Lekcija 01

Uvodna razmatranja, Uvod u C

Miljan Milošević

UVODNA RAZMATRANJA, UVOD U C

	01	02	03	04
Uvod	**Uvodna razmatranja**	**Faze generisanja programa**	**Osnovna anatomija C programa**	**Program sa više fajlova**
	➢ *Istorijat i razvoj programskih jezika C i C++*	➢ *Razvoj programa u programskim jezicima C i C++* ➢ *Kompajleri* ➢ *Pretprocesor, kompajler i linker* ➢ *Korišćenje integrisanih razvojnih okruženja u cilju razvoja C i C++ programa*	➢ *Anatomija programa u C-u* ➢ *Rezervisane reči i dozvoljeni karakteri* ➢ *Prvi program u C-u* ➢ *Osnovi o operatorima i promenljivama* ➢ *Komentari*	➢ *Fajlovi zaglavlja* ➢ *Pisanje sopstvenih header fajlova*

UVODNA RAZMATRANJA, UVOD U C

05

Standardni ulaz i izlaz u C-u

- *Šta je standardni ulaz i izlaz?*
- *Funkcije scanf i printf*
- *Osnovni simboli konverzije pri radu sa printf i scanf*
- *Osnovni primer ulazno/izlaznih funkcija*
- *I/O funkcije za rad sa karakterima getchar() i putchar()*

06

Tipovi podataka

- *Definisanje novih tipova podataka*

07

Promenljive i konstante

- *Konverzija promenljivih*

08

Tipovi operatora

- *Uvod u pokazivače*

09

Upotreba razvojnog okruženja Visual Studio 2010

- *Visual Studio projekat sa jednim fajlom*
- *Korišćenje komandne linije*

UVODNA RAZMATRANJA, UVOD U C

10

Vežba - Ulaz i izlaz različitih podataka

- *Primeri 1*
- *Primeri 2*
- *Primeri 3*
- *Primeri 4*
- *Primeri 5*

11

Vežba - Tipovi podataka, promenljive, operatori

- *Osnovni primeri sa tipovima podataka*
- *Upotreba različitih tipova podataka i operatora*
- *Upotreba adresnog i indirektnog operatora*
- *Upotreba binarnih operatora*
- *Upotreba binarnih operatora u kombinaciji sa operatorom dodele*

12

Zadaci za samostalan rad

- *Zadaci za samostalno vežbanje*

UVOD

Ova lekcija treba da ostvari sledeće ciljeve:

U okviru ove lekcije studenti se upoznaju sa osnovnim pojmovima u vezi C i C++ programskih jezika:

- Istorijat programskih jezika C i C++.
- Faze u razvoju softvera: upoznavanje sa kompajlerom, pretprocesorom i linkerom.
- Pisanje prvog programa i analiza osnovne anatomije programa u C programskom jeziku.
- Opis integrisanih razvojnih okruženja (IDE) za kreiranje C/C++ aplikacija.
- Ulazno/izlazne funkcije u C-u za rad sa različitim tipovima podataka.
- Tipovi podataka u C-u, Promenljive i konstante, Definisanje novih tipova podataka.
- Tipovi operatora i osnovi o pokazivačima.

Najveći broj ljudi u toku svakodnevnog rada na računaru ne mora da poznaje dobro šta je kompjuterski program i šta on ustvari radi. Međutim, za razliku od običnih ljudi, programer mora da poznaje odgovore na ova i slična pitanja, kao na primer šta je programski jezik, i kako ustvari programski jezici C i C++ funkcionišu.

Ljudi koriste mozak (memoriju) za smeštanje i pristupanje informacijama. Po istom principu rade i računari. Međutim, računarska memorija je znatno drugačija nego naša. Informacije, odnosno podaci, imaju različite oblike tj. forme. Neki podaci su numerički, drugi su tekstualni itd. Ljudi koriste imena da bi se obraćali jedni drugima. Na sličan način, da bi ste se u kodu obratili nekom delu informacije, neophodno je da to takođe uradite preko imena. Za imenovanje informacije među 1000 drugih informacija u memoriji, koristimo promenljive.

Mnogi računarski programi imaju za cilj da izvršavaju složena izračunavanja. Stoga, računari, osim mogućnosti da skladište ogromne količine podataka, imaju i mogućnost da računaju mnogo brže i preciznije od nas.

Uvodna razmatranja

pojmovi jezika, podela jezika, istorija, tok razvoja

- *Istorijat i razvoj programskih jezika C i C++*

01

ISTORIJAT I RAZVOJ PROGRAMSKIH JEZIKA C I C++

Jezik C je nastao sasvim spontano u Bell-ovim laboratorijama 1972. godine. C++ je nastao 1983. godine kao proširenje jezika C u cilju podrške objektno orijentisanom programiranju

Programski jezik C je nastao spontano, bez projekata, specifikacija i zahteva u Belovim laboratorijama. Naziv C se pojavio po tome što je C bio naslednik prethodne verzije za interno korišćenje koji se zvao B. Jezik C razvili su 1972. godine Denis Riči (Dennis Ritchie) i Brajan Kernigan (Brian Kernighan) evolucijom jezika BCPL i B.

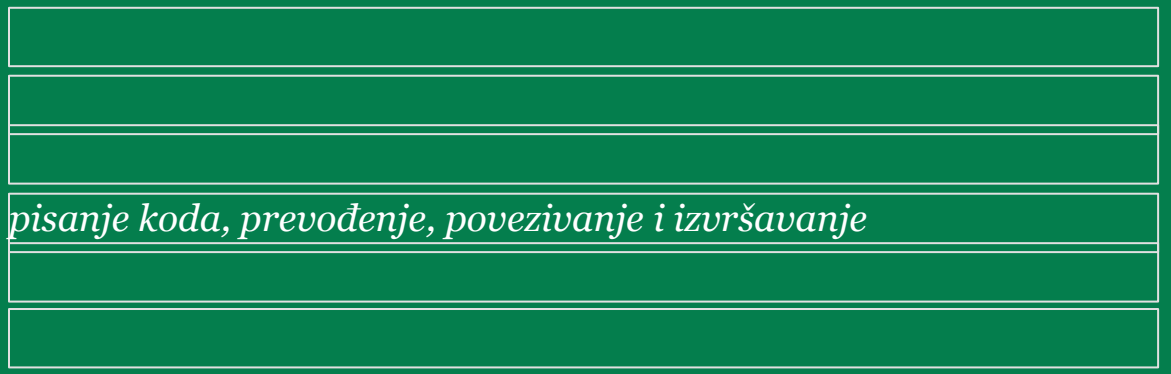
Danas C ima široku primenu, a neki od primera upotrebe C jezika su: operativni sistemi, kompajleri, asembleri, editori teksta, mrežni drajveri, moderni programi, baze podataka, interpretatori, itd.

Početkom osamdesetih godina takođe u Belovim laboratorijama Bjorn Stroustrup radeći na proširivanju jezika C razvio je suštinski nov jezik koga je nazvao “C sa klasama”. Autor je želeo da poboljša jezik C razvojem podrške objektno orijentisanom programiranju. U 1983. godini taj naziv je promenjen u C++.

Jezik C++ se razlikuje od običnog C-a pre svega podrškom objektno-orijentisanom programiraju. Ali, u njemu ima i niz novih mogućnosti, koje nisu objektnog karaktera, zbog kojih je pisanje programa koji nisu objektne prirode udobnije realizovati u C++-u nego u C-u.

C++ je danas nesporno jedan od najmoćnijih, ali i najkompleksniji programski jezik. Zbog kompaktnosti programa, brzine izvršavanja i prenosivosti predstavlja nezamenjiv alat svakog profesionalca. Zbog ovih osobina zauzima jedno od dominantnih mesta u svetu profesionalnog programiranja.

Faze generisanja programa



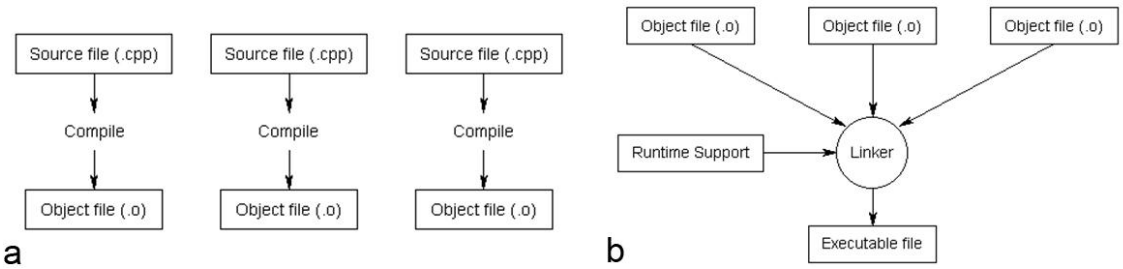
- *Razvoj programa u programskim jezicima C i C++*
- *Kompajleri*
- *Pretprocesor, kompajler i linker*
- *Korišćenje integrisanih razvojnih okruženja u cilju razvoja C i C++ programa*

02

Razvoj programa se sastoji iz sledećih koraka: pisanje programa, prevođenje, povezivanje programa (linkovanje), punjenje memorije i izvršavanje programa

Razvoj programa se sastoji iz sledećih koraka:

- Pisanje programa - Pisanje programa predstavlja kucanje programa na ekranu nekog profesionalnog editora i zapisivanje u fajl. Ekstenzija fajlova koji sadrže izvorni kod (source code) za programski jezik C je *.c, a u programskom jeziku C++ je *.cpp.
- Prevođenje - Izvorni kod napisan u editoru je razumljiv čoveku ali ne i računaru. Zato je neophodno izvršiti prevođenje programa na mašinski jezik koji je razumljiv procesoru. Tako se dobija datoteka (fajl) sa ekstenzijom *.obj.
- Povezivanje programa - Povezivanje programa (linkovanje, eng. linking) se vrši sa ciljem uključivanja standardnih funkcija iz C/C++ biblioteka, kao i novo napravljenih funkcija od strane samog programera.



Slika-1 a) Postupak prevođenja programa na jezik razumljiv računaru; b) povezivanje programa u izvršni fajl

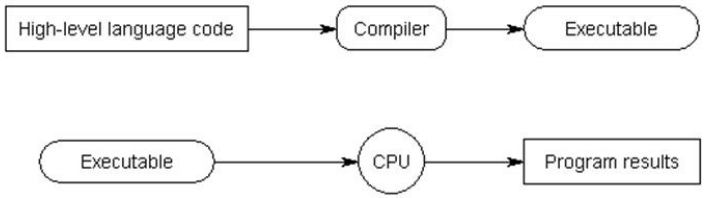
- Punjenje memorije - predstavlja fazu neposredno pre samog izvršenja programa i služi za učitavanje svih neophodnih vrednosti za inicijalizaciju promenljivih.
- Izvršenje programa - predstavlja startovanje aplikacije koja je napravljenja prevođenjem i povezivanjem. Ekstenzija tako dobijenog programa je u Windows okruženju je *.exe.

U nastavku će detaljno biti opisane neke od faza u razvoju softvera.

KOMPAJLERI

Kompajler čita izvorni kod i proizvodi nezavisan izvršni fajl koji procesor (CPU) direktno izvršava

Kompajler je program koji, uprošćeno rečeno, čita izvorni kod i proizvodi nezavisan izvršni fajl koji CPU procesor može direktno da razume i izvršava. Kada je kod jednom pretvoren u izvršni fajl, nema potrebe za ponovnim pozivanjem kompajlera. Na sledećoj slici 2 je uprošćeno prikazan postupak kompajliranja programa:



Slika-2 Postupak povezivanja programa korišćenjam kompajlera i postupak izvršavanja programa na procesoru

Kompajleri šalju poruke o sintaksnim greškama u delu izvornog koda koji se ne može prevesti u izvršnu verziju. Kompajler jednostavno taj deo koda ne razume jer se ili ne može protumačiti ili je nekonzistentan sa ostatkom koda. Takođe, kompajleri daju upozorenja o delu izvornog koda koji možda može da pravi greške kod izvršavanja programa. Izvršni program se ne može dobiti dok se sintaksne greške ne isprave.

Uprošćeno, postupak generisanja programa korišćenjem prethodno opisanih koraka, može biti prikazan na sledeći način. Pomoću kompajlera se izvorni kod programa napisan u C/C++, npr. datoteka hello.c prevodi u oblik hello.exe, koja je izvršna. Takođe, postoji i međufaza, gde se između izvorne verzije i izvršne verzije, kreira tzv. objektna verzija tj. „objektni kod“. Izvorni kod programa ima nastavak imena ‘.cpp’. Međuverzija između izvornog koda i izvršnog koda je tzv. objektna verzija, *object code*, koja ima oznaku ‘.obj’ ili ‘.o’. Izvršna verzija programa ima nastavak .exe, ili je bez ikakvog nastavka (ekstenzije).

