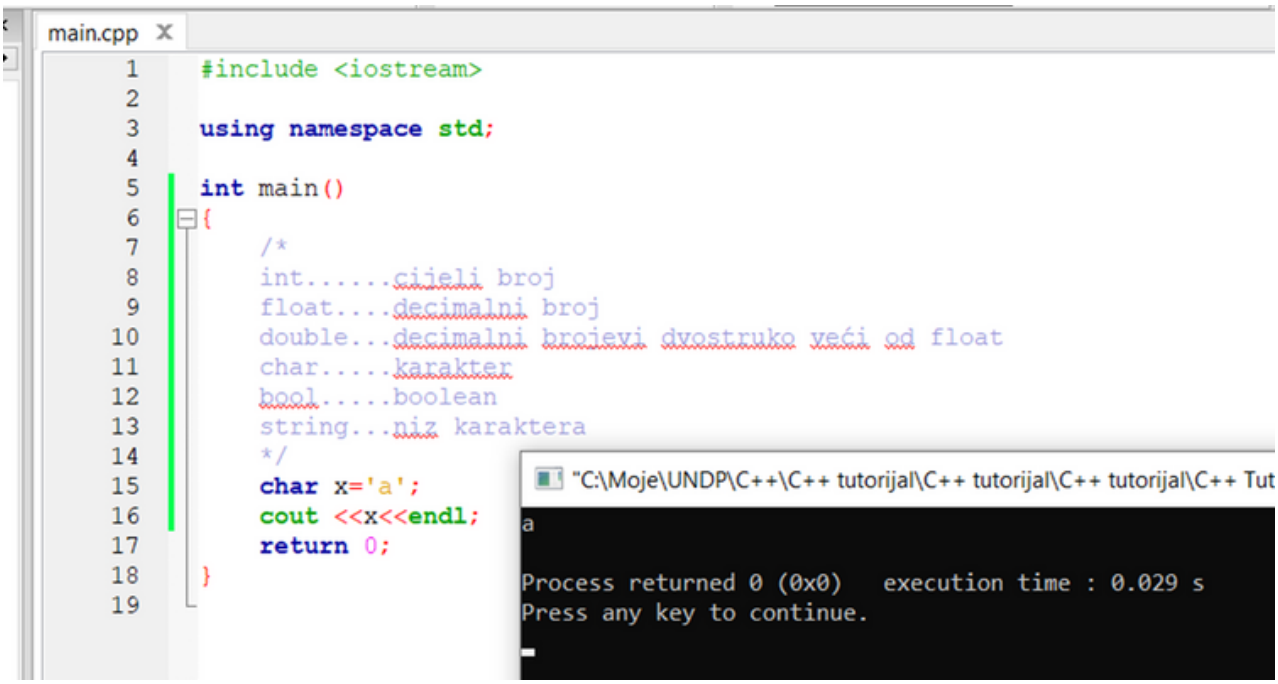
Double tip podatka daje okvirno duplo veću preciznost od float tipa, ali zauzima i duplo više memorije.



Slika 35: double

3.3.4 Character

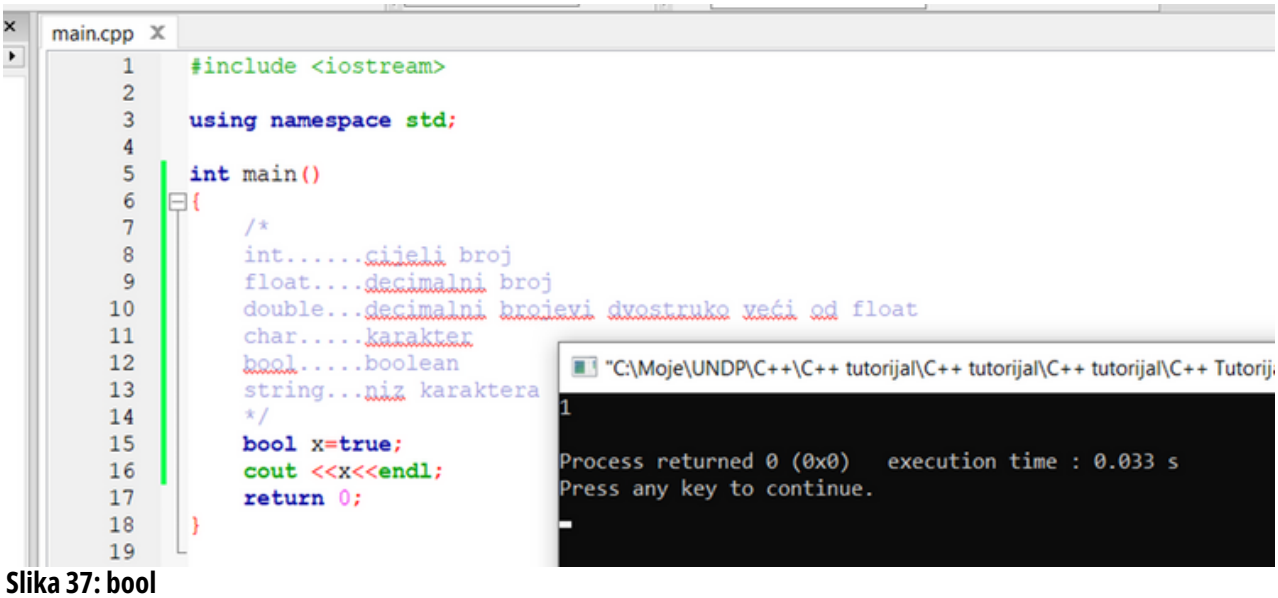
Za razliku od prethodna tri tipa, ovaj tip podataka (skraćenica je char) ne predstavlja brojeve niti znamenke već znakove (karaktere). Može sadržavati samo jedan znak (znamenku, slovo ili simbol). Prilikom inicijalizacije ovog tipa podataka koriste se jednostruki navodnici.



Slika 36: char

3.3.5 Boolean

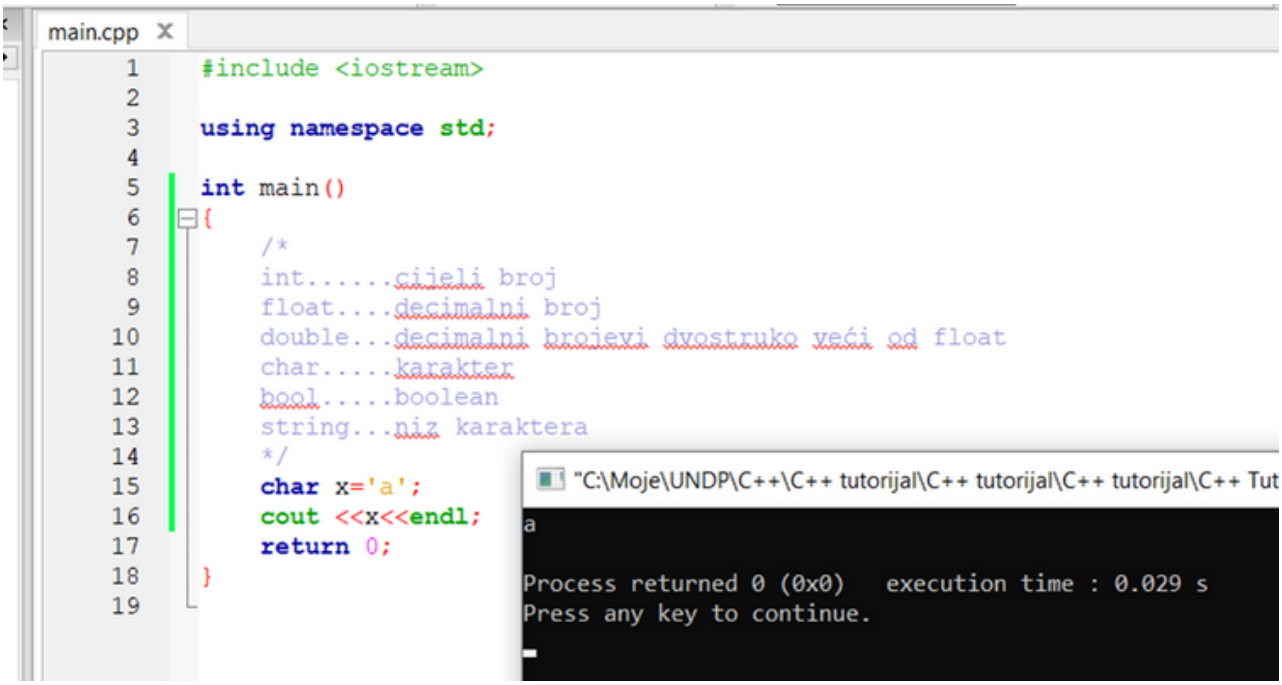
Ovaj tip podataka (skraćenica je bool) ima predodređene samo dvije vrijednosti. To su vrijednosti true (istina) i false (laž). Sinonim za te vrijednosti su ujedno i znamenke 1 i 0 gdje znamenka 0 (nula) predstavlja laž (false) a istina se predstavlja znamenkom 1 (jedan). Važno je napomenuti da varijabli koja je tipa bool možemo dodijeliti i vrijednost 2, 3, 5, itd. ali će ta vrijednost biti zapisana kao 1 (jedan) odnosno kao istina. Takav pristup nemojte koristiti već koristite true i false vrijednosti.