PRETPROCESOR, KOMPAJLER I LINKER

Pretprocesor je poseban i nezavisan program koji se izvršava pre nego što kompajler počne da prevodi program. Linker konačno povezuje objektne fajlove u izvršnu verziju programa

Da bi se neki izvorni kod programa preveo u izvršni fajl koji kompjuter može da izvršava, koriste se tri programa,

- Pretprocesor
- Kompajler
- Linker

Pretprocesor je najbolje posmatrati kao poseban i nezavisan program koji se izvršava pre nego što kompajler počne sa prevođenjem (kompajliranjem) programa. Svrha pretprocesora je da procesira direktive. Pretprocesor je program koji pregleda izvorni kod programa da bi našao pretprocesorske direktive, npr. direktivu #include. Zatim, pretprocesor insertuje u izvorni kod sve fajlove koji su uključeni preko direktiva #include. Npr. stdio.h fajl standardne C biblioteke se ubacuje u izvorni kod. Kompajler je program koji prevodi izvorni kod u mašinski jezik, i ovaj fajl koji se dobija posle kompajliranja se zove objektni fajl, object file, i ima nastavak u imenu .obj. Kompajler ne samo da prevodi izvorni kod u mašinski jezik, već i proverava spelovanje i gramatiku jezika C/C++, jer ovaj jezik ima svoja pravila pisanja reči i gramatiku iskaza.

Konačno, „linker“ je program koji se koristi da poveže run-time biblioteku (gde se npr. smeštaju instrukcije za delovanje na eksterni hardver npr. monitor ili tastatura, itd.) sa objektnim fajlom, da bi se dobio program koji se može izvršavati i koji ima nastavak u imenu .exe. Inače, napomenimo, run-time biblioteka je uključena u integrisano razvojno okruženje (integrated development environment - IDE), pa pri instalaciji integrisanog okruženja obezbeđujemo i run-time biblioteku ukoliko ona nije već postojala u okviru operativnog sistema. Neki kompajleri direktno prave .exe fajl, i tada nije potrebno koristiti posebno program linker.

KORIŠĆENJE INTEGRISANIH RAZVOJNIH OKRUŽENJA U CILJU RAZVOJA C I C++ PROGRAMA

Integrisano razvojno okruženje objedinjava sve korake odnosno faze razvoja softvera u jedinstvenu celinu što olakšava pisanje i testiranje programa

Umesto da koristite različite programe za prethodno opisane funkcije (razvoj koda, kompajliranje, linkovanje, kao i debug-iranje), softverski paket poznat kao integrisano razvojno okruženje (integrated development environment – IDE) integriše sve ove funkcije u jednu jedinstvenu celinu što olakšava pisanje i testiranje programa. Uz tipično razvojno okruženje, korisnik dobija koristan editor koda koji u sebi sadrži opcije za numeraciju linija koda i isticanje (highlighting) grešaka u sintaksi. Jedan tipičan IDE će automatski generisati parametre koji su neophodni za prevođenje i povezivanje vašeg koda u izvršnu verziju (executable), pa i kada se radi o uključivanju mnogobrojnih fajlova. U slučaju da treba da debug-irate odnosno ispravljate greške u vašem programu, IDE sadrži i debug-er, tj. integrisani program koji omogućava praćenje izvršavanje vašeg koda liniju po liniju što olakšava pronalaženje grešaka. Osim ovih osnovnih opcija, IDE obično uključuje i veliki broj drugih pomoćnih alata, kao što je integrisani program za pomoć (help), dovršavanje reči (autocomplete), itd.

Neki od najpoznatijih integrisanih razvojnih okruženja, čije se korišćenje istovremeno preporučuje u okviru predmeta C/C++ programski jezik, su:

• Microsoft's Visual C++	Windows	X
• Code::Blocks	Windows	Linux
• Netbeans C/C++	Windows	Linux

Osnovna anatomija C programa

osnovna sintaksa, rezervisane reči, dozvoljeni karakteri

- *Anatomija programa u C-u*
- *Rezervisane reči i dozvoljeni karakteri*
- *Prvi program u C-u*
- *Osnovi o operatorima i promenljivama*
- *Komentari*

03

ANATOMIJA PROGRAMA U C-U

Jedan C program se u osnovi sastoji iz: pretprocesorkih naredbi, funkcija, promenljivih, iskaza, izraza i komentara

Da bi se napisao bilo koji program u nekom programskom jeziku potrebno je poznavati neka osnovna pravila. Pravila koja važe prilikom pisanja programa u jeziku C (a takođe i u C++-u) su:

- Program se sastoji iz službenih reči.
- Prisutni su karakteri i specijalni znakovi.
- Sve naredbe se završavaju sa tačka-zarez (;).
- C je jezik slobodnog formata.
- Svi programi u jeziku C uključuju i fajlove sa ekstenzijom *.h u kojima se nalaze prototipovi (deklaracije) funkcija standardnih biblioteka.

Jedan C (takođe i C++) program se u osnovi sastoji iz sledećih delova:

- Pretprocesorkse naredbe
- Funkcije
- Promenljive
- Iskazi i izrazi
- Komentari.

REZERVISANE REČI I DOZVOLJENI KARAKTERI

Rezervisane reči ne mogu biti korišćene kao imena promenljivih, konstanti i drugih identifikatora

U jeziku C postoje 32 rezervisane reči i to:

auto	default	enum	if	short	typedef
break	do	extern	int	sizeof	union
case	double	float	long	static	unsigned
char	else	for	register	struct	void
continue	goto	return	switch	volatile	const

Ove rezervisane reči ne mogu biti korišćenje u druge svrhe osim u one za koje su namenjene. Uvek se pišu malim slovima. Može se javiti još rezervisanih (službenih) reči u određenim kompajlerima (prevodiocima) za jezik C/C++ i one se moraju dokumentovati sa detaljnim objašnjenjem u odgovarajućoj pratećoj dokumentaciji.

Karakteri koji se koriste prilikom pisanja programa u jeziku C su:

- A – Z (velika i mala slova engleske abecede)
- 0 – 9 (cifre)
- #%&!_{}[]<>| (simboli)
- Tab dugme . , : ; ’ $ ” (specijalni simboli)
- + - / * = (osnovne računske operacije).

PRVI PROGRAM U C-U

C program se sastoji iz funkcija i promenljivih, i svaki program mora imati main() funkciju

Program u C-u se sastoji od funkcija i promenljivih. Glavna funkcija je **main** od koje počinje izvršavanje programa. Svaki program mora imati **main** funkciju, koja zatim poziva sve druge funkcije. Najbolji način da se nauči neki programski jezik je da se odmah počne sa pisanjem u njemu. Napišimo jedan kratki program koji će odštampati na ekranu poruku “Nole Sampion!”

```
include <stdio.h>
void main()
{
 printf("Nole Sampion!\n");
}
```

Prvi red je uključivanje standardne biblioteke stdio.h pomoću pretproceserkse naredbe #include. main je glavna funkcija i već smo napomenuli da je obavezna za svaki C program. Ukoliko nemamo argumente za neku funkciju onda pišemo program sa otvorenim i zatvorenim zagradama (). Velike zagrade (vitičaste zagrade {}) predstavljaju početak i kraj nekog programskog bloka u jeziku C kao što je u ovom slučaju funkcija main. Da bi smo ispisali nešto na ekranu koristimo funkciju printf. Ono što se ispisuje na ekranu se navodi unutar navodnika “ … ”. Na kraju samog teksta koji se ispisuje na ekran se nalaze dva znaka: \n. To je ustvari jedan znak koji omogućava prelazak u novi red. Na kraju se kao i svaka naredba u jeziku C stavlja tačka zarez (;) da bi kompajler znao gde je kraj naredbe.

Ukoliko bi smo zapisali naš program pod imenom Nole.c dobili bismo nakon prevođenja i povezivanja program koji startovanjem ispisuje poruku na ekranu (konzolu): “Nole Sampion!”.

OSNOVI O OPERATORIMA I PROMENLJIVAMA

C program se može sastojati iz više fajlova ali samo jedan fajl može da sadrži funkciju main()

Analiziraćemo jos jedan jednostavan primer da bi stekli bolju predstavu o strukturi C programa:

```
#include <stdio.h>
void main()
{
 int broj1, broj2, rezultat;
 broj1 = 8;
 broj2 = 16;
 rezultat = broj1 + broj2;
 printf("Zbir %d + %d = %d \n", broj1, broj2, rezultat);
}
```

U programskom jeziku C kao i u drugim jezicima, kod se može nalaziti u jednom ili više fajlova. Samo jedan od tih fajlova mora da sadrži funkciju main. Da bi mogao da koristi funkcije iz sistemske biblioteke u programu se mora navesti direktiva pretprocesoru kojoj prethodi znak #. Pretprocesor vrši analizu i pripremu programa pre kompajliranja. U navedenom primeru ponovo pozivamo direktivu #include <stdio.h> kojom se uključuju sistemske funkcije (standardne ulazne/izlazne funkcije itd), a takođe koristimo glavnu funkciju main. Opis funkcije u C-u se sastoji iz zaglavlja i tela funkcija. Zaglavlje funkcije u ovom prostom obliku čine ime funkcije i zagrade (). Iza zaglavlja se navodi telo funkcije čiji se početak označava znakom {, a kraj znakom }. Opisni operator:

```
int broj1, broj2, rezultat;
```

definiše tri nove promenljive koje su celobrojnog tipa. Operator dodele: broj1 = 8; obezbeđuje dodelu celobrojne vrednosti 8 promenljivoj broj1. Izlazni operator:

```
printf("Zbir %d + %d = %d \n", broj1, broj2, rezultat);
```

obezbeđuje ispis rezultata, pri čemu znaci %d obezbeđuju prevodiocu informaciju o poziciji i formatu gde će on oštampati vrednosti promenljivih broj1, broj2, rezultat. Standardna izlazna funkcija printf će biti razmatrana nešto kasnije.

KOMENTARI

Uvek pišite komentare jer niste jedini koji će raditi na složenom softveru. Ostavite nešto korisno iza vas i olakšajte posao drugima

Komentari predstavljaju pomoćni tekst u C/C++ programu koji se ignoriše od strane kompajlera. Komentarom se u C-u smatra svaki niz simbola koji počinje parom znakova /* a završava se znakovima */ kao što je prikazano u nastavku:

```
/* my first program in C */
```

Kod Microsoft kompajlera postoji još jedan oblik komentara koji se naziva jednolinijski komentar. On počinje simbolom // i proteže se do kraja linije, tj. ukazuje da kompajler treba da ignoriše sve što se nalazi od simbola do kraja linije. Na primer:

```
printf("Hello world!"); // Everything from here to the right is ignored.
```

Običaj je da se jednolinijski komentari koriste sa ciljem da se na brz način opiše jedna od linija koda.

```
printf("Hello world!"); // printf lives in the stdio.h library
printf("It is very nice to meet you!"); // these comments make the code hard to read
printf("Yeah!"); // especially when lines are different lengths
```

Pravilno pisanje i korišćenje komentara

Postoje tri glavna razloga zbog kojih se pišu komentari:

- Da se na nivou biblioteka, programa ili funkcija opiše čemu ustvari odgovarajuće biblioteke, programi i funkcije služe.
- U odgovarajućim bibliotekama, programu ili funkcijama, komentare treba koristiti sa ciljem da se opiše na koji način će odgovarajući delovi koda da ostvare svoj cilj (odnosno da reše odgovarajući problem).
- Na nivou iskaza i izjava, komentare treba pisati sa cijem da se opiše zašto kod radi odgovarajuću stvar, a ne da se opiše šta on ustvari radi.

Program sa više fajlova

sourse fajl, header fajl, direktiva include, standardna biblioteka

- *Fajlovi zaglavlja*
- *Pisanje sopstvenih header fajlova*

04

OSNOVI O KORIŠĆENJU VIŠE FAJLOVA

Sa porastom obima i kompleksnosti programa prostoji potreba da se kod podeli u više različitih fajlova u cilju što bolje organizacije

Kako programi postaju obimniji i kompleksniji, nije neobično podeliti delove koda u nekoliko različitih fajlova u cilju što bolje organizacije koda. Jedna od prednosti integrisanih razvojnih okruzenja (IDE) je da ona rad sa više fajlove čine mnogo lakšim i efikasnijim. U razvojnim okruženjima postoji opcija dodavanja novih fajlova koja naravno olakšava pisanje programa.

U sledećem primeru ćemo opisati korišćenje više fajlova pri pisanju programa. Pretpostavimo da imamo dva fajla, jedan je glavni "main.c" fajl, a drugi je pomoćni fajl "add.c" u kome se nalazi definicija neke funkcije za sabiranje dva broja:

add.c:

```
int add(int x, int y)
{
    return x + y;
}
```

main.c:

```
#include <stdio.h>
int add(int x, int y); // forward declaration using function prototype
int main()
{
    printf("The sum of 3 and 4 is: %lf", add(3, 4));

    return 0;
}
```

Treba napomenuti da u programskom jeziku C ne morate navesti deklaraciju funkcije add u fajlu sa glavnom main funkcijom. C kompajler će i bez navođenja deklaracija iskompajlirati program.