Slika 37: bool

3.3.6 String

Od svih predstavljenih tipova podataka, ovaj tip jedini ne spada u primitivne tipove. Važno je znati da je string tip podatka zapravo skup znakova (simbola, slova i brojki). Nečije ime bi bilo tipa string, ali isto tako možemo i brojeve zapisati kao stringove no tada se nad njima neće moći vršiti matematičke operacije. Važno je zapamtiti da se stringovi inicijaliziraju tako da se varijabli pridružuje neka vrijednost koja se nalazi pod dvostrukim navodnicima i da sve što je string tretiramo kao tekst nezavisno od toga što se nalazi između dvostrukih navodnika.



Slika 38: string

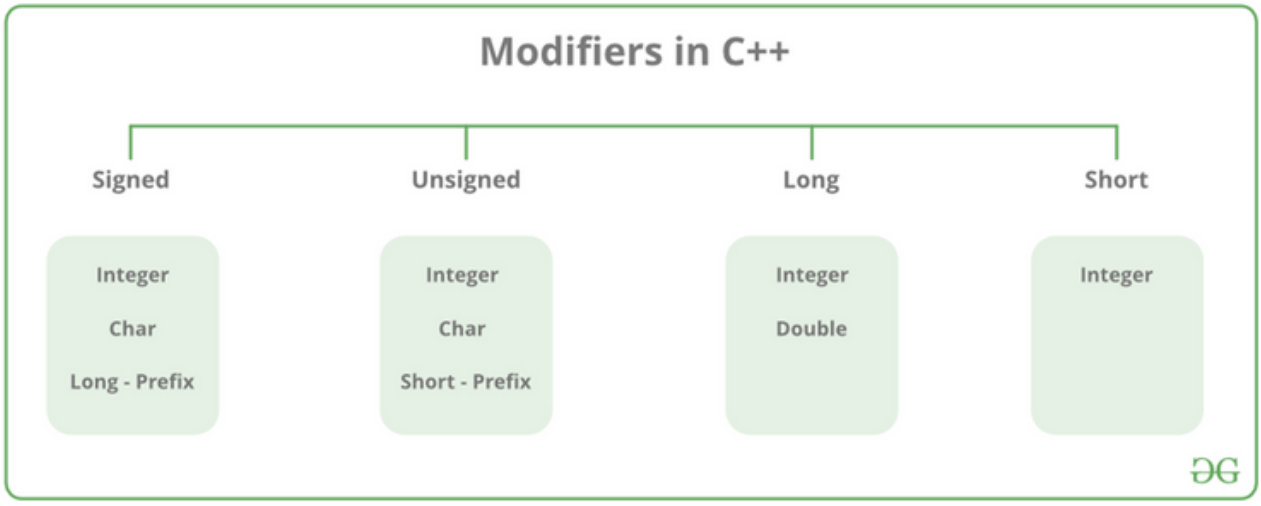
3.4 KLJUČNE RIJEČI - ZABRANJENE RIJEČI

Lista ključnih riječi u C++ programskom jeziku:

and	decltype	new	
and_eq	default	noexcept	switch
alignas	delete	not	template
alignof	double	not_eq	this
asm	dynamic_cast	nullptr	thread_local
auto	else	operator	throw
bitand	enum	or	true
bitor	explicit	or_eq	try
bool	export	private	typedef
break	extern	protected	typeid
case	false	public	typename
catch	float	register	union
char	for	reinterpret_cast	unsigned
char16_t	friend	return	using
char32_t	goto	short	virtual
class	if	signed	void
compl	inline	sizeof	volatile
const	int	static	wchar_t
constexpr	long	static_assert	while
const_cast	mutable	static_cast	xor
continue	namespace	struct	xor_eq

Slika 39: Ključne riječi

U nastavku je dato detaljno objašnjenje nekoliko ključnih riječi u C++.



Slika 40: Ključne riječi short/long/signed/unsigned

3.4.1 Short

Short je skraćivanje veličine. Npr. short int će zauzeti manje memorije od int. Kratka ključna riječ mijenja minimalne vrijednosti koje tip podataka može zadržati. Koristi se za male cijele brojeve koji se nalaze u rasponu od −32,767 do +32,767.

Primjer:

short int x=2;

3.4.2 Long

Duga ključna riječ mijenja maksimalne vrijednosti koje tip podataka može zadržati. Koristi se za velike cijele brojeve koji su u rasponu od -2147483647 do 2147483647.

Primjer:

long int y = 26936;

3.4.3 Signed

Signed varijable mogu pohraniti pozitivne, negativne cijele brojeve i nulu.

Primjer:

signed int a = 45;

signed int b = -67;

signed int c = 0;

gdje je

'a' pozitivan cijeli broj;

'b' negativan cijeli broj;

'c' cijeli broj nulte vrijednosti.

3.4.4 Unsigned

Unasigned varijable mogu pohraniti samo negativne cjelobrojne vrijednosti.

Primjer:

unsigned int a = 9;

unsigned int b = 0;

gdje je

'a' pozitivan cijeli broj;

'b' cijeli broj nulte vrijednosti.

3.5 PETLJE I KONTROLE TOKA

U programiranju često koristimo kontrole toka i petlje koje nam omogućuju kontroliranje izvedbe našeg programa ili ponavljanje neke operacije određeni broj puta.
Petlje i kontrole toka su izuzetno važan dio programiranja, no kao i kod svega u programiranju, neke petlje su češće u upotrebi a neke rjeđe.

Kontrole toka se koriste želimo li neku operaciju izvršiti isključivo ako je neki uslov zadovoljen. Npr. površinu pravouglog trougla ćemo računati samo ako je korisnik/ca ispravno unio/jela sve tražene parametre. Ako nije, zahtijevaćemo od njega/nje ponovni upis.

Petlje nam omogućuju ponavljanje neke linije ili skupa linija određeni broj puta. Najčešće se unaprijed ne zna tačan broj koliko se puta nešto treba izvršiti u petlji. Možda zvuči malo čudno, ali je istinito. Npr. izradite li aplikaciju koja će na ekran ispisati slovo „A" onoliko puta koliko puta korisnik/ca kaže, nećete moći unaprijed znati tačno koliko će se slova ispisati. Unese li korisnik/ca da želi slovo „A" ispisati 3 puta, vi u svom programu možete imati sljedeće opcije:

- opcija 1 (korisnik/ca je unio/jela da se slovo A ispiše nula puta odnosno niti jednom
  - vaš pseudokod bi glasio: nemoj ništa ispisati
- opcija 2 (korisnik/ca je unio/jela da se slovo A ispiše jednom)
  - vaš pseudokod bi glasio: ispiši slovo „A"
- opcija 3 (korisnik/ca je unio/jela da se slovo A ispiše dva puta)
  - vaš pseudokod bi glasio: ispiši slovo „A" i opet ispiši slovo „A"
- opcija 4 (korisnik/ca je unio/jela da se slovo A ispiše tri puta)
  - vaš pseudokod bi glasio: ispiši slovo „A" i opet ispiši slovo „A" i opet ispiši slovo „A"

Takav pristup naravno nema smisla posebno razmotrimo li mogućnost da korisnik/ca može poželjeti slovo „A" ispisati 10.000 puta. To znači da bi morali imati 10.001 opciju i da bi u posljednjoj opciji morali deset hiljada puta ponoviti naredbu „ispiši slovo A". Ispravan pristup je da napravimo petlju koja će ispisivati slovo „A" određeni broj puta. To znači da pitamo korisnika/cu koliko puta želi ispisati slovo „A" i petlji damo uslov „ispisuj na ekran slovo 'A' dok god ne ispišeš onoliko puta koliko je korisnik/ca zatražio/la".

3.6 NAREDBA IF

Jedan od najčešćih načina kontroliranja toka programa je korištenje „If naredbe" (engl. If statement). Sve kontrole toka (kao i petlje i većina toga u programiranju) se pišu malim slovima. „if" nam omogućuje izvršavanje nekog dijela našeg koda samo ako je neki uslov zadovoljen.

Sintaksa:

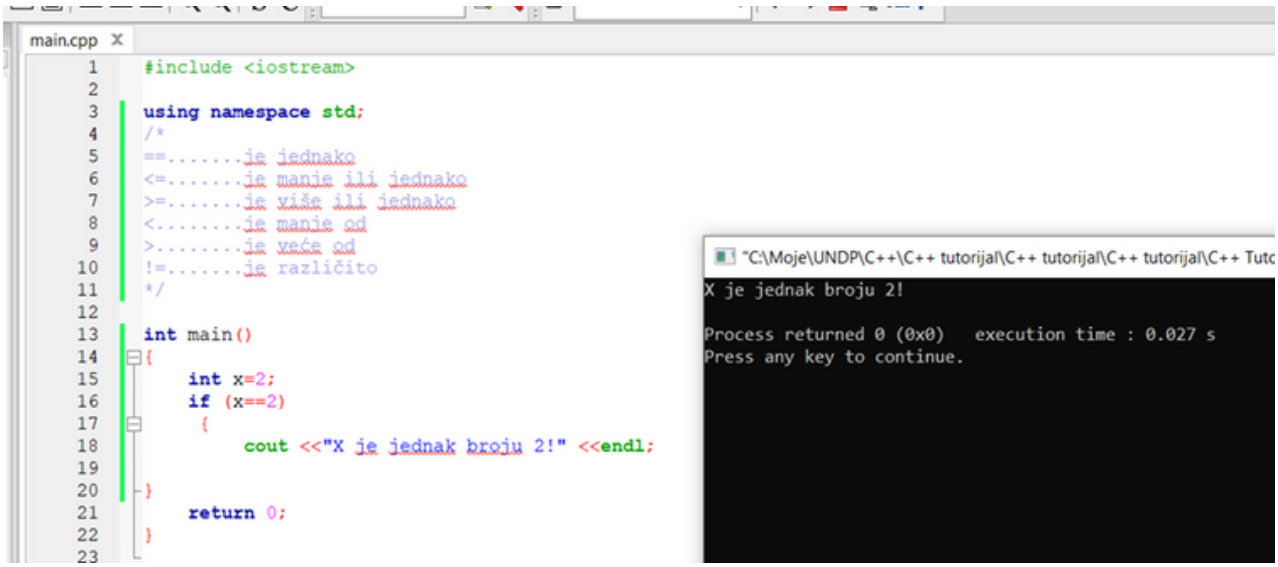
if (uslov koji treba biti zadovoljen)