Fajlovi zaglavlja

standarna zaglavlja, direktiva include, standardna biblioteka

- *Osnovni o fajlovima zaglavlja*
- *Uključivanje header fajlova*
- *Korišćenje header fajlova standardne biblioteke*

04

OSNOVNI O FAJLOVIMA ZAGLAVLJA

Fajlovi zaglavlja sadrže deklaracije i makro definicije koje će biti korišćenje u drugim fajlovima sa izvornim kodom

Header fajl ili fajl zaglavlja je datoteka sa ekstenzijom *.h koja obično sadrži deklaracije funkcija i makro definicije koje će biti korišćenje u drugim fajlovima sa izvornim kodom. Postoje dve vrste fajlova zaglavlja i to: korisnički fajlovi i header fajlovi koji idu uz kompajler. Da bi se neki header fajl koristio u programu, neophodno je pre toga uključiti ga pomoću pretprocesorske direktive #include kao što je bilo prikazano u prvom primeru ove lekcije gde je uključen header fajl stdio.h koji je standardni header fajl C/C++ kompajlera.

Uključivanje header fajlova možemo bukvalno shvatiti kao kopiranje koda pre kompajliranja. Češća praksa u jeziku C (takođe i C++) je da se konstante, makroi, globalne promenljive, zaglavlja funkcija (prototipovi) smeste u header fajl, a da se zatim po potrebi taj fajl zaglavlja uključi tamo gde je to potrebno.

- **Pretprocesorska naredba include**

Fajlovi zaglavlja, bilo da su kreirani od strane programera, bilo da su deo standardne C (C++) biblioteke, uključuju se u program korišćenjem pretprocesorske direktive #include. Jedan od oblika je:

```
#include <file>
```

Ovaj oblik direktive include se koristi za sistemske fajlove zaglavlja. Kompajler u direktorijumu standardne biblioteke pre kompajliranja koda traži da li postoji datoteka pod imenom file. Drugi oblik korišćenja include direktive je:

```
#include "file"
```

Ovaj oblik se koristi za uključivanje fajlova zaglavlja napravljenih od strane korisnika (programera) i najčešće smeštenih u direktorijumu gde su i ostali izvorni fajlovi projekta. Na ovaj način se ukazuje kompajleru da pretraži direktorijum gde su i ostali fajlovi projekta u potrazi za datotekom “file”.

UKLJUČIVANJE HEADER FAJLOVA

Fajlovi zaglavlja standardne biblioteke se u program uključuju direktivom #include iza koje sledi ime biblioteke uokvireno znakovima < i >

Posmatrajmo sledeći program:

```
#include <stdio.h>
int main()
{
    printf("Hello, world!");
    return 0;
}
```

Ovaj program štampa poruku "Hello, world!" u konzolu pomoću funkcije printf. Međutim, u našem programu nigde ne postoji deklaracija ove funkcije pa se postavlja pitanje sta printf ustvari znači? Odgovor je da je printf deklarisan u okviru header fajla koji se naziva "stdio.h". Korišćenjem linije koda:

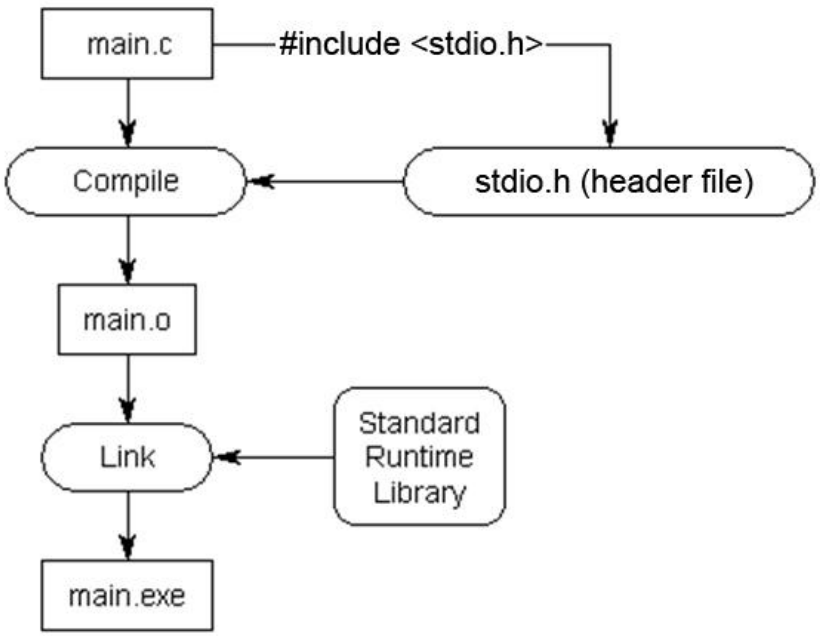
```
#include <stdio.h>
```

mi ustvari poručujemo kompajleru da treba da pronađe i pročita sve deklaracije koje se nalaze u fajlu sa zaglavljima po nazivu "stdio.h".

Treba imati na umu da se u fajlovima zaglavlja obično nalaze deklaracije, pa stoga oni ne definišu kako će nešto biti implementirano, a dobro je poznato da program neće moći da bude povezan (linkovan) ukoliko ne mogu da se pronađu implementacije nečega što koristite u programu. Stoga se postavlja jedno logično pitanje, a to je: ukoliko je funkcija printf samo definisana u fajlu zaglavlja stdio.h gde je ustvari njena implementacija? Naime, ona je implementirana u okviru izvršne biblioteke (runtime library), koja je automatski uključena u program prilikom faze povezivanja (linkovanja).

KORIŠĆENJE HEADER FAJLOVA STANDARDNE BIBLIOTEKE

Biblioteka se sastoji iz fajlova zaglavlja u kojima se nalaze deklaracije za sve što je proglašeno javnim i dostupnim korisnicima



Slika-1 Postupak uključivanja fajla zaglavlja u vaš program, za kojim sledi prevođenje i povezivanje programa u izvršnu verziju

Standardna izvršna biblioteka (Standard Runtime Library) je ustvari jedan paket koda (package of code) koji će po potrebi biti više puta korišćenjen u programima. Obično, biblioteka se sastoji iz fajlova zaglavlja u kojima se nalaze deklaracije za sve što je proglašeno javnim i dostupnim korisnicima, odnosno piscima programskog koda, i gotovih unapred iskopajliranih (precompiled) objekata čiji je kod već preveden na mašinski jezik. Ove biblioteke obično imaju ekstenziju .lib ili .dll na Windows platformi, odnosno .a ili .so ekstenziju na Unix-u. Zbog čegu su biblioteke iskompajlirane? Prvo, s obzirom da se izvršne biblioteke retko menjaju, njihovo kompaljiranje nije potrebno izvršiti često, a u većini slučajeva nijednom. To bi bio gubitak vremena da se one kompaljiraju svaki put kada pišete program. Drugi razlog je taj, s obzirom jer su prekompajlirani objekti već prevedeni na mašinski jezik, da se spreči da im korisnici pristupe ili menjaju izvorni kod, što može da bude veoma bitno kod specifičnih tipova poslova gde je neophodno zaštiti kod kao intelektualnu svojinu koju bi drugi mogli da prisvoje i ostvare koristi od toga.

Pisanje sopstvenih header fajlova

sopstveni header fajl, direktiva include, pravila pisanja

- *Osnovi o pisanju sopstvenih fajlova zaglavlja*
- *Faze u razvoju programa sa više fajlova*
- *Preporuke kod upotrebe header fajlova*

04

OSNOVI O PISANJU SOPSTVENIH FAJLOVA ZAGLAVLJA

Fajlovi zaglavlja koje sami napravimo se u program uključuju direktivom #include iza koje sledi ime fajla uokvireno znakovima “ i “ (#include “file.h“)

Karakteristika header fajlova je da oni mogu biti napisani jednom, a uključeni u onolikom broju drugih fajlova koliko je to potrebno. Pisanje sopstvenih fajlova je iznenadjujuće lak posao. Header fajl se sastoji iz dva dela. Prvi deo je rezervisan za pretprocesorske naredbe, a drugi deo je ustvari sadržaj header fajla u kome se nalazi deklaracija svih funkcija koje želimo da budu vidljive i korišćene od strane delova koda u drugim fajlovima. Svi korisnički napravljeni header fajlovi treba da budu sa ekstenzijom *.h. U nastavku je dat primer korišćenja header fajla add.h:

```
int add(int x, int y); // function prototype for add.h
```

U fajlu add.h se nalazi deklaracija funkcije, dok se u sledećem fajlu, add.c, nalazi njena definicija (o funkcijama u narednoj lekciji):

```
int add(int x, int y)
{
    return x + y;
}
```

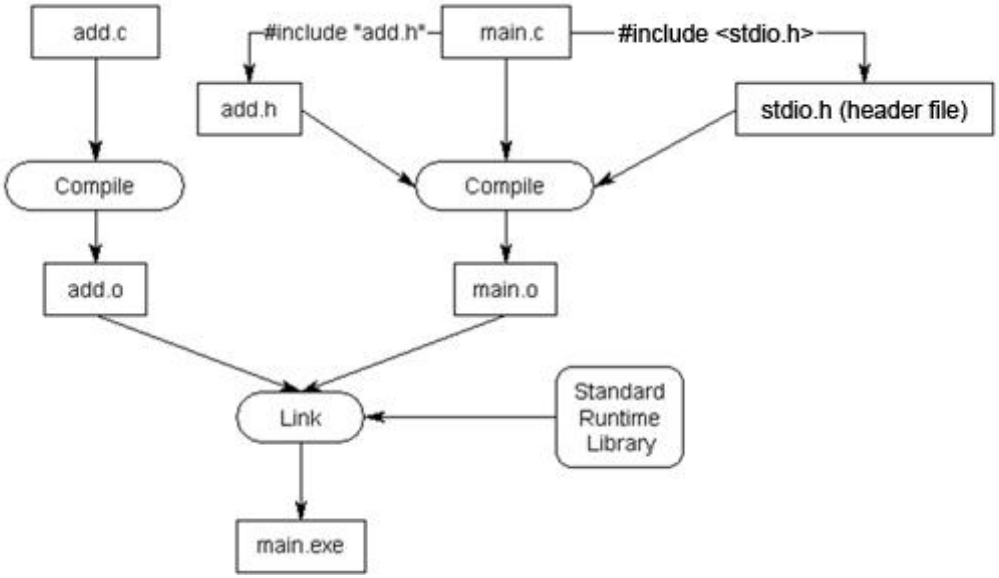
U cilju uključivanja ovog fajla u glavnom fajlu, neophodno je da ga uključimo odgovarajućom #include naredbom. Fajl main.c koji uključuje add.h header fajl će imati sledeći oblik:

```
#include <stdio.h>
#include "add.h" // this brings in the declaration for add()

int main()
{
    printf("The sum of 3 and 4 is %lf",add(3, 4));

    return 0;
}
```

FAZE U RAZVOJU PROGRAMA SA VIŠE FAJLOVA

Pri radu sa više fajlova kompajler prevodi svaki pojedinačni fajl, pri čemu se može javiti situacija da prilikom prevođenja jednog fajla treba potražiti informaciju definisanu u drugom fajlu

U nastavku je dat klasičan primer faza u razvoju složenog C programa sa više fajlova, gde se koriste sopstveni fajlovi zaglavlja kao i fajlovi standardne biblioteke, slika 1:



Slika-1 Primer složenog C programa sa više fajlova. Uključivanje sopstvenih i standardnih fajlova zaglavlja u program, za kojima sledi prevođenje i povezivanje programa u izvršnu verziju

PREPORUKE KOD UPOTREBE HEADER FAJLOVA

Poznavanje uobičajene prakse pri upotebi fajlova zaglavlja olakšava pisanje složenih softvera i smanjuje rizik da se pojave greške u kodu

U nastavku su date preporuke pri korišćenju header fajlova:

- Uvek koristiti #include pretposecorsku direktivu za uključivanje drugih fajlova.
- Ne definišite promenljive unutar header fajlova osim ako se radi o konstantama. Header fajlovi treba jedino da služe kao fajlovi sa deklaracijama.
- Ne vršiti definiciju funkcija unutar header fajlova osim ako su definicije trivijalne. Na ovaj način fajlovi postaju teži za čitanje od strane drugih programera.
- Svaki header fajl treba da obavlja specifični tip zadatka, i da bude što je moguće nezavisan u odnosu na druge fajlove. Na primer, možete ubaciti sve deklaracije koje se odnose na neki zadatak A u okviru A.h fajla, odnosno sve deklaracije koje se odnose na neki zadatak B u B.h. Tako, ubuduće ukoliko želite da izvršite zadatak A neophodno je da uključite samo A.h fajl i da zanemarite sve deklaracije vezane za zadatak B.
- Uvek uključiti koliko je god moguće više drugih fajlova u vaš header fajl.

Standardni ulaz i izlaz u C-u

Ulazne funkcije, izlazne funkcije, standardni fajlovi

- *Šta je standardni ulaz i izlaz?*
- *Funkcije scanf i printf*
- *Osnovni simboli konverzije pri radu sa printf i scanf*
- *Osnovni primer ulazno/izlaznih funkcija*
- *I/O funkcije za rad sa karakterima getchar() i putchar()*

05

ŠTA JE STANDARDNI ULAZ I IZLAZ?