{ kod će se izvršiti ako je uslov zadovoljen }

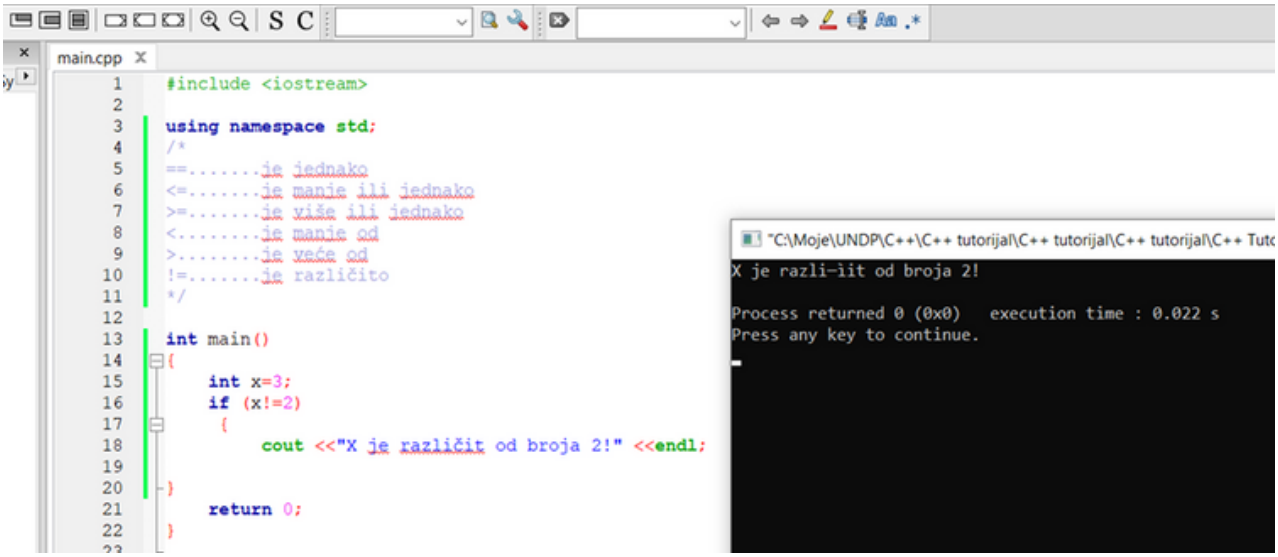
Program sa slike 41 radi sljedeće:

- Deklarišemo integer varijablu X i dodijelimo joj vrijednost 2
- Zatim radimo provjeru, ako je varijabla X jednaka broju 2, ispiše se tekst "X je jednak broju 2!". Ova provjera jednakosti radi se znakom ==.

Na slici 41 u komentarima koda možete vidjeti koje još opcije postoje osim == (jednako).



Slika 41: if naredba



Slika 42: if naredba za znak različito

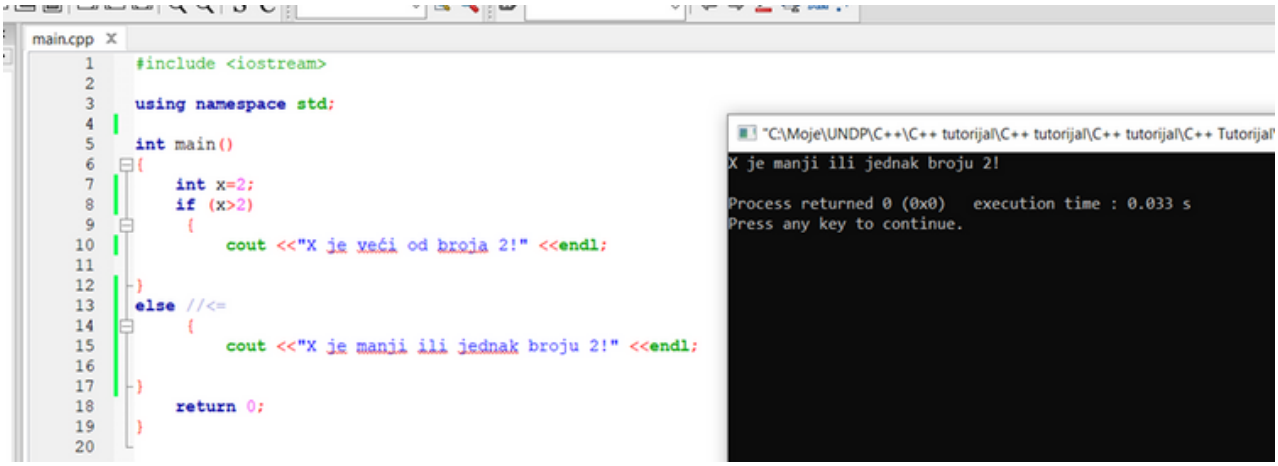
Važno je napomenuti kako se unutar jednog „if"-a može nalaziti još jedan „if" i unutar njega još jedan i tako možemo ići u dubinu koliko god poželimo. Naravno, pisanje desetak „if"-ova jednog unutar drugog je krajnje nečitko pa time i nema smisla. Postavljanje jednog „if"-a unutar drugog se zove ugniježđeni if.

3.7 NAREDBA ELSE

Else nam omogućuje izvršavanje koda ukoliko jedan ili više uslova unutar „if"-a nisu zadovoljeni. Dakle, da bismo izbjegli if naredbu za slučaj kada uslov nije zadovoljen, koristimo else naredbu.

Sintaksa:

else { kod koji će se izvršiti ukoliko uslov u if-u nije zadovoljen }



Slika 43: else naredba

U primjeru sa slike 43 „if" naredba će se izvršavati za one vrijednosti kada je x veće od broja 2. U svim ostalim slučajevima, izvršavaće se else naredba tj. kada je x manje ili jednako broju 2.

Lako je uočiti kako nam „else" omogućava puno jednostavnije i sažetije pisanje koda. Važno je napomenuti kako se unutar svakog „else"-a može nalaziti novi „if" i novi „else" i unutar u dubinu možemo ići koliko god poželimo.

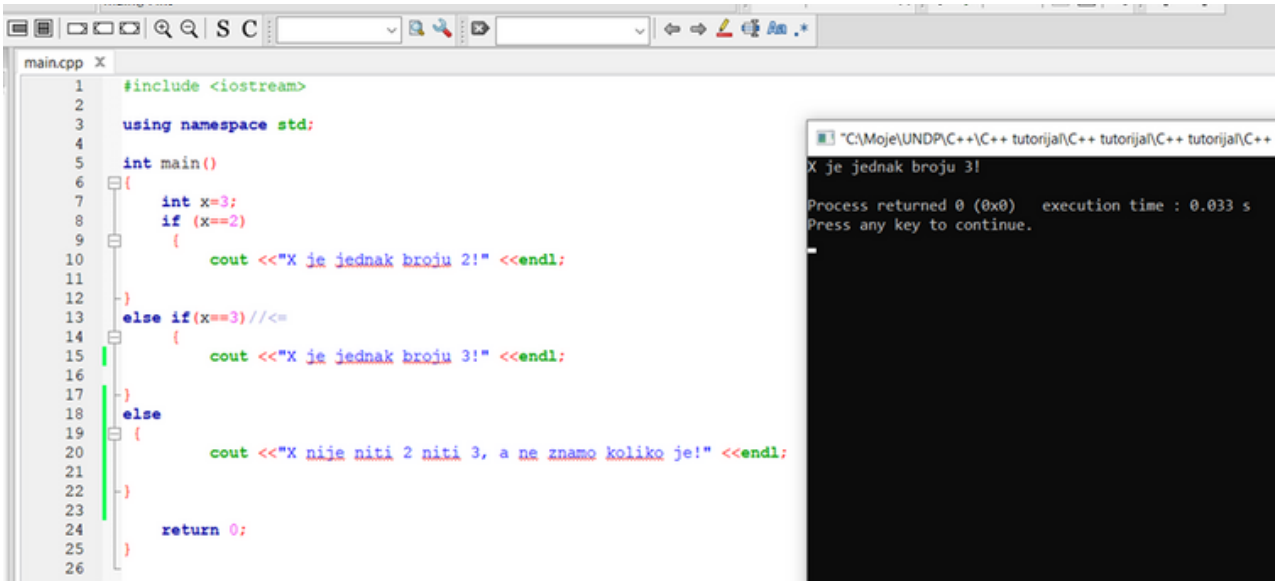
Pokušamo li upisati „else" bez „if"-a dobićemo sintaksnu pogrešku jer moramo reći kompajleru koji je primarni uslov i ukoliko on nije izvršen, što da se izvrši. Suprotno „else"-u, „if" može biti zapisan bez „else"-a.

3.8 NAREDBA IF...ELSE

If...else naredba je proširenje if naredbe. If naredba omogućava izvršavanje nekog koda ukoliko je uslov zadovoljen, a ukoliko nije, neće se izvršiti ništa. Ako bismo htjeli napisati takav kod da, ukoliko uslov nije zadovoljen, da se izvrši neki drugi kod, onda koristimo if...else naredbu.

Sintaksa if...else naredbe:

```
if (logički izraz) {
        ...
}
else if (logički izraz) {
        ...
}
else if (logički izraz) {
        ...
}
```



Slika 44: if...else naredba

3.9 UNOS VARIJABLI

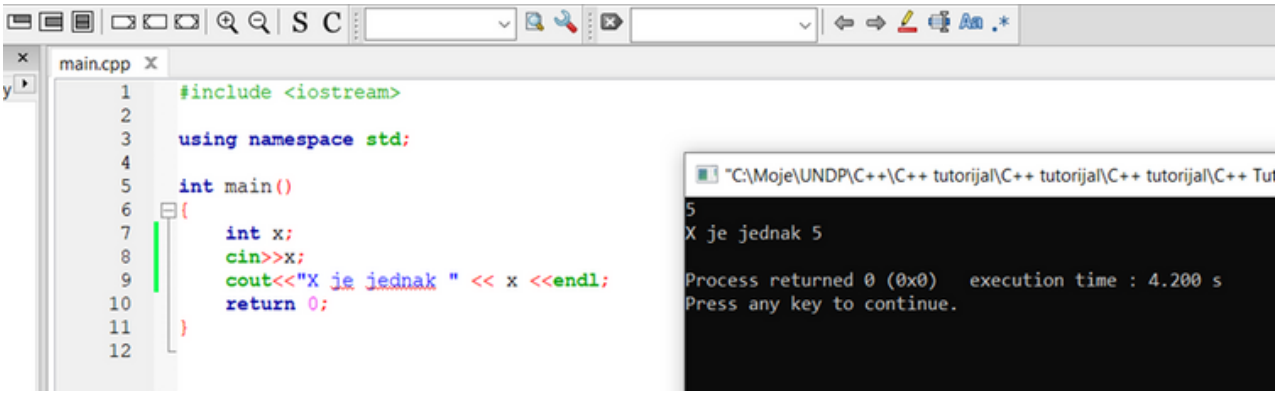
Da bismo ispisali podatke u programskom jeziku c++ koristi se naredba cout (koja je već prije i spomenuta), na sljedeći način:

Cout << „Ovo će se ispisati".

Prilikom izvršenja toga, na ekranu će se ispisati tekst „Ovo će se ispisati".

Međutim, programi koji imaju samo mogućnosti ispisa i nisu baš nešto korisni, tako da je pored ove naredbe za neki osnovni rad potrebna i naredba za unos podataka, a to je naredba cin.

Cin >> x;// naredba za unos varijable



Slika 45: cin naredba

Program sa slike 45 prikazuje način kako se unosi neka varijabla. Ovdje je pokrenuti program zahtijevao unos varijable x, gdje je unesena vrijednost 5. Program je tada ispisao tekst „X je jednak 5".

3.10 PONAVLJANJE

Nakon završetka poglavlja, čitatelju bi trebali biti poznati sljedeći pojmovi:

- Aritmetički operatori
- Tipovi podatka
- Ključne riječi
- Petlje i kontrole toka
- Unos varijabli

4 NAPREDNE NAREDBE

4.1 NAREDBA SWITCH

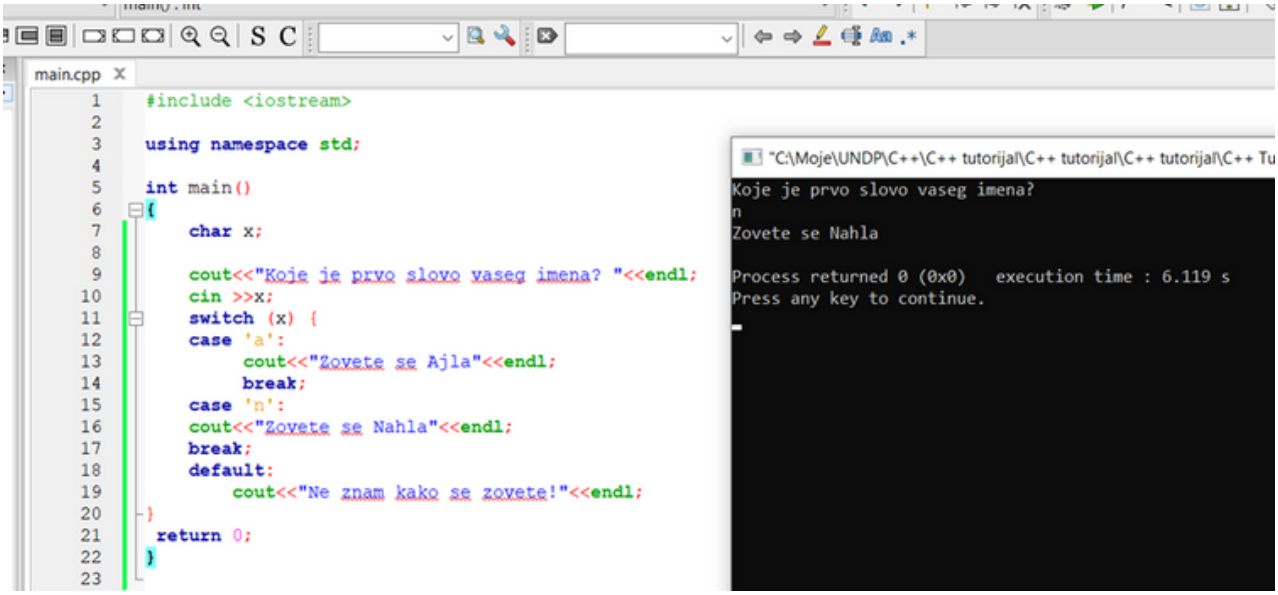
Ova naredba služi za provjeravanje više uslova odjednom i izvršavanja onoliko radnji koliko je uslova zadovoljeno odnosno tačno. Ova naredba je korisna ukoliko trebamo provjeriti više uslova, te je veoma slična if...else naredbi i koristi se isto kao i ta naredba s tim da je kod pregledniji ukoliko se radi o više različitih uslova.

Sintaksa switch naredbe:

```
switch ( test )
{
Slučaj 1:
{
kod koji će se izvršiti ako je ovaj uslov broj 1 zadovoljen;
break;
}
Slučaj 2:
{
kod koji će se izvršiti ako je ovaj uslov broj 2 zadovoljen;
break;
}
Slučaj 3:
{
kod koji će se izvršiti ako je ovaj uslov broj 3 zadovoljen;
break;
}
default:
{
kod koji će se izvršiti ako niti jedan od gornjih uslova nije
zadovoljen;
}
}
```

Naredba radi na sljedeći način:

- Switch naredba se evaluira jednom
- Vrijednost izraza se upoređuje s vrijednostima svakog slučaja
- Ako postoji podudaranje, pridruženi blok koda se izvršava
- Ključne riječi break i default su opciono opisane u nastavku



Slika 46: switch naredba

Kod sa slike 46 koristeći switch naredbu radi provjeru imena.

Default ključna riječ specificira neki kod za pokretanje ako nema podudaranja.

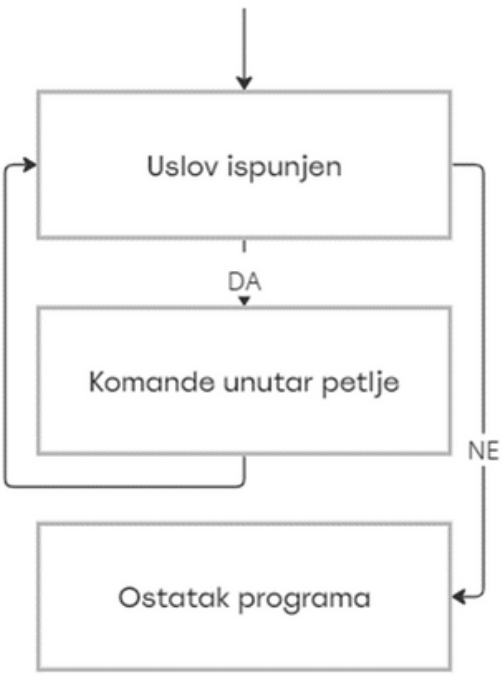
„Break“ naredba govori našem programu da izađe iz petlje odnosno u našem slučaju da prekine sa izvođenjem „switch“ naredbe.

4.2 NAREDBA WHILE

Ova petlja nam služi za definisanje ciklusa s poznatim i nepoznatim brojem ponavljanja.