Ulaz se obično odnosi na tastaturu ili datoteku, dok se izlaz odnosi na nešto što treba biti prikazano na ekranu, konzoli ili zapisano u datoteku na hard disk-u

Kada kažemo ulaz obično mislimo na unos nekih podataka u program. Unos se može ostvariti ili iz fajla ili iz komandne linije. C programski jezik omogućuje unos podataka korišćenjem velikog seta ugrađenih (standardnih bibliotečnih) funkcija koje čitaju podatke i dopremaju ih tamo gde se to traži u programu. Kada kažemo izlaz (output) obično mislimo da nešto treba biti prikazano na ekranu, oštampano na štampaču ili zapisano u neku datoteku. C funkcije standardne biblioteke obezbeđuju ovu funkcionalnost (pa i štampanje podataka u binarne fajlove).

Standardni fajlovi

C programski jezik tretira sve uređaje kao fajlove (datoteke). Stoga se uređaj kao što je ekran tretira kao neki fajl pa se sledeća tri fajla automatski otvaraju pri izvršavanju programa da bi se obezbedio pristup tastaturi ili ekranu.

Standardni fajl	Pokazivač na fajl (adresa fajla)	Uredjaj
Standardni ulaz(input)	stdin	Tastatura
Standardni izlaz(output	stdout	Ekran
Standard error	stderr	Vaš ekran

Slika-1 Standardni fajlovi koji se koriste za ulazno izlazne operacije

Pokazivač na fajl (čuva adresu fajla) je tip podataka koji služe da bi se pristupilo fajlu u svrhu čitanja ili upisivanja (o pokazivačima će biti reči na kraju ove lekcije, a detaljnije u lekciji o pokazivačima). U nastavku će biti opisani postupci koji se koriste u programskom jeziku C u cilju učitavanja vrednosti sa tastature i štampanja podataka na ekran.

FUNKCIJE SCANF I PRINTF

Funkcija scanf() učitava podatke sa standardnog ulaza (tastatura) dok funkcija printf() štampa podatke na ekranu. Neophodno je uključiti stdio.h header fajl u cilju njihovog korišćenja

Da bi smo mogli da koristimo najosnovnije programe napisane u C-u neophodno je da upoznamo sredstva koja omogućavaju da saopštimo programu ulazne podatke, da nad njima bude izvršena obrada, i da računar, u formi koja njemu odgovara, prikaže izlazne rezultate. Funkcije printf i scanf nisu deo C jezika i zato se koriste iz standardne sistemske biblioteke C prevodioca. Datoteka stdio.h sadrži podatke neophodne za pravilno funkcionisanje ulazno-izlaznih funkcija pa se mora uključiti u program odgovarajućom direktivom.

Ukoliko želimo da sa tastature unesemo neki ceo broj, neophodno je da deklarišemo promenljivu koja će biti korišćena u programu:

```
int a;
```

i da zatim pozovemo funkciju scanf koja će imati sledeći oblik.

```
scanf("%d",&a) ;
```

Oznaka %d predstavlja format za unos celobrojnih promenljivih a &a predstavlja adresu promenljive gde želimo da upišemo novu vrednost koja se unosi.

Za sada nećemo ulaziti u detaljnije objašnjenje zašto se baš koristi adresa promenljive a, jer se to obrađuje u odeljku za pokazivače. Ukoliko želimo da ispišemo neki rezultat na ekranu koristimo funkciju printf. Opšti oblik funkcije printf je:

```
printf(format, promenljiva1, … promenljivaN);
```

Prvi argument funkcije printf je format koji se sastoji iz niza znakova u kome se zadaju konverzije koje treba da se izvršavaju u toku ispisivanja podataka. To znači da kompajler ispisuje na ekranu sve što smo zadali u format do pojave procenta (%) posle čega se čita odgovarajući znak i onda se ispisuju promenljive onim redom kako je u formatu zadato. Recimo, izvršili smo neke operacije nad promenljivom a i želimo da ispišemo vrednost te promenljive na ekran. U tom slučaju koristimo sledeći oblik funkcije printf:

```
printf("Vrednost promenljive a je:  %d", a);
```

Kao što vidimo opet, koristimo %d za format ispisa celobrojne promenljive.

OSNOVNI SIMBOLI KONVERZIJE PRI RADU SA PRINTF I SCANF

Svaki tip podatka ima sopstveni simbol konverzije koji se koristi pri ulazno izlaznim operacijama u C programskom jeziku

U sledećoj tabeli (slika 2) su opisani simboli konverzije koji se koriste kod ulazno/izlaznih funkcija

Simbol	Tip Podatka	Osobine izlazne informacije
c	char	Jedan znak
d	int	Ceo dekadni broj
u	int	Ceo dekadni broj bez znaka
o	int	Ceo oktalni broj bez znaka
x,X	int	Ceo heksadekadni broj bez znaka
s	string	String – niz karaktera
f	float,double	Dekadni zapis realnog broja
e,E	float,double	Eksponencijalni zapis
g,G	float,double	Kraci zapis, izmedju %f i %e

Slika-2 Simboli konverzije različitih tipova podatka koji se koriste kod ulazno izlaznih funkcija

U nastavku su dati neki od primera korišćenja simbola konverzije:

%d - prikazati kao dekadni ceo broj

%6d - prikazati kao dekadni ceo broj u polju od bar 6 znakova

%f - prikazati kao broj sa pokretnim zarezom

%6f - prikazati kao broj sa pokretnim zarezom u polju od bar 6 znakova

%.2f - prikazati kao broj sa pokretnim zarezom sa dve decimalne cifre

%6.2f - prikazati kao broj sa pokretnim zarezom u polju od bar 6 znakova sa dve decimalne cifre.

OSNOVNI PRIMER ULAZNO/IZLAZNIH FUNKCIJA

Funkcija scanf() očekuje da ste uneli podatke u istom formatu koji je definisan korišćenjem odgovarajućih oznaka (npr %c ili %d, itd)

U nastavku je dat osnovni primer korišćenja ulazno/izlaznih funkcija scanf i printf:

```
#include <stdio.h>
int main( )
{
   char str;
   int i;
   printf( "Enter a value :");
   scanf("%c %d", &str, &i);
   printf( "\nYou entered: %c %d ", str, i);
   return 0;
}
```

Kada se prethodno napisani kod prevede i izvrši, on će sačekati korisnika da unese neki tekst. Nakon unosa teksta i pritiska na Enter, program će izvršiti učitavanje teksta koji ste uneli i u konzoli će biti ispisane sledeće linije:

```
Enter a value: a 7
You entered: a 7
```

Ovde se naravno podrazumeva da funkcija scanf() očekuje da ste uneli podatke u istom formatu koji je definisan korišćenjem oznaka %c i %d.

I/O FUNKCIJE ZA RAD SA KARAKTERIMA GETCHAR() I PUTCHAR()

Funkcija getchar učitava prvi sledeći dostupan karakter, dok funkcija putchar konvertuje ASCII kod u karakter i taj karakter štampa na ekran

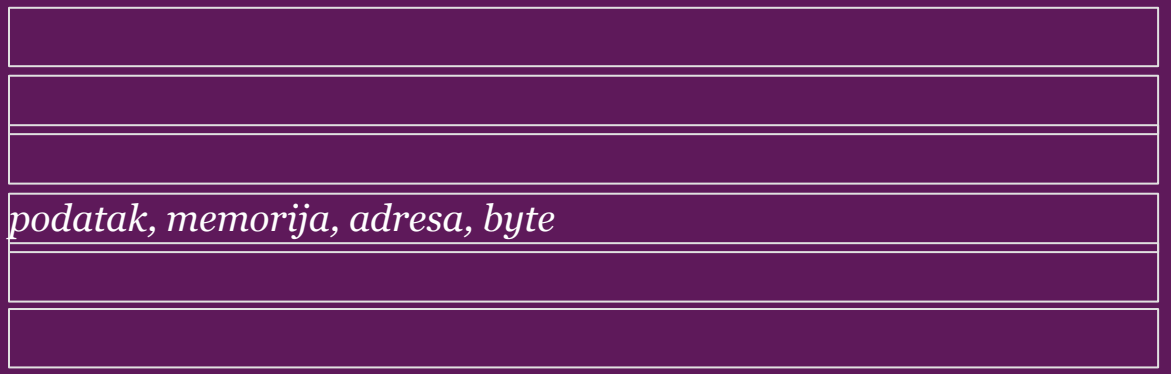
- Funcija int getchar(void) učitava sledeći dostupan karakter sa ekrana i vraća rezultat koji je tipa int (konvertuje karakter u ASCII kod). Ova funkcija čita samo jedan karakter, ali je moguće koristiti ovu funkciju u petlji ukoliko je neophodno učitati niz karaktera sa ekrana.
- Funkcija int putchar (int c) konvertuje celobrojni ASCII kod u karakter i taj karakter štampa na ekran. U nastavku je dat primer korišćenja funkcija getchar i putchar:

```
#include <stdio.h>
int main( )
{
   int c;

   printf( "Enter a value :");
   c = getchar( );

   printf( "\nYou entered: ");
   putchar( c );

   return 0;
}
```

Nakon kompajliranja i pokretanja programa, program čeka na korisnika da unese neki tekst, a nakon unosa teksa i pritiska na Enter, učitava se samo prvi karakter unetog teksta, pa se na ekranu štampa sledeći rezulatat (za uneti tekst: this is test):

```
Enter a value : this is test
You entered: t
```

Tipovi podataka



- *Definisanje novih tipova podataka*

06

PREGLED TIPOVA PODATAKA U C-U

Tipovi podataka u C-u mogu biti osnovni i složeni. Osnovni tipovi predstavljaju bazu za građenje složenih tipova (nizova, struktura, klasa, itd…)

Tipovi podataka u C-u mogu biti osnovni i složeni. Osnovni tipovi podataka su: celobrojni, realni ili u pokretnom zarezu, znakovni, nabrojivi i prazan. Ovi tipovi predstavljaju osnovu za građenje složenih tipova (nizova, struktura, klasa, itd) i ne mogu se razlagati na elementarnije tipove. Tipovi podataka u C jeziku su dati sledećom tabelom:

Tip	Memorija	Opseg Vrednosti
char	1	-128 ... +127
int	2 ili 4	-32768 ... +32768 ili -2,147,483,648 to 2,147,483,647
float	4	3.4E-38... 3.4E+38
double	8	1.7E-308...1.7E+308
long double	10	3.4E-4932 ... 1.1E+4932
void	-	-

Slika-1 Osnovni primitivni tipovi podataka

Osnovna lista tipova može se proširiti uvođenjem sledećih modifikatora za tip **int** i **char**.

Modifikator	Tip	Memorija	Opseg Vrednosti
short	int	2	-32768 ... +32768
long	int	4	-2,147,483,648 ... +2,147,483,647
unsigned	int	2	0 ... 65535
unsigned	short		
unsigned	long int	4	0 ... 4,294,967,295
unsigned	char	1	0 ... 255

Slika-2 Modifikatori za proširivanje liste tipova podataka

VELIČINA TIPA PODATKA

Da bi smo dobili informaciju o veličini memorije koju zauzima neki tip podataka možemo da koristimo funkciju sizeof()

Veličina promenljive može da se razlikuje od vrednosti iz tabele, što zavisi od kompajlera i tipa računara koji se koriste. Da bi smo dobili informaciju o veličini memorije koju zauzima neki tip podataka možemo da koristimo funkciju sizeof. Kao rezultat, funkcija sizeof(type) vraća vrednost memorije u bajtovima koju zauzima podataka označen sa type. U nastavku je dat jedan primer korišćenja funkcije sizeof() za određivanje veličine momorije koju zauzima tip int:

```
#include <stdio.h>
#include <limits.h>
int main()
{
   printf("Storage size for int : %d \n", sizeof(int));
   return 0;
}
```

Analiziraćemo još jedan primer, u kome koristimo header fajl float.h. U ovom fajlu zaglavlja se definišu makroi koji omogućavaju korišćenje vrednosti u vezi binarne reprezentacije realnih brojeva u vašim programima. Naredni primer će kao rezultat oštampati veličinu koju zauzima tip float kao i opseg vrednosti u kojima se tip podatka float kreće:

```
#include <stdio.h>
#include <float.h>
int main()
{
   printf("Storage size for float : %d \n", sizeof(float));
   printf("Minimum float positive value: %E\n", FLT_MIN );
   printf("Maximum float positive value: %E\n", FLT_MAX );
   printf("Precision value: %d\n", FLT_DIG );
   return 0;
}
```

PODACI I MEMORIJA

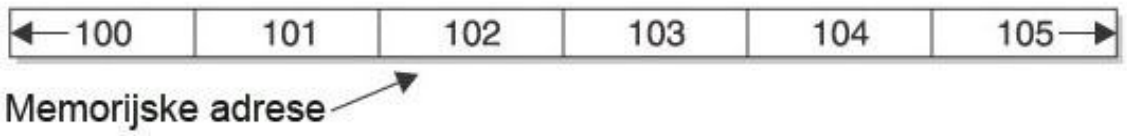
Podaci u računaru su smešteni u nekoj od tri memorijske lokacije: CPU, RAM ili rezervna memorija

Podaci sadrže razne informacije i podaci su ono sa čime program operiše. Neki od mogućih skupova podataka su npr. lista studenata fakulteta i podaci o tim studentima, imena i adrese i telefoni itd. Program operiše sa podacima na razne način, npr. može da pretraži podatke da bi našao informaciju o određenom studentu ili o određenoj grupi studenata. I programske instrukcije i podaci moraju biti smešteni u kompjuterskoj memoriji. Međutim ima raznih tipova kompjuterske memorije pa se postavlja pitanje gde i kako se instrukcije i podaci memorišu u kompjuterskoj memoriji.

Postoje tri memorijske lokacije u kompjuteru:

- Centralna procesorska jedinica, CPU
- Random access memory, RAM
- Rezervna tj. „trajna“ memorija, persistent storage (hard disk, CD-ROM, itd...)

Kada govorimo o C-u, najviše nas interesuje RAM memorija, a posle toga trajna memorija, dok je CPU memorija od interesa više za druge neke jezike kao npr. assembly jezik. Lokacije u memoriji se predstavljaju preko odgovarajuće adrese. Ove adrese se obično pišu koristeći heksadecimalne brojeve (osnova ovih brojeva je 16), npr. 0x8fc1. Međutim, nezavisno od načina pisanja tih brojeva, memorijske adrese se formiraju na isti način kao i ulične adrese, broj za brojem u nizu, kao što je dole ilustrovano:

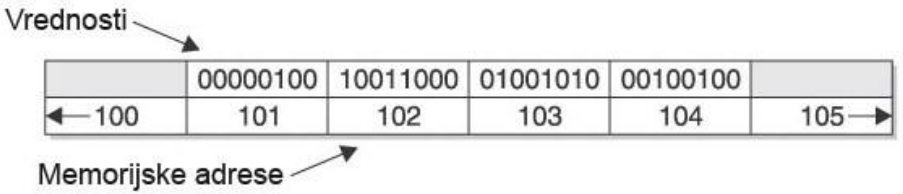


Slika-3 Sekvenca memorijskih adresa

ADRESE U MEMORIJI

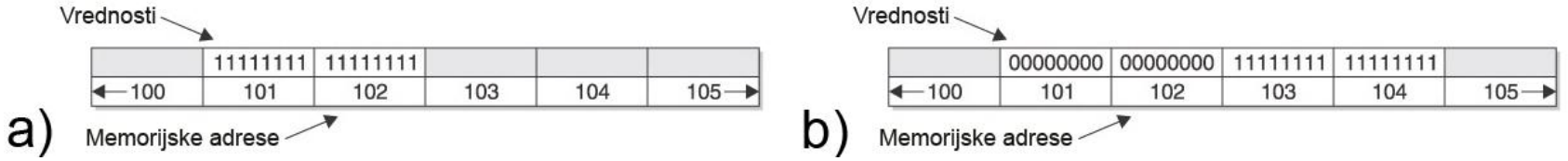
Adrese u memoriji su izražene korišćenjem heksadecimalnih brojeva. U svakoj pojedinačnoj adresi može se smestiti jedan „bajt“ informacija

Heksadecimalni brojevi koriste osnovu 16, što znači da je svaka pojedinačna cifra između 0 i 15, a ne između 0 i 9, pri čemu se 10 piše kao „a“, 11 kao „b“, itd. Pošto su memorijske adrese veliki brojevi koristi se heksadecimalno izražavanje tih brojeva, pa se npr. broj 1000000 (osnova 10) piše kao heksadecimalni f4240. U svakoj pojedinačnoj adresi može se smestiti jedan „bajt“ , *byte*, informacija, a ovaj jedan bajt se sastoji od 8 bitova (*bits*), što je ustvari niz od 8 ćelija, gde svaka ćelija ima vrednost 0 ili 1, dakle 8 nula ili jedinica na tih 8 ćelija (slika 2).



Slika-4 Sekvenca memorijskih adresa, zajedno sa vrednošću na adresi (u byte-ovim)

U gornjoj tabeli vidimo da veličina celih brojeva u zavisnosti od tipa varira od 2 do 4 bajta. Npr. broj 65365 u binarnoj formi se piše kao 1111111111111111, 16 jedinica u nizu, što zahteva 2 bajta memorije. A int tip zahteva 4 bajta memorije. U nastavku je dat primer kako je broj 65365, koji je tipa unsigned int, smešten u odgovarajuću memorijsku lokaciju.



Slika-5 a) Smeštanje broja 65365 u memorijsku lokaciju, gde je broj tipa unsigned short b) Smeštanje broja 65365 u memorijsku lokaciju, gde je broj tipa unsigned int

Definisanje novih tipova podataka

tip podatka, typedef, enum

- *Definisanje novih tipova korišćenjem ključne reči typedef*
- *Nabrojivi tip - enum*
- *Upotreba nabrojivog tipa u programu*

06

DEFINISANJE NOVIH TIPOVA KORIŠĆENJEM KLJUČNE REČI TYPEDEF

U cilju definisanja novih tipova koristi se naredba typedef za kojom slede naziv osnovnog a zatim naziv novodefinisanog tipa podatka. Definisanje novih tipova ima primenu kod slogova (struct)

Da bi se u programu uveli tipovi podataka koji nisu standardni deo C/C++ jezika, koristi se naredba typedef, čiji je opšti oblik:

```
typedef ime_tipa1 ime_tipa2;
```

U najjednostavnijem slučaju ova naredba se može iskoristiti za kreiranje sopstvenih tipova koji su istovetni kao neki od osnovnih tipova podataka, pa se novodefinisani tip može ravnopravno koristiti u programu. Definisanje novog tipa podatka i njegovo korišćenje u C-u može biti ostvareno na sledeći način:

```
typedef int Ceo_broj;
Ceo_broj a,b,x,y=0;
/* prethodne programske linije imaju isto dejstvo kao int a,b,x,y=0; */
```

pri čemu se definiše se novi tip Ceo_broj koji je ekvivalentan standardnom tipu int. U nastavku je dat još jedan primer definisanja novog tipa podatka:

```
#include <stdio.h>
#include <limits.h>
int main()
{
   typedef long long int LLI;
   printf("Storage size for long long int data " \
          "type  : %ld \n", sizeof(LLI));
   return 0;
}
```

Rezultat prethodnog programa biće: Storage size for long long int data type : 8

O definisanju novih tipova će biti više reči u lekciji o strukturama (struct).

NABROJIVI TIP - ENUM

Korišćenjem nabrojivih tipova poboljšava se čitljivost programa i njegovo lakše razumevanje. Nabrojivi tip se definiše kao uređen skup nabrojanih imena, koja predstavljaju konstante tipa

Korišćenjem nabrojivih tipova poboljšava se čitljivost programa i njegovo lakše razumevanje. Nabrojivi tip se definiše kao uređen skup nabrojanih imena, koja predstavljaju konstante tipa. Rezervisana reč koja se koristi da se definiše nabrojivi tip je enum, a sintaksa ima sledeći oblik:

```
enum type_name{ value1, value2,...,valueN };
```

U prethodnom primeru, type_name predstavlja ime nabrojivog tipa, dok value1, value2,...., valueN predstavlja skup vrednosti (ili opseg vrednosti) nabrojivog tipa type_name. Ukoliko se eksplicitno ne navede drugačije value1 će inicijalno biti jednako 0, value2 će biti 1 i tako redom, Naravno programer uvek može da izvrši dodelu vrednosti na način koji će njemu najviše odgovarati na osnovu specifikacije proglema koji rešava. U nastavku je dat jedan prost primer:

```
enum suit
{
   club=0,
   diamonds=10,
   hearts=20,
   spades=3
};
```

Pošto u programskom jeziku C ne postoji logički tip (bool, boolean) mi ga možemo definisati korišćenjem nabrojivog tipa na sledeći način:

```
enum boolean {
    false,
    true
};
enum boolean check;
```

U prethodnom primeru smo definisali novi nabrojivi tip boolean, i deklarisali smo promenljivu check koja je tipa enum boolean.

UPOTREBA NABROJIVOG TIPA U PROGRAMU

Nabrojivi tip (enum) može biti korišćen za dane u nedelji, mesece u godini, boje, strane sveta, i u svim drugim primerima gde se zna uredjen skup mogućih vrednosti promenljive

Analizirajmo još jedan primer. Pretpostavimo da treba da napišemo program u kome se izvršavaju određene akcije koje se odnose na svaki dan u nedelji. Jedan od načina je da dane nedelje predstavimo brojevima od 0 do 7. Korišćenjem nabrojivog tipa to možemo uraditi na sledeći način:

```
enum dani{ ponedeljak, utorak, sreda, cetvrtak, petak, subota, nedelja};
```

Korišćenjem oznake nabrojivog tipa dani može se definisati promenljiva dan na sledeći način:

```
enum  dani dan;
```

U nastavku je dat detaljan primer korišćenje nabrojivog tipa.

```
#include <stdio.h>
enum dani{ ponedeljak, utorak, sreda, cetvrtak, petak, subota, nedelja};
int main()
{
    enum dani danas;
    danas=sreda;
    printf("%d day", danas +1);
    return 0;
}
```

Promenljive i konstante

Promenljive, opseg vrednosti, konstante, konverzija

- *Konverzija promenljivih*

07

UVODNA RAZMATRANJA

Promenljiva je objekt jezika koji ima ime i kome se može menjati sadržaj u toku izvršavanja programa, za razliku od konstanti koje dobijaju vrednost pre izvršavanja i ostaju nepromenjene

Promenljive i konstante su osnovni oblici podataka sa kojima se operiše u programu. Računar može obavljati razne operacije (aritmetičke, sortiranja, itd..) ali u svim tim procesima on mora raspolagati sa podacima: brojevima, simbolima, odnosno objektima koji sadrže određenu informaciju neophodnu za pravilno izvršavanje programa kojim se rešava neki problem. Promenljiva je objekat jezika koji ima ime i kome se može menjati sadržaj u toku izvršavanja programa. Imena promenljivih su sastavljena od slova i cifara. Prvi znak mora biti slovo ili znak podvučeno. U jeziku C postoji razlika između malih i velikih slova.

Objekti jezika koji dobijaju vrednost pre nego što počne izvršavanje programa i u toku programa se ne mogu menjati se nazivaju konstante. Postoje dva načina da se definišu konstante u C/C++:

- Korišćenjem direktive pretprocesora #define,
- Korišćenjem rezervisane reči const (u Java-i je final).

DEFINICIJA I DEKLARACIJA PROMENLJIVIH

Definicija promenljive je ustvari poruka kompajleru da u memoriji odvoji određeni prostor na kome će biti čuvan podatak vezan za promenljivu

Definicija promenljive je ustvari poruka kompajleru da na određenoj memorijskoj lokaciji odvoji tačno određeni prostor na kome će biti čuvan podatak vezan za promenljivu. Definicija promenljive specificira tip podatka, i može da sadrži listu više od jedne promenljive, opšti oblik deklaracije je sledeći:

```
type variable_list;
```

dok su u nastavku dati primeri deklaracije promenljivih različitih tipova:

```
int    i, j, k;
char   c, ch;
float  f, salary;
double d;
```

Linija int i, j, k; predstavlja deklaraciju i definiciju promenljivih i, j, k; a istovremeno se šalje poruka kompajleru da u memoriji odvoji prostor za promenljive i, j, k koje su celobrojnog tipa (int). Promenljive mogu biti i inicijalizovane prilikom deklaracije, pri čemu se inicijalizacija sastoji iz operatora = i konstantnog izraza, pa je opšti oblik incijalizacije pri deklaraciji:

```
type variable_name = value;
```

Neki od primera inicijalizacije pri deklaraciji su dati u nastavku:

```
int d = 3, f = 5;           // definicija i inicijalizacija d i f.
byte z = 22;                // definicija i inicijalizacija z.
char x = 'x';               // the variable x has the value 'x'.
```

OPSEG VREDNOSTI PROMENLJIVE

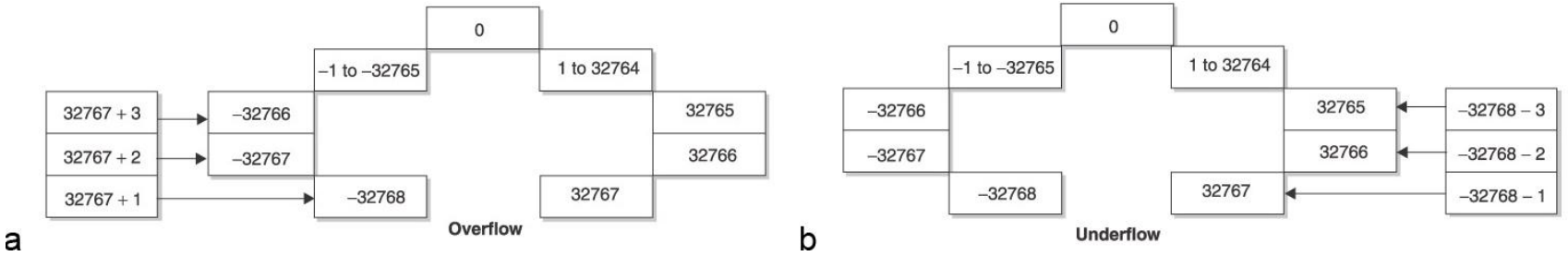
Svaki tip podatka ima dozvoljen opseg vrednosti i ukoliko se probije taj opseg program počinje da greši. Probijanje gornjeg limita zove se „numeric overflow“, a donjeg limita „numeric underflow“