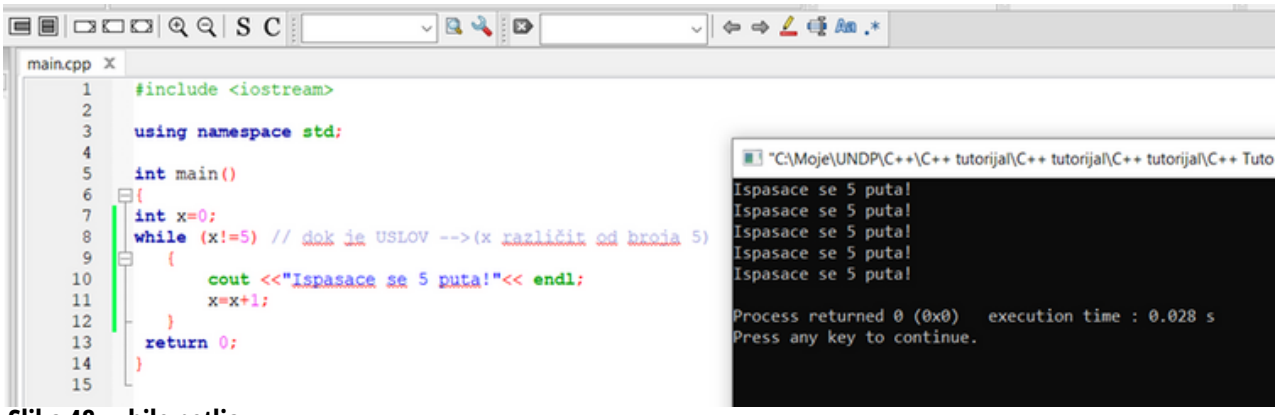
Sintaksa:

```
while (uslov)
{
naredbe;
}
```



Slika 47. Dijagram while petlje

Ključna reč while označava petlju. Petlja je dio koda koji se izvršava dokle je USLOV te petlje zadovoljen. Dakle, kada dođemo do petlje u kodu, računar provjerava da li je USLOV ispunjen, ako jeste, on izvršava KOMANDE UNUTAR PETLJE. Nakon što ih izvrši, vraća se ponovo do USLOVA, ako je on i dalje ispunjen, KOMANDE UNUTAR PETLJE se ponovo izvršavaju. Sve se to ponavlja dokle god uslov ne postane netačan.



Slika 48. while petlja

4.3 NAREDBA DO WHILE

Već iz samog imena možete zaključiti kako je ova vrsta petlje slična prethodno objašnjenoj „while" petlji. Sintaksa im je poprilično slična uz jedan dodatak bloka koda. U „do while" petlji prvo navodimo kod koji želimo da se izvrši, a u drugom dijelu navodimo uslov koji treba biti zadovoljen. Kako znamo da se naš program izvodi od najgornje linije prema dolje možemo uočiti kako u „do while" petlji to znači da će prvo doći do dijela u kojem se opisuje koji kod se treba izvesti, a tek nakon toga dolazi do dijela koda gdje je naveden uslov. Upravo to je svojstvo „do while" petlje i to je jedina petlja za koju kažemo da će se sigurno izvršiti barem jednom.

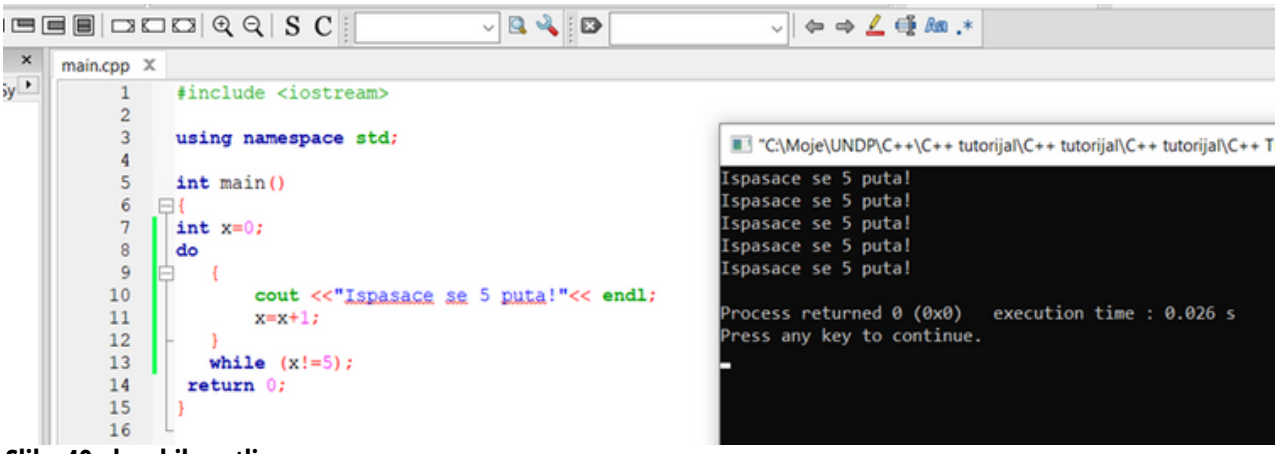
Sintaksa:

do

{ kod koji će se izvršiti

}

while (uslov);



Slika 49. do while petlja

4.4 NAREDBA FOR

For petlja svoju vrlo čestu primjenu nalazi prilikom izvršavanja istog koda više puta. Sintaksa je nešto komplikovanija za shvatiti naspram ostalih petlji, no nakon nekoliko korištenja nećete imati problema.

Sintaksa:

for (inicijalizacija; uslov; promjena vrijednosti) naredba;.

ili korištenjem blokova:

for (inicijalizacija; uslov; promjena vrijednosti) {

naredbe;

}

Uslov mora biti logički izraz, dok inicijalizacija i promjena vrijednosti mogu biti bilo kakvi izrazi. Petlja će se izvršavati dok je uslov tačan.

Slika 50. Dijagram for petlje

4.5 PONAVLJANJE

Nakon završetka poglavlja, čitatelju bi trebali biti poznati sljedeći pojmovi:

- Naredba switch
- Petlja
- While
- Do while
- For

5 FUNKCIJE, VARIJABLE I OPERATORI

5.1 UVOD U FUNKCIJE

Funkcije su jedno veliko poglavlje i njihova primjena je od sasvim jednostavnih aplikacija do krajnje kompleksnih aplikacija i računarskih igara. Funkcija u programiranju predstavlja najčešće skup radnji. Funkcije su blokovi koda koji se pokreću samo kada su pozvani.

Prednosti funkcija:

- Nema dupliciranja koda,
- Kod je lakši za čitati,
- Kod je lakši za održavati,
- Lakše je pronaći i ispraviti greške.

Uzmemo li za primjer ljudsko tijelo. Funkcije tijela su micanje lijeve ruke, micanje desne ruke, disanje, ramišljanje..... Dakle, naše tijelo je main funkcija, a sve ostalo su mnoge „akcije".

Sintaksa:

tip_podatka_koji_funkcija_vraća ime_funkcije (parametri_ako_ih_ima)

{

Kod koji se izvršava return naredba (osim u slučaju ako je tip podatka „void")

}

Napisaćemo funkciju Disanje.

Za početak za prvu funkciju treba nam povratni tip funkcije. S obzirom na to da naša prva funkcija neće vraćati ništa, stavićemo ključnu riječ void. Nakon void, dajemo ime funkciji. Nakon toga su zagrade koje služe za parametar (za prvu funkciju radićemo bez parametara).

Ova funkcija se sastoji iz 4 dijela:

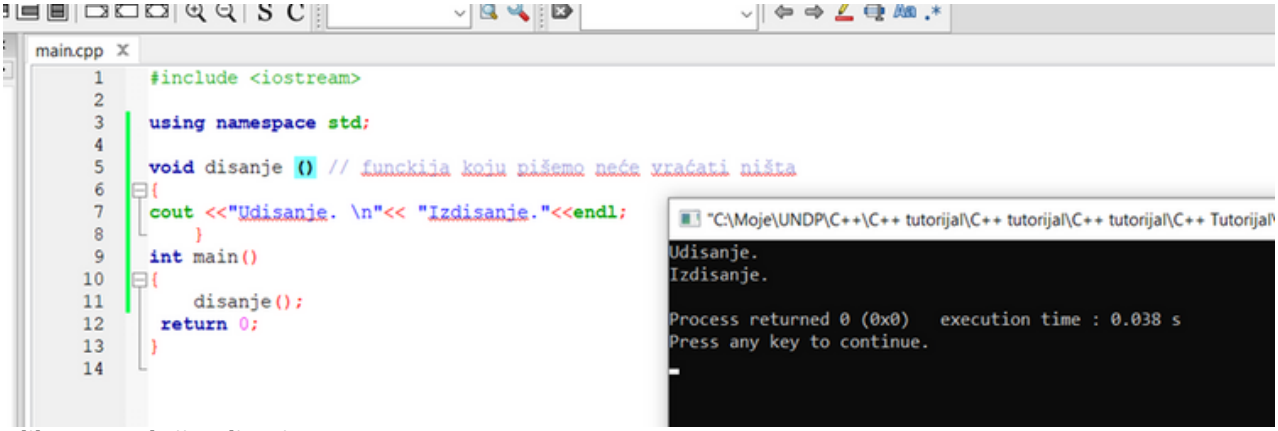
1. Disanje - Naziv funkcije.
2. void - Tip podatka koji funkcija vraća.
3. () - Parametri koje funkcija prima.
4. cout <<"Udisanje. \n"<< "Izdisanje."<<endl;- Tijelo funkcije.

Dakle, funkciju je potrebno definisati iznad mjesta njenog prvog poziva (u ovom slučaju prije same main funkcije) kako bi se ona mogla pozvati.

Poziv funkcije se desio u 11. liniji koda u primjeru sa slike 51 (disanje()).

Poziv funkcije se sastoji iz 2 dijela:

1.disanje - Naziv funkcije.
2.() - Parametri koje funkcija prima (ova funkcija ne prima nikakve parametre).



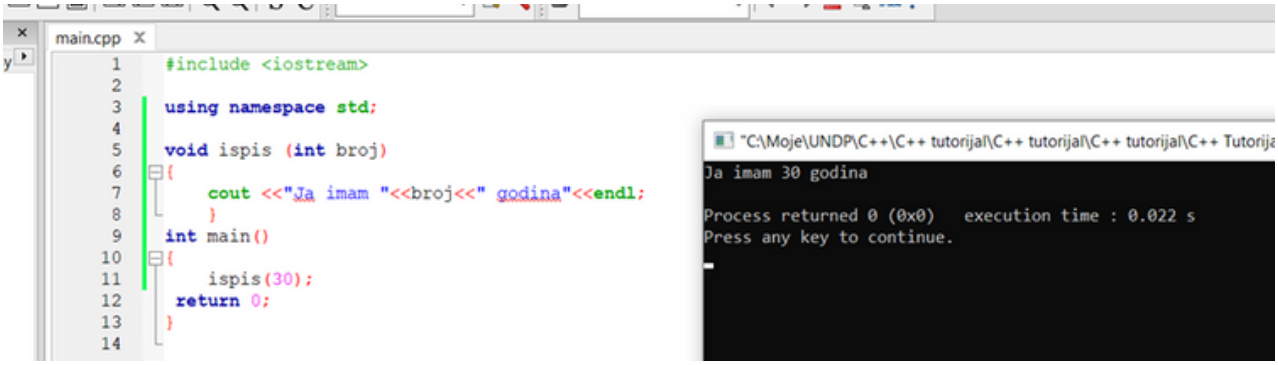
Slika 51. Funkcija „disanje"

5.2 FUNKCIJE S JEDNIM PARAMETROM

Funkcija koju smo posmatrali u prethodnom primjeru je veoma jednostavna. Ne prima nikakve parametre, niti ima povratnu vrijednost. Sada slijedi primjer funkcije koja ima parametar.

Ako pogledamo deklaraciju ove funkcije, vidimo da parametri koje funkcija prima više nisu (). (int broj) naglašava da pri pozivu ove funkcije moramo proslijediti neki integer funkciji.

U ovom slučaju možemo primijetiti da se i poziv funkcije razlikuje. Sada smo u zagradi pri pozivu funkcije obavezni proslijediti i jedan broj. Varijabla broj u funkciji je kopija vrijednosti koja je proslijeđena pri pozivu funkcije.

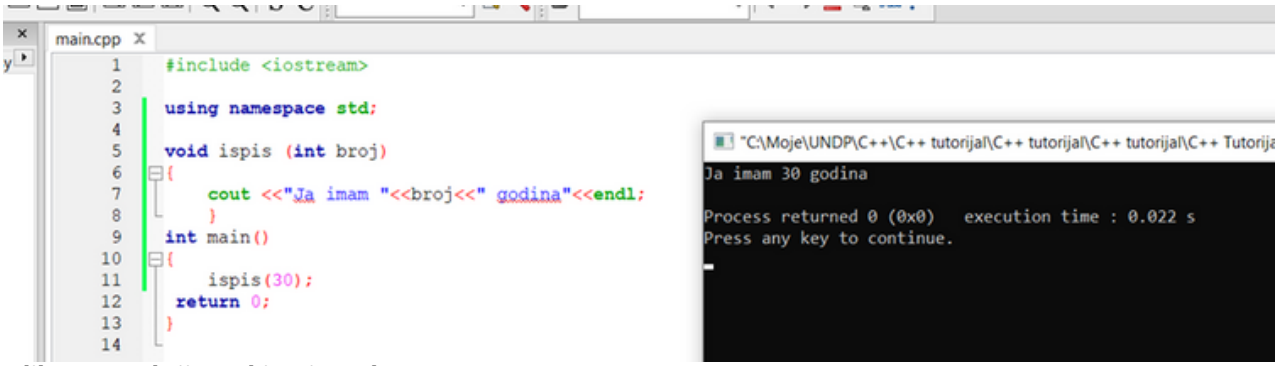


Slika 52. Funkcija „ispis" s jednim parametrom

5.3 FUNKCIJE S VIŠE PARAMETARA

Funkcija može imati i više parametara. Kao što postoje različiti tipovi podataka za varijable, tako razlikujemo i tipove podataka za funkcije. Same vrste tipova su iste, a sama vrsta tipa koju navedemo prije naziva funkcije govori našem programu kakvu će vrijednost vratiti naša funkcija odnosno što će funkcija vratiti kao rezultat preko naredbe „return". Ono što je važno za uočiti i zapamtiti je naredba „return" koja se nalazi na kraju funkcije. Naredba „return" simbolizira kraj funkcije i tu navodimo što želimo da nam funkcija vrati odnosno navodimo rezultat nakon što je funkcija obavila svoj dio koda koji ćemo koristiti u daljnjem dijelu aplikacije. Tip podatka koji funkcija vraća u „return" naredbi mora odgovarati tipu podatka koji smo naveli prilikom deklaracije funkcije.
U primjeru na slici 53 prikazana je funkcija koja sabira dva broja i vraća rezultat sabiranja:

return rezultat;



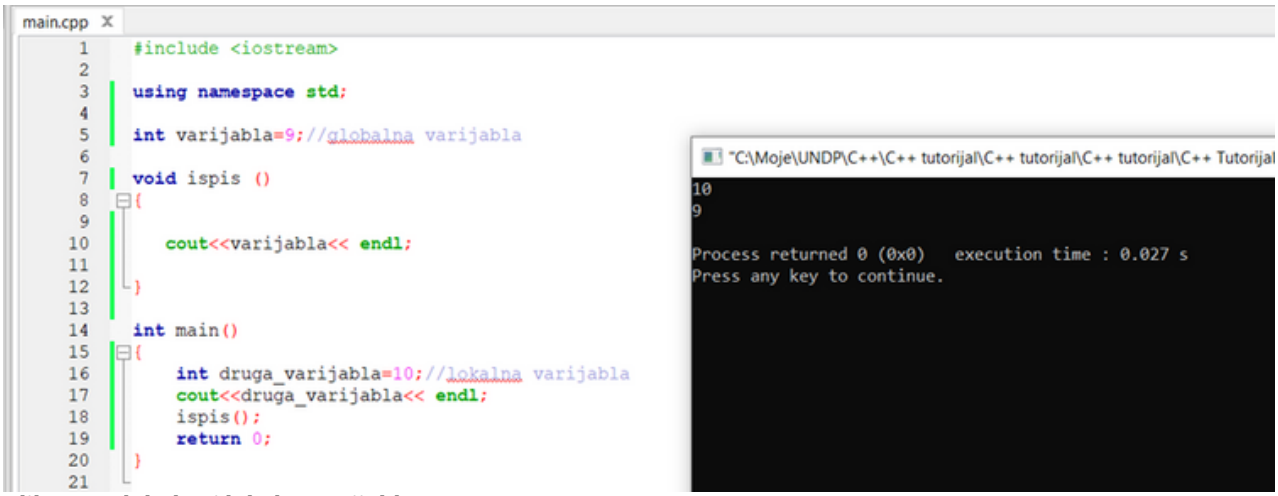
Slika 53. Funkcija „sabiranje" s dva parametra

5.4 LOKALNE I GLOBALNE VARIJABLE

Varijable koje smo do sada koristili bile su lokalne i vrijedile su samo u funkciji main(). Globalne varijable najavljuju se izvan svih funkcija i mogu se koristiti u cijelom programu, dok su lokalne dostupne samo unutar funkcije u kojoj su najavljene. Ukoliko želimo iz funkcije vratiti više vrijednosti, to ne možemo učiniti naredbama „return" jer funkcija završava izvršavanje nakon izvršavanja naredbe „return", no možemo pohranom u globalne varijable. Doseg globalne varijable može se suziti na datoteku u kojoj je definisana upotrebom oznake static, npr.

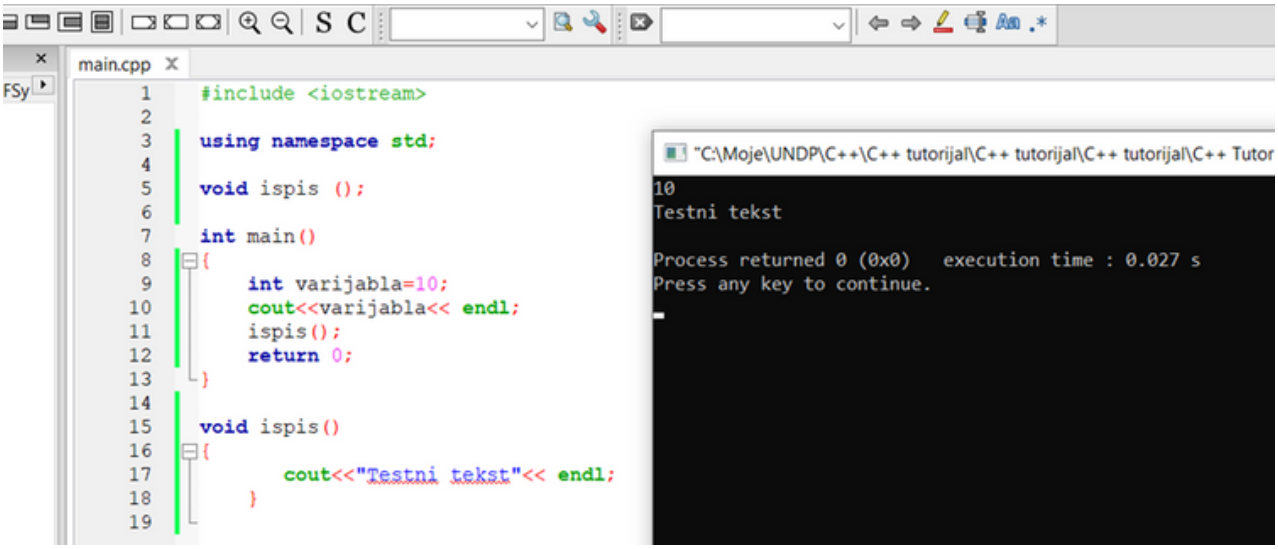
static double x;

Varijabla tipa static ima doseg globalne, a trajanje lokalne varijable, tj. vrijednost statičke lokalne varijable ostaje sačuvana do ponovnog poziva funkcije.