Svaki kompjuter ima limit tj. opseg za celobrojne veličine. Kod realnih kompjutera taj opseg je relativno veliki ali ipak ograničen. Npr. opseg za celobrojne promenljive je od -2147483648, do 2147483647, znači približno plus minus 2 milijarde, dok je za decimalni broj, i decimalni broj sa duplom tačnošću, limit mnogo veći, npr. 10^{38}. Kod većine kompjutera, tzv. duga celobrojna promenljiva omogućuje 4,294,967,296 različitih vrednosti. Ipak je to konačna vrednost. Ako se pređe taj limit kompjuter počinje da greši, i ta greška se zove: numeric overflow (slika 1a). Izlazak brojne vrednosti iz opsega tipa podatka možemo videti na sledećem primeru:

```
short testScore = 32768;
printf("Your test score is %d\n", testScore);
```

Rezultat programa je: “Your test score is –32768.” a ne 32768 kao što smo uneli. Na sličan način se može dogoditi da se probije donja granica, i taj proboj se obično naziva underflow (slika 1b). Ako posmatramo naredni primer primetićemo da će vrednost -32768 probiti donji opseg pa će na ekranu biti oštampan rezultat: “Your test score is –32767”.

```
short testScore = -32769;
printf("Your test score is %d\n", testScore);
```



Slika-1 a) Overflow. Izlazak celobrojne vrednosti van gornje granice opsega celobrojnog tipa short b) Underflow. Izlazak celobrojne vrednosti van donje granice opsega celobrojnog tipa short

DEFINISANJE KONSTANTI KORIŠĆENJEM PRETPROCESORSKE NAREDBE

Definisanje konstanti počinje pretprocesorskom naredbom #define za kojoj slede simboličko ime i vrednost konstante

Konstante u C-u mogu biti definisane pomoću direktiva pretpocesora jezika C. Definicija konstante počinje direktivom pretprocesoru:

```
#define identifier value
```

za kojom slede simboličko ime i vrednost konstante. Vrednost konstante može biti ceo ili realan broj, kao i tekstualni podatak (niz karaktera, tj, string). U nastavku je dat detaljan primer korišćenja konstanti definisanih pomoću #define.

```
#include <stdio.h>

#define LENGTH 10
#define WIDTH  5
#define NEWLINE '\n'

int main()
{

   int area;

   area = LENGTH * WIDTH;
   printf("value of area : %d", area);
   printf("%c", NEWLINE);

   return 0;
}
```

Nakon kompajliranja i povezivanja prethodnog programa dobiće se sledeći rezultat: 50

KLJUČNA REČ CONST

Definisanje konstanti je moguće izvršiti korišćenjem ključne reči const. Najčešća programerska praksa je da se nazivi konstanti pišu velikim slovima

Osim pretprocesorske direktive, moguće je koristiti i prefix const u cilju deklarisanja konstanti. Opšti oblik korišćenja ključne reči const je dat u nastavku:

```
const type variable = value;
```

kao i osnovni primer korišćenja ključne reči const:

```
#include <stdio.h>
int main()
{
   const int  LENGTH = 10;
   const int  WIDTH  = 5;
   const char NEWLINE = '\n';
   int area;

   area = LENGTH * WIDTH;
   printf("value of area : %d", area);
   printf("%c", NEWLINE);

   return 0;
}
```

Nakon kompajliranja i izvršenja programa koji sadrži prethodne linije koda, na ekranu ce biti prikazan sledeći rezultat:

value of area : 50

Treba napomenuti da je najčešća programerska praksa da se konstante pišu velikim slovima.

Konverzija promenljivih

konverzija, cast-ovanje

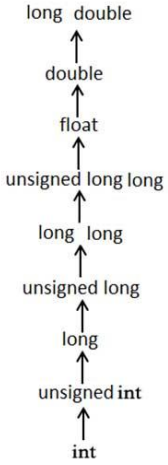
- *Implicitna (automatska) konverzija*
- *Eksplicitna konverzija (cast-ovanje)*

07

IMPLICITNA (AUTOMATSKA) KONVERZIJA

Ako se u aritmetičkom izrazu pojave operandi različitih tipova tada se jedan konvertuje tako da odgovara operandu drugog tipa

Ako se u nekom aritmetičkom izrazu pojave operandi različitih tipova tada se jedan konvertuje tako da odgovara operandu drugog tipa. Osnovni tipovi podataka imaju sledeći hijerarhijski poredak (konverzija donjeg u tip koji je iznad njega):



Slika-1 Hijerarhijski poredak tipova podataka

Konverzija se realizuje i pri operaciji dodeljivanja. Bez obzira kakav je tip izraza desno od znaka dodele, on se pretvara u tip promenljive levo od znaka dodele. U slučaju pretvaranja realnog u celobrojni podatak uvek se vrši odsecanje razlomljenog dela.

Posmatrajmo sledeći primer gde se vrši sabiranje dva različita tipa podatka (int i char) i rezultat smešta u promenljivu tipa int:

```
#include <stdio.h>

void main()
{
   int  i = 17;
   char c = 'c'; /* ascii value is 99 */
   int sum;

   sum = i + c;
   printf("Value of sum : %d\n", sum );

}
```

Nakon izvršenja prethodnog koda na ekranu će biti prikazan sledeći rezultat:

```
Value of sum : 116
```

U prethodnom primeru, promenljiva sum će imati vrednost 116 jer kompajler implicitno karakter 'C' pretvara u njegov ASCII kod, pa tek onda taj kod sabira sa i i na kraju rezultat (116) dodeljuje promenljivoj sum.

EKSPLICITNA KONVERZIJA (CAST-OVANJE)

Eksplicitna konverzija (cast-ovanje) je jedan od načina da se izvrši konverzija vrednosti promenljive iz jednog tipa podatka u drugi

Osim automatske konverzije tipova može se izvršiti i eksplicitna konverzija odnosno cast-ovanje. Eksplicitna konverzija (cast-ovanje) je jedan od načina da se izvrši konverzija vrednosti promenljive iz jednog tipa podatka u drugi. Opšti oblik operacije kastovanja (cast) je prikazan na sledeći način:

```
(type_name) expression
```

Posmatrajmo još jedan primer gde je operacija konverzije izvršena pri deljenju dva cela broja, pri čemu se prvi broj konvertuje u double pa tek onda se deli sa drugim brojem:

```
#include <stdio.h>
void main()
{
   int sum = 17, count = 5;
   double mean;

   mean = (double) sum / count;
   printf("Value of mean : %f\n", mean );
}
```

Nakon izvršavanja prethodnog programa dobija se sledeći rezultat:

```
Value of mean : 3.400000
```

Ovde treba naglasiti da operator konverzije (cast operator) ima prednost u odnosu na deljenje, što znači da je vrednost promenljive sum prvo konvertovana u tip double pa je tek onda podeljena sa vrednošću promenljive count, što je dovelo da se kao rezultat dobije realan broj tipa double. Kao što se može videti iz prethodnih primera konverzija može biti eksplicitna (korišćenje cast operatora) ili automatska (konverzija od strane kompajlera), ali je ipak poželjno da se ipak koristi cast operator gde god je konverzija neophodna.

Tipovi operatora

operator, tipovi, prioriter, asocijativnost

- *Uvod u pokazivače*

08

PODELA TIPOVA OPERATORA

Operator je simbol koji koji ukazuje kompajleru da treba da izvrši neku matematičku ili logičku operaciju. Operacije se vrše nad objektima koji se nazivaju operandi

Operatori jezika C/C++ sa mogu svrstati u nekoliko funkcionalnih grupa i one su prikazane u ovom odeljku. Pošto je najveći deo operatora isti kao i u ostalim programskim jezicima (npr. Java), o njima neće biti previše reči i detalja. Biće detaljno opisani oni operatori koji su karakteristični za programske jezike C/C++.

C/C++ jezici sadrže sledeće tipove operatora:

- Aritmetički operatori
- Operatori poređenja
- Logički operatori
- Operatori nad bitovim
- Operatori dodele vrednosti
- Operatori specijalne namene (mešoviti operatori)

U okviru operatora specijalne namene nalaze se dva operatora za rad sa pokazivačima tako će se u ovoj lekciji studenti upoznati i sa osnovima deklaracije i upotrebe pokazivača.

ARITMETIČKI OPERATORI I OPERATORI DODELE VREDNOSTI

Aritmetički operatori su binarni operatori jer uvek rade sa dve vrednosti. Operatori dodele vrednosti se koriste sa ciljem da se promenljivoj dodeli neka vrednosti

- **Aritmetički operatori**

-	negativni predznak ili oduzimanje
+	sabiranje
*	množenje
/	deljenje
%	moduo (celobrojni ostatak pri deljenju)

Primena: rezultat = a operator b

Slika-1 Aritmetički operatori

- **Operatori dodele vrednosti:**

=					elementarna dodela vrednosti
-=	+=	*=	/=	%=	aritemetička operacija sa dodelom vrednosti
<<=	>>=	&=	|=	^=	operacija nad bitovima sa dodelom vrednosti

Primena: a = b

Primena: a operator= b (ekvivalentno je: a = a operator b)

Slika-2 Operatori dodele vrednosti i način primene

Operator	*Ime*	*Primer*	*Isto kao:*
+=	sabiranje i dodela	**i += 8**	**i = i + 8**
-=	oduzimanje i dodela	**i -= 8**	**i = i – 8**
***=**	množenje i dodela	**i *= 8**	**i = i * 8**
/=	deljenje i dodela	**i /= 8**	**i = i / 8**
%=	ostatak pri deljenju i dodela	**i %= 8**	**i = i % 8**

Slika-3 Primena dodele vrednosti i opis svake od operacija

OPERATORI POREĐENJA I LOGIČKI OPERATORI

Operatori poređenja su takođe binarni operatori jer uvek vrše poređenje dva operanada. Logički operatori omogućavaju programu da donese odluku zasnovanu na više mogućih ishoda

- **Operatori poređenja**

<	manje
<=	manje ili jednako
>	veće
>=	veće ili jednako
==	jednako
!=	nije jednako

Primena: rezultat = a operator b

Slika-4 Operatori poređenja i način primene

U nastavku je dat prost primer korišćenja operatora poređenja:

```
int rezultat;
float  a=5.2, b=11.4;
rezultat = a  >=  b;                /* rezultat=0 */
rezultat = a  <  b;                 /* rezultat=1 */
rezultat =  a  !=  b;               /* rezultat=1 */
```

- **Logički operatori**

&&	logičko "i"
||	logičko "ili"
!	logicka negacija

Primena: rezultat = log_izraz1 operator log_izraz2

Slika-5 Logički operatori i način upotrebe

U nastavku je dat prost primer korišćenja logičkih operatora:

```
int rezultat;
float  a=5.2;
char b='0';
rezultat = (a == 5. || a > 5.1) && (b != '\n');
/* rezultat = ( 0 || 1) && (1) = (1) && (1) = 1 */
```

OPERATORI UVEĆANJA I UMANJENJA, OPERATORI NAD BITOVIMA I OPERATORI SPECIJALNE NAMENE

Operatori uvećanja/umanjenja omogućavaju uvećanje/smanjenje vrednosti promenljive za jedan.

Operatori nad bitovima služe za promenu individualnog bita operanda

- **Operatori umanjenja i uvećanja :**

--	unarno dekrementiranje (umanjenje vrednosti za 1)
++	unarno inkrementiranje (uvećanje vrednosti za 1)

Slika-6 Operatori umanjenja i uvećanja

Primer korišćenja operatora umanjenja i uvećanja:

```
int rezultat, a=3, b=3;
rezultat = a * --b; /* rezultat = 6 */
/*     umesto prethodne linije mogu se pisati
sledeće dve linije:
 b = b - 1;  rezultat =  a * b;   */
b++;   /*  b = 3  */
rezultat = a-- * b; /* rezultat = 9 */
/*     umesto prethodne linije mogu se pisati
sledeće dve linije:
 rezultat =  a * b;  a =  a - 1; */
```

- **Operatori nad bitovima:**

<<	pomeranje ulevo
>>	pomeranje udesno
&	logičko "i" za niz bitova
|	logičko "ili" za niz bitova
^	isključivo logičko "ili" za niz bitova

Primena: rezultat = a operator b

~	komplement

Slika-7 Osnovni operatori nad bitovima

- **Operatori specijalne namene**

,	operator comma (zarez)
? :	operatori grananja
(tip)	cast operator (konverzija tipova)
sizeof()	operator sizeof

Slika-8 Neki od operatora specijalne namene

U nastavku je dat set primera za neke od pomenutih operatora:

```
// Upotreba comma operatora
int rezultat, i=1, j;
rezultat = ( i = i+4 , j = --i);/* rezultat = 4 */
// Upotreba operatora grananja:
int maksimum, a=4, b=6;
maksimum = ( a >= b ) ? a : b;/* maksimum = 6 */
// Upotreba operatora konverzije tipova
float a=1.;
double rezultat;
rezultat = ( (double) a )/3. ;
//Upotreba sizeof() operatora
int rezultat;
rezultat = sizeof(short int);/* rezultat = 2 */
rezultat = sizeof(long int);/* rezultat = 4 */
```

PRIORITET I ASOCIJATIVNOST OPERATORA

Prioritet operatora definiše način grupisanja u nekom složenom izrazu

Prioritet operatora definiše način grupisanja u nekom složenom izrazu, pa stoga utiče na način kako će se neki izraz sračunati. Neki operatori imaju veći prioritet u odnosu na ostale, npr. operator množenja ima veći prioritet od operatora sabiranja, itd.