Slika 54. Globalne i lokalne varijable

Ukoliko funkciju deklarišemo iza „main" funkcije, tada ćemo morati koristiti prototip funkcije. Prototip funkcije je prvi red funkcije (tip podatka koji vraća funkcija, ime funkcije, parametri) koji se stavlja prije „main" funkcije. Razlog tome je što prilikom izvođenja programa, kada kompajler dođe do dijela gdje pozivamo neku funkciju koju smo napisali iza „main" funkcije, kompajler neće znati što želim od njega jer se programi izvode s vrha prema dnu i u trenutku kada naiđe na našu funkciju on se još nije susreo s njom pa time niti ne zna što mu pokušavamo reći. Korištenjem prototipa funkcije mi kompajleru govorimo da će se negdje u našem kodu pojaviti funkcija određenog imena i da smo je deklarisali negdje iza „main" funkcije. Kada kompajler u našem kodu naiđe na ime funkcije koju smo deklarisali iza „main" funkcije i postavili joj prototip, kompajler će znati da je ta funkcija negdje i zapisana te će ići potražiti je i naravno upotrijebiti.



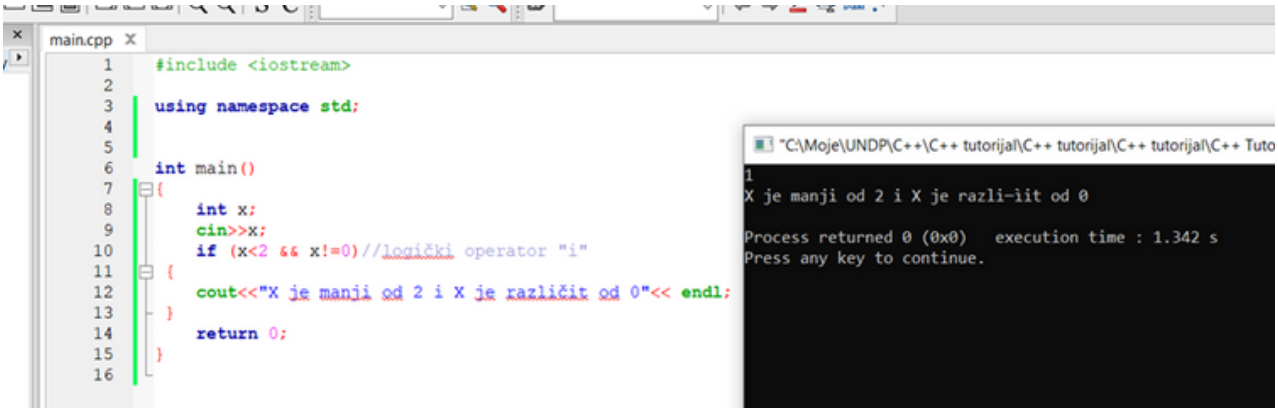
Slika 55. Prototip funkcije

5.5 LOGIČKI OPERATORI

Za rad s logičkim podacima postoje logičke funkcije. Logičke funkcije se zapisuju logičkim operatorima. Logički operatori mogu biti unarni i binarni. Opis logičkih operatora možete vidjeti na slici 56.

Logički operatori		
Oznaka operatora	Funkcija	Operator
!	Negacija (unarni operator koji 1 pretvara u 0 i obratno)	NOT
&&	Logički I (binarni operator)	AND
||	Logički ILI (binarni operator)	OR

Slika 56. Logički operatori



Slika 57. Logički operator „i"

5.6 PONAVLJANJE

Nakon završetka poglavlja, čitatelju bi trebali biti poznati sljedeći pojmovi:

- Funkcije
- Funkcije s jednim parametrom
- Funkcije s dva parametra ili više njih
- Naredba return
- Lokalne i globalne varijable
- Prototip funkcije
- Logički operatori

6 NIZOVI

6.1 NIZOVI

U programiranju ćete se često sresti s upotrebom polja odnosno nizova. Korištenjem nizova možemo na smislen nam način grupisati više podataka istog tipa. Npr. želimo li izračunati, ali i pohraniti ocjene iz testa učenika/ca nekog razreda jedan od načina je korištenje nizova. Nizove možete zamisliti kao jednu vreću u kojoj se nalaze čokolade, ali samo čokolade iste vrste. Tako možemo imati dvije vreće, jednu za crnu čokoladu a drugu za bijelu čokoladu, a unutar svake vreće ćemo staviti podvrste čokolada. U vreću s crnom čokoladom možemo staviti čokoladu s lješnjacima i s orasima dok u onu s bijelim čokoladama možemo staviti čisto mliječnu čokoladu i čokoladu s kokosom. Svaka od tih čokolada je element svog niza odnosno element svoje vreće i svaka ima svoj redni broj odnosno svoj indeks. Postoje jednodimenzionalni i dvodimenzionalni nizovi.

Sintaksa:
tip_podatka ime_niza [veličina_niza];

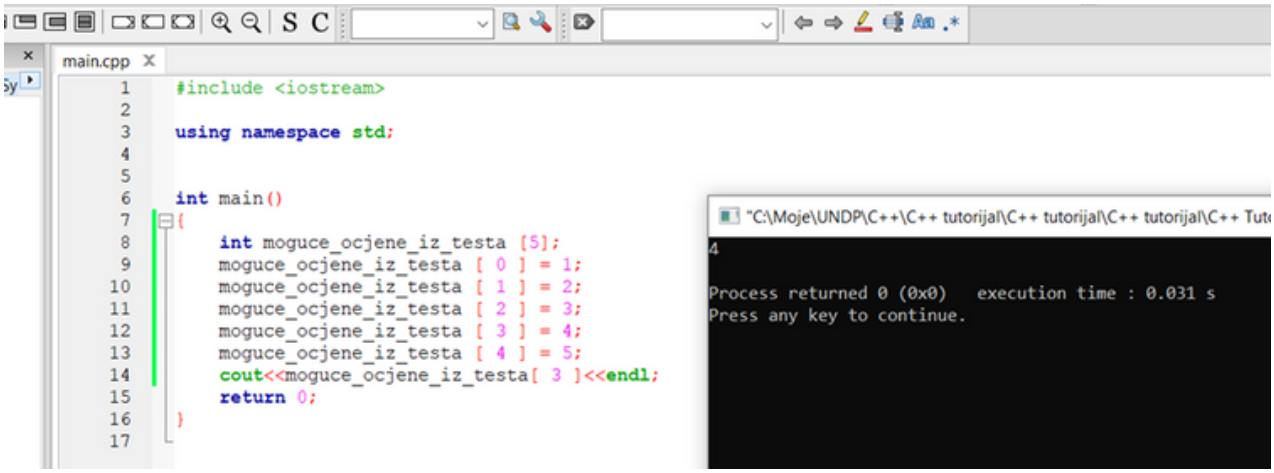
Primjer niza i pojašnjenje :
int moguce_ocjene_iz_testa [5];

Sada smo deklarisali niz realnih brojeva čije je ime „moguce_ocjene_iz_testa" i koji u sebi može sadržavati 5 elemenata. Taj niz će nam služiti za upis mogućih ocjena iz testa kako i samo ime kaže (vrijednosti od 1 do 5 uključujući granične brojeve). Sada se postavlja pitanje kako se pridjeljuju vrijednosti nizu. Svaki niz, zavisno od veličine koju mu unaprijed odredimo, ima određeni broj elemenata. U našem slučaju taj broj je 5. Svakom elementu niza se pristupa (čitanje vrijednosti nekog elementa ili upis vrijednosti u neki element) pomoću indeksa tog elementa. Želimo li vizualno predstaviti naš niz to će izgledati kao u narednoj tabeli:

Indeks	0	1	2	3	4
Vrijednost	1	2	3	4	5

Kao što vidite, naš prvi element nema indeks niza 1 (jedan), već ima indeks niza 0 (nula). U programerskom odnosno računarskom svijetu brojevi počinju od broja 0 (nula) a ne od broja 1 (jedan) kako su ljudi navikli brojati. Vjerovatno će takav način brojanja biti pomalo zbunjujući na početku, no nakon nekoliko vježbi će zasigurno ući pod normalno shvaćanje.

Navedeni primjer niza možete vidjeti na slici 58.



Slika 58. Nizovi

U polju realnih brojeva koje smo nazvali „moguce_ocjene_iz_testa" naš prvi element koji ima indeks 0 (nula) sada ima vrijednost 1 (jedan). Drugi element s indeksom 1 (jedan) ima vrijednost 2 (dva). Proces se nastavlja do zadnjeg elementa niza.