Operator	Asocijativnost
() [] -> . ++ - -	Sa leva na desno
+ - ! ~ ++ - - (tip) * & sizeof	Sa desna na levo
* / %	Sa leva na desno
+ -	Sa leva na desno
<< >>	Sa leva na desno
< <= > >=	Sa leva na desno
== !=	Sa leva na desno
&	Sa leva na desno
^	Sa leva na desno
|	Sa leva na desno
&&	Sa leva na desno
||	Sa leva na desno
?:	Sa desna na levo
= += -= *= /= %=>>= <<= &= ^= |=	Sa desna na levo
,	Sa leva na desno

Slika-9 Prioritet i asocijativnost operatora

Uvod u pokazivače

promenljiva, adresa, pokazivač, adresni operator

- *Indirektni i adresni operator*
- *Osnovi o pokazivačima*
- *Upotreba pokazivača*

08

INDIREKTNI I ADRESNI OPERATOR

Adresni operator & vraća adresu operanda odnosno promenljive uz koju stoji u izrazu.
*Indirektni operator * vraća vrednost promenljive na adresi na koju pokazuje pokazivač*

C obezbeđuje dva operatora za rad sa pokazivačima, i to (a) adresni operator & i (b) indirektni *. Pre svega da kažemo nešto o pokazivačima. Pokazivač je promenljiva koja sadrži adresu druge promenljive. Drugim rečima, pokazivač je promenljiva koja pokazuje na adresu druge promenljive, odnosno pokazuje na drugu promenljivu. Promenljiva na koju pokazuje pokazivač može biti bilo koji tip podatka kao npr. objekti i strukture, pa čak može biti i pokazivač (pokazivač na pokazivač).

*	operator indirektnog (posrednog) pristupa
&	adresa

Primena: *operator a*

Slika-1 Operatori za rad sa pokazivačima

Adresni operator &

Operator & je unarni operator koji kao rezultat vraća memorijsku adresu operanda odnosno promenljive uz koju stoji u izrazu. Na primer, ukoliko je var celobrojna promenljiva, onda &var predstavlja njenu adresu. Operator & se čita kao "adresa od" što znači da će &var biti pročitano kao "adresa promenljive var".

Indirektni operator *

Drugi operator za rad sa pokazivačima je indirektni operator *, i on je komplement operatora &. Operator * je unarni operator i on vraća vrednost promenljive na adresi na koju pokazuje pokazivač.

Kao što smo već spomenuli pokazivač (pointer) je promenljiva čija vrednost predstavlja adresu neke druge promenljive. Kao pri radu sa bilo kojom drugom promenljivom ili konstantom, prvo je neophodno deklarisati pokazivače da bi ih mogli koristiti.

OSNOVI O POKAZIVAČIMA

Vrednost pokazivača je ustvari heksadecimalni broj koji predstavlja memorijsku adresu na kojoj će biti smeštena promenljiva na koju on pokazuje

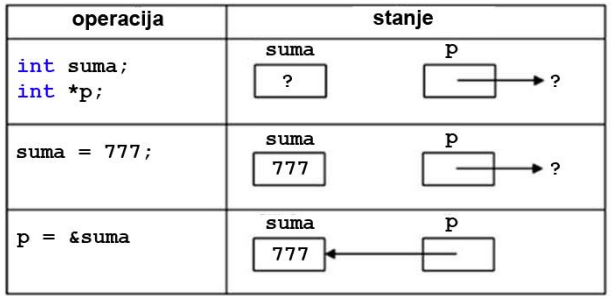
Osnovni oblik deklaracije pokazivača (pokazivačke promenljive) je:

```
type *var-name;
```

Ovde se type odnosi na tip podatka promenljive, i to mora biti validni C/C++ tip podatka (osnovni ili složen), dok je var-name ime pokazivačke promenjive. Pri deklaraciji se koristi znak zvezdica "*" (asterisk) da bi se ukazalo da se radi o pokazivačkoj promenljivoj. U nastavku su dati primeri deklaracija pokazivača:

```
int    *ip;    // pokazivač na integer
double *dp;    // pokazivač na double
float  *fp;    // pokazivač na float
char   *ch     // pokazivač na character
```

Bez obzira da li pokazivač pokazuje na promenljivu tipa int, double, float ili char, vrednost pokazivača je ustvari heksadecimalni broj koji predstavlja memorijsku adresu na kojoj će biti smeštena promenljiva tipa int, double, float ili char, na koju pokazivač "pokazuje". Jedina razlika između pokazivača na različite tipove podatka je ta da će promenljive na koje pokazuju biti različitog tipa. Sledeća slika 2 daje prikaz operacija s pokazivačem na celobrojnu promenljivu.



Slika-2 Primer deklaracije i upotrebe pokazivača

UPOTREBA POKAZIVAČA

Pokazivači mogu da se koriste za bilo koji validni C/C++ primitivni ili složeni tip podatka

U nastavku je dat osnovni primer korišćenja pokazivača:

```
#include <stdio.h>

int main ()
{
   int  var;
   int  *ptr;
   int  val;

   var = 3000;

   // take the address of var
   ptr = &var;

   // take the value available at ptr
   val = *ptr;
   printf("Value of var : %d\n", var);
   printf("Value of ptr : %x\n", ptr);
   printf("Value of val : %d\n", val);

   return 0;
}
```

Nakon izvršavanja prethodnog programa dobija se sledeći rezultat:

```
Value of var :3000
Value of ptr :0xbff64494
Value of val :3000
```

Upotreba razvojnog okruženja Visual Studio 2010

IDE, upotreba rayvojnih okruženja Visual Studio 2010

- *Visual Studio projekat sa jednim fajlom*
- *Korišćenje komandne linije*

09

UVODNA RAZMATRANJA

U nastavku će biti ukratko opisan način pisanja programa korišćenjem integrisanog okruženja Visual C++

Biće opisano integrisano razvojno okruženje (**Integrated Development Environment - IDE**) softverskog paketa Microsoft Visual C++ 2010, i biće prikazane opcije pisanja koda, izmena koda, kompajliranja, linkovanja i izvršavanja C/C++ programa u jednom takvom okruženju.

Jedan prost C/C++ projekat, koji se sastoji iz samo jednog fajla sa izvornim kodom je korišćen kao demonstracioni primer, tako da se studenti trebaju fokusirati na rad u Visual C++ 2010 IDE-u umesto na kod koji u ovom delu kursa može samo da ih zbuni. Radi potpunosti detalja, na kraju odeljka je dat i izvorni kod (source code) primera.

Napomena: Ukoliko studenti žele mogu da koriste druga dva dozvoljena razvojna okruženja za kreiranje C/C++ aplikacija ali njihov opis nećemo razmatrati u okviru ove lekcije. Uputstva za neophodna podešavanja i kreiranje osnovnih projekata se mogu naći na sledećim linkovima:

- za Netbeans:

https://netbeans.org/kb/docs/cnd/quickstart.html

https://netbeans.org/community/releases/80/cpp-setup-instructions.html3

- za CodeBlocks:

http://wiki.codeblocks.org/index.php/Creating_a_new_project

http://www.cplusplus.com/doc/tutorial/introduction/codeblocks/

Visual Studio projekat sa jednim fajlom

Visual Studio, Netbeans C/C++, Code::Blocks, konzola

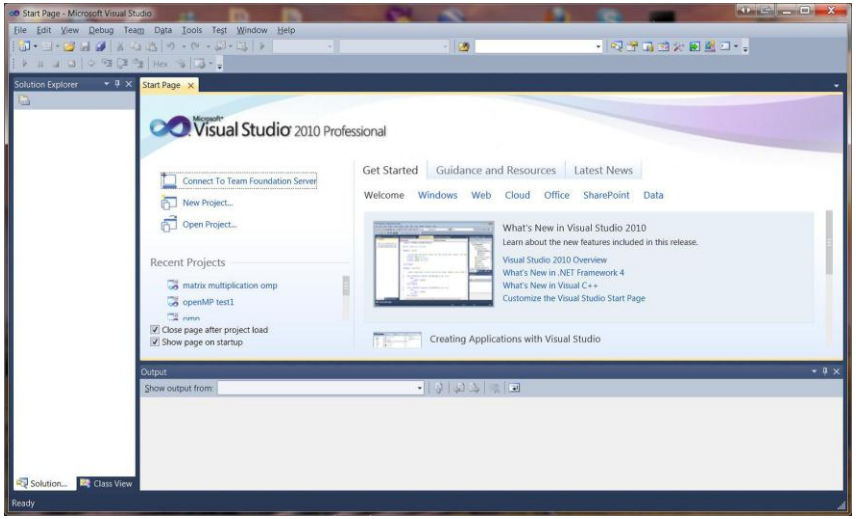
- *Korak 1 – Pokretanje aplikacije*
- *Korak 2 – Kreiranje novog projekta konzolne aplikacije*
- *Korak 3 – Dodavanje izvornih fajlova u projekat*
- *Korak 4 – Pisanje koda u editoru razvojnog okruženja*
- *Korak 5 – Kompajliranje i izvršavanje aplikacije*

09

KORAK 1 – POKRETANJE APLIKACIJE

Korak 1: Pokrenuti MS Visual C/C++ 2010 aplikaciju sa desktop-a ili iz start menija

Korak 1: Pokrenuti MS Visual C/C++ 2010 aplikaciju sa desktop-a ili iz start menija. Otvoriće se glavni prozor aplikacije koji može izgledati kao na sledećoj slici:



Slika-1 Izgled prozora nakon koraka 1

Napomena: Detaljno uputstvo za kreiranje projekta u programskom paketu Visual C++ se može naći na sledećim internet linkovima:

http://cs-cit.wpunj.edu/dotAsset/308705.pdf

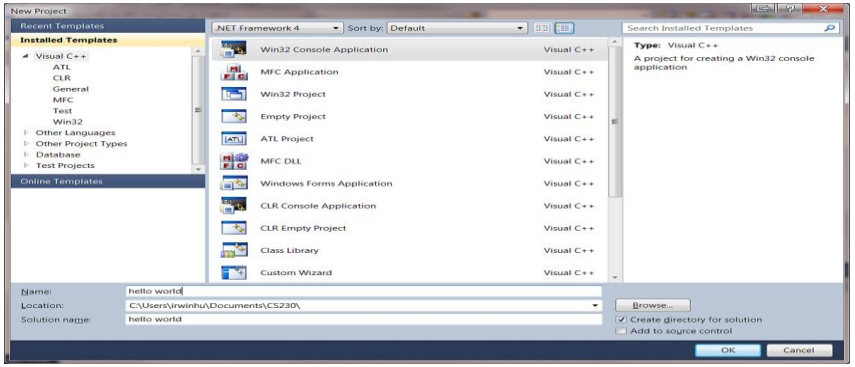
http://www.cs.fsu.edu/~myers/howto/createProgMSVCEX.pdf

http://www.cplusplus.com/doc/tutorial/introduction/visualstudio/

KORAK 2 – KREIRANJE NOVOG PROJEKTA KONZOLNE APLIKACIJE

*Korak 2: Iz menu bar-a, kliknuti na **File-New -Projects…** da bi se otvorio **New Project** dijalog prikazan na sledećoj slici:*

U **New Project** dialogu prikazanom na slici ispod, selektovati **Visual C++** u okviru **Installed Template** okna a zatim **Win32 Console Application** koji se nalazi u središnjem prozoru. Zatim u **Name** box-u treba upisati naziv projekta (npr, *hello world* kao što je priazano), izabrati lokaciju ili direktorijum u kome će biti snimljeni fajlovi novo-kreiranog projekta (npr, C:\Users\irwinhu\Documents\CS230\ kao što je prikazano) klikom na dugme **Browse…**