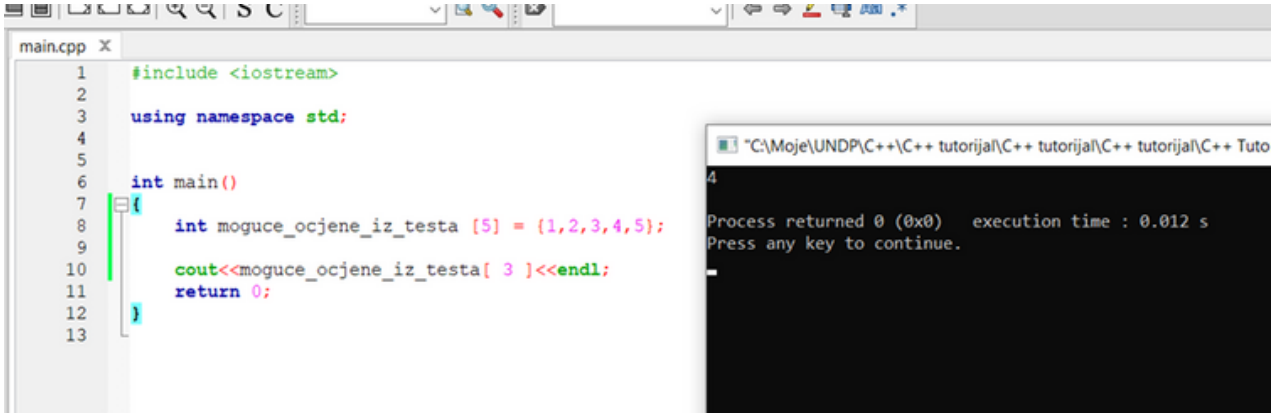
Ispis vrijednosti nekog elementa u našem primjeru će ovako izgledati:

cout<<moguce_ocjene_iz_testa[3]<<endl;

6.2 ALOCIRANJE NIZOVA

Ukoliko su nam podaci niza već zadani, niz se može deklarisati i na drugačiji način. Slijedi primjer za zadatak iz prethodnog poglavlja za ispis ocjena iz testa:

int moguce_ocjene_iz_testa [5] = {1,2,3,4,5};



Slika 59. Alociranje nizova

Osnovni nedostaci nizova su to što niz nikada ne može biti veći od zadane vrijednosti tj. ako smo mi u prethodna dva primjera rekli da korisnik/ca može unijeti 5 vrijednosti, a zatreba mu/joj opcija da unese i šest, to nije moguće, jer se unaprijed mora definisati broj elemenata niza.

Također može biti i obrnuto, ako mi osiguramo 100 elemenata niza, a koristimo samo 5, program će naravno raditi, ali se alocira velika količina memorije bespotrebno.

6.3 UNOS I ISPIS VRIJEDNOSTI NIZA

Zamislimo li niz koji ima 10 ili čak više elemenata i da svaki element moramo „ručno" čitati, lako je za shvatiti da takav pristup nema smisla. Ovdje nam pomaže korištenje „for" petlje. Kako „for" petlja kreće brojati od nekog broja do nekog broja mi možemo napraviti petlju koja će brojati od 0 (nula) do 5 (pet) kako bi nam ispisala sve vrijednosti nekog niza. Važno je zapamtiti da indeksi elemenata u nizu kreću s indeksom 0 (nula) a ne od broja 1 (jedan).

Primjer za čitanje iz niza s 5 elemenata:

```
for ( int i = 0; i<5;i++)
{
cout << moguce_ocjene_iz_testa [ i ] << endl;
}
```

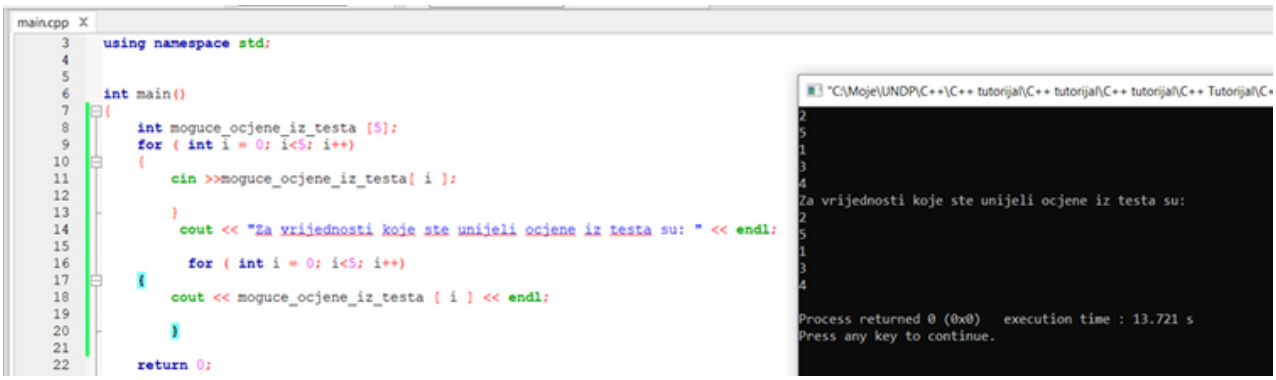
Detaljno objašnjenje primjera:

1. Stvaramo „for" petlju s početkom od broja 0 (nula) do broja 5 (ne uključujući broj 5 jer smo rekli da mora biti striktno manje od 5. Mogli smo napisati i „<=4" što bi bilo potpuno identično), uz povećanje nakon svakog koraka (jedan korak se smatra kada petlja prođe kroz sav kod napisan u njenom tijelu).
2. Provjera da li je vrijednost varijable „i" manja od 5 (0 < 5)
3. Ispiši na ekran vrijednost elementa koji se nalazi u nizu imena „moguce_ocjene_iz_testa" na indeksu čiju vrijednost predstavlja vrijednost varijable „i",što je u ovom slučaju 0. Nakon ispisa poruke preskoči u novi red.
4. Uvećaj vrijednost varijable „i" za jedan (i = 1)
5. Provjera da li je vrijednost varijable „i" manja od 5 (1 < 5)
6. Ispiši na ekran vrijednost elementa koji se nalazi u nizu imena „moguce_ocjene_iz_testa" na indeksu čiju vrijednost predstavlja vrijednost varijable „i",što je u ovom slučaju 1. Nakon ispisa poruke preskoči u novi red.
7. Uvećaj vrijednost varijable „i" za jedan (i = 2)
8. Provjera da li je vrijednost varijable „i" manja od 5 (2 < 5)
9. Ispiši na ekran vrijednost elementa koji se nalazi u nizu imena „moguce_ocjene_iz_testa" na indeksu čiju vrijednost predstavlja vrijednost varijable „i",što je u ovom slučaju 2. Nakon ispisa poruke preskoči u novi red.
10. Uvećaj vrijednost varijable „i" za jedan (i = 3)
11. Provjera da li je vrijednost varijable „i" manja od 5 (3 < 5)
12. Ispiši na ekran vrijednost elementa koji se nalazi u nizu imena „moguce_ocjene_iz_testa" na indeksu čiju vrijednost predstavlja vrijednost varijable „i", što je u ovom slučaju 3. Nakon ispisa poruke preskoči u novi red.
13. Uvećaj vrijednost varijable „i" za jedan (i = 4)
14. Provjera da li je vrijednost varijable „i" manja od 5 (4 < 5)
15. Ispiši na ekran vrijednost elementa koji se nalazi u nizu imena „moguce_ocjene_iz_testa" na indeksu čiju vrijednost predstavlja vrijednost varijable „i", što je u ovom slučaju 4. Nakon ispisa poruke preskoči u novi red.
16. Uvećaj vrijednost varijable „i" za jedan (i = 5)
17. Provjera da li je vrijednost varijable „i" manja od 5 (5 < 5)
18. Izađi iz „for" petlje (prekini njeno izvršavanje)
19. Nastavi s daljnjim izvođenjem iz programa

Još ispravniji način gornjeg koda bi bio da smo unutar „for" petlje drugačije postavili vrijednost do koje će se petlja izvršavati. Kako smo prije stvaranja niza imali konstantnu varijablu imena „velicina" u koju smo pohranili vrijednost o veličini našeg niza, mogli smo (i trebali) ovako napisati uslov „for" petlje:

```
for ( int i = 0; i<velicina;i++)
{
cout << moguce_ocjene_iz_testa [ i ] << endl;
}
```

Rezultat i opis koraka su identični, ali je ovo puno kvalitetniji pristup. Dođe li do promjene veličine niza sada smo se osigurali da će se i „for" petlja pravilno izvršiti odnosno da će ispisati sve vrijednosti našeg niza. Primjerice, proširimo li polje na 6 elemenata tako da element s indeksom 0 (nula) ima vrijednost 0 (nula) što simbolizira da učenik/ca nije pisao/la test, a ostavimo staru „for" petlju, tada bi nam se ispisale vrijednosti mogućih ocjena 0, 1, 2, 3, 4, ali ocjena 5 se ne bi ispisala jer se ona sada nalazi na indeksu broj 5 do kojeg „for" petlja neće doći.



Slika 60. Upis i ispis vrijednosti niza s for petljom

6.4 PONAVLJANJE

Nakon završetka poglavlja, čitatelju bi trebali biti poznati sljedeći pojmovi:

- Niz
- Indeks niza
- Alociranje niza
- Glavni nedostatak niza
- Upis i ispis niza s for petljom

7 LITERATURA

1. C++ programiranje za apsolutne početnike.....Jakopec Ratko, ing
2. Osnove programiranja.....Alen Jakupović, Sabrina Šuman
3. Uvod u programiranje.....Saša Fajković
4. https://www.w3schools.com/cpp/default.asp
5. http://digis.edu.rs/course/view.php?id=299

Priručnik je izrađen u saradnji sa Pedagoškim zavodom Zeničko-dobojskog kantona, a u okviru projekta "Boljom upravom do bržeg ekonomskog rasta" (EGG2) kojeg podržava i finansira Vlada Kraljevine Norveške, a provodi Razvojni program Ujedinjenih nacija (UNDP) u BiH. Sadržaj priručnika ne odražava nužno stavove Vlade Kraljevine Norveške, niti UNDP-a.