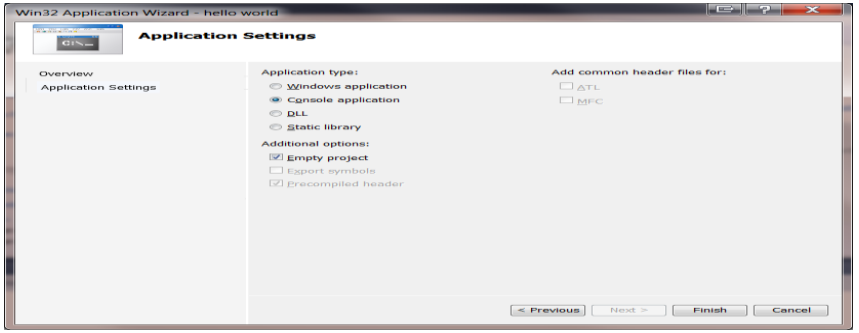


Slika-2 Prozor New Project

Zatim kliknuti na dugme **OK** i otvoriće se dijalog **Win32 Application Wizard – hello world** kao što je prikazano na sledećoj slici:

Slika-3 Dijalog Win32 Aplication Wizard nakon otvaranja

Kliknuti na dugme **Next** da bi se otvorio sledeći dijalog:

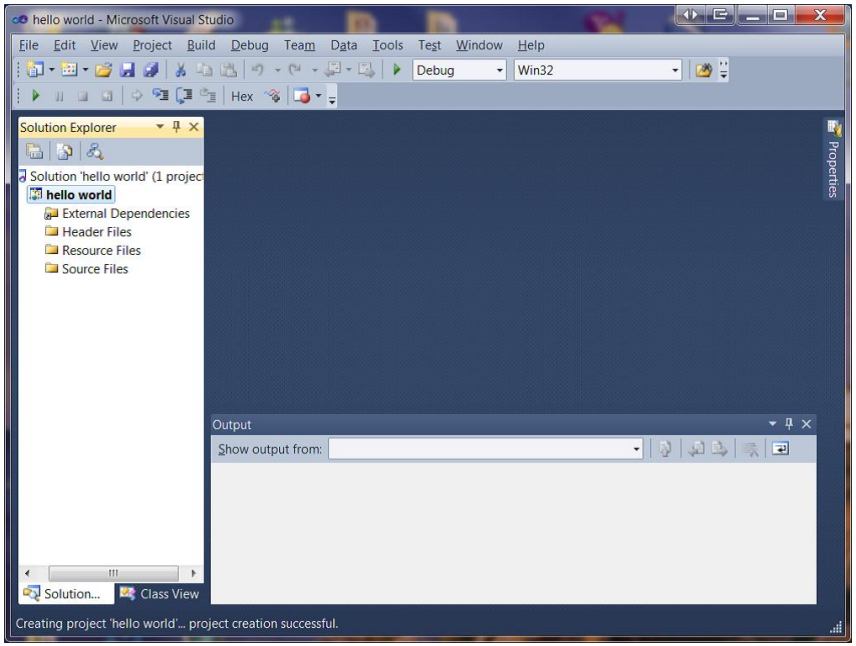


Slika-4 Win32 Application Wizard – izbor opcije Empty project

Čekirati **Empty project box** kao što je prikazano na prethodnoj slici i pritisnuti dugme **Finish** da bi se pristupilo sledećem koraku.

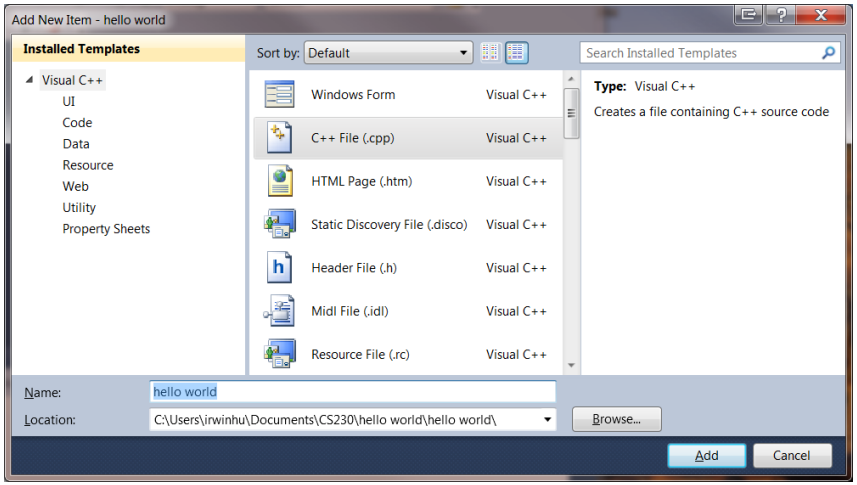
KORAK 3 – DODAVANJE IZVORNIH FAJLOVA U PROJEKAT

Korak 3: Sada će programsko okruženje Visual Studio 2010 aplikacija imati sledeći izgled



Slika-5 Izgled VS 2010 nakon kreiranja praznog projekta

Postaviti kursor miša na direktorijum **Source Files** okviru **Solution Explorer** okna koje se nalazi na levoj strain aplikacije. Pritiskom na desni taster miša otvoriće se popup meni, gde treba kliknuti na **Add** pa onda **New Item**… nakon čega će se otvoriti **Add New Item – *hello world*** dijalog:



Slika-6 Izgled dijaloga Add New Item

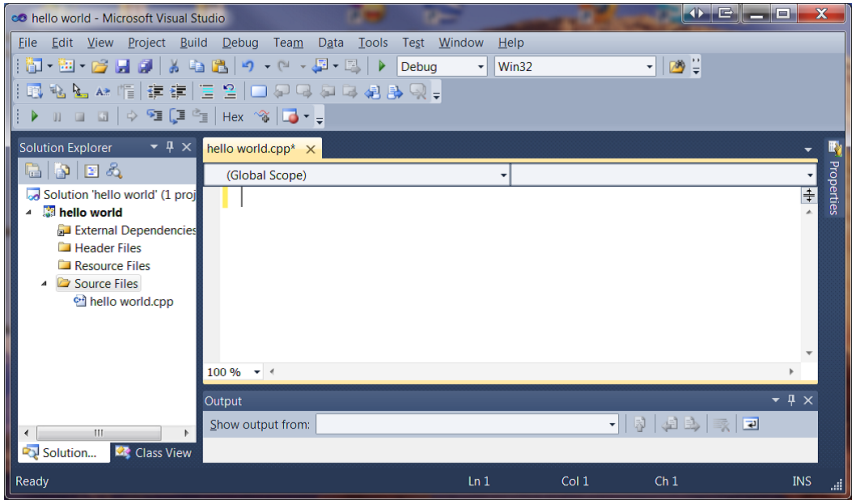
Selektovati **C++ Files (.cpp)** klikom na opciju u središnjem prozoru i izabrati proizvoljno ime fajla (npr, *hello world*, isto ime koje smo koristili za ime projekta). Pritisnuti dugme **Add** da bi ste prešli na sledeći korak.

Napomena: Za rad sa programskim jezikom C treba promeniti ekstenziju fajla *.cpp u *.c.

KORAK 4 – PISANJE KODA U EDITORU RAZVOJNOG OKRUŽENJA

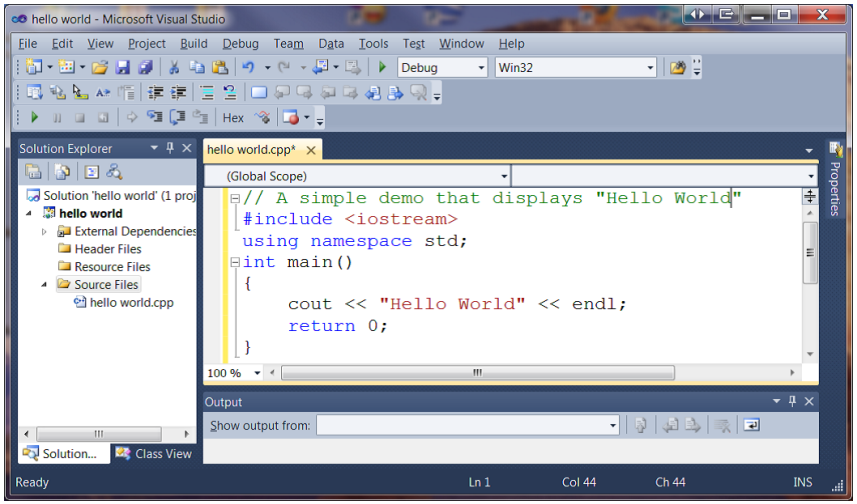
Korak 4: Kostur početne aplikacije

Korak 4: Na ekranu će biti prikazan prozor kao na sledećoj slici. U okviru prozora **Solution Explorer** može se primetiti 1) direktorijum **Source Folder** (proveriti da li je aktivan tab **Solution Explorer Class View** taba) u kome se nalazi *hello world.cpp* fajl koji smo upravo kreirali i 2) prazan tekstualni editor sa naslovom *hello world.cpp* *u kome možete kucati vaš* C/C++ izvorni kod.



Slika-7 Otvaranje editora duplim klikom na ime fajla “hello world”

Nakon unosa koda prozor će imati sledeći izgled:



Slika-8 Izgled tekstualnog editor-a nakon kucanja koda

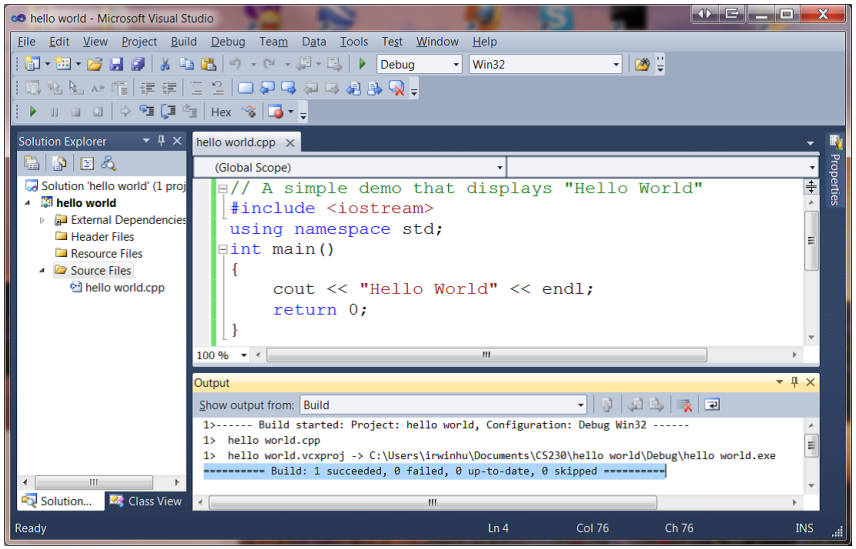
Napomena: U slučaju rada sa projektom koji se sastoji iz više izvornih fajlova, neophodno je u direktorijum **Source Files**, odnosno **Header files** dodati nove izvorne fajlove (*.c) ili fajlove zaglavlja (*.h). Pokušati kompajliranje fajlova add.c i main.c u okviru sekcije 4 – Program sa više fajlova.

KORAK 5 – KOMPAJLIRANJE I IZVRŠAVANJE APLIKACIJE

Korak 5: Pokretanje programa se vrši klikom na Debug iz menija pa onda klikom na Start Without Debugging

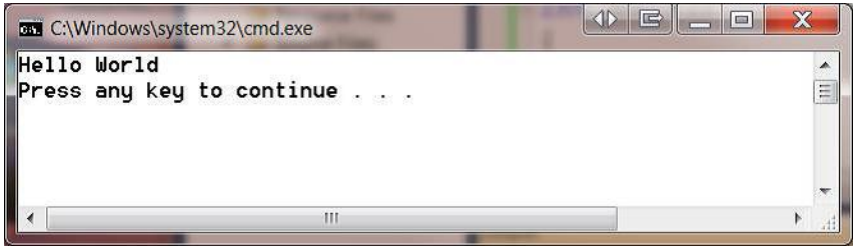
Korak 5: Da bi ste kompajlirali, povezali (linkovali) i izvršili program kliknuti na **Debug** u meni baru, a zatim na **Start Without Debugging** u okviru padajućeg menija. Ukoliko ne postoji greška u vašem kodu biće prikazana sledeća poruka u okviru prozora **Output** kao što je prikazano na slici ispod

=== Build: 1 succeeded, 0 failed, 0 up-to-date, 0 skipped ===



Slika-9 Izveštaj o uspešnom kompajliranju

Takođe, u posebnom prozoru (konzoli - C:\Windows\system32\cmd.exe) biće prikazan izlaz programa:



Slika-10 Izgled konzole nakon izvršavanja programa

U nastavku je dat izvorni koda za prethodni primer: The Hello World Program

Ime fajla: Hello World.cpp

```
#include <iostream>
Using namespace std;
int main()
{
            cout << “Hello World!” << endl; return
0;
}
```

Napomena: Umesto prethodnog C++ koda moguće je zalepiti (paste) bilo koji C program sa predavanja i probati ponovo celokupan postupak do izvršavanja programa.

Korišćenje komandne linije

Komandna linija, cl kompajler

- *Osnovi rada u komandnoj liniji*
- *Kreiranje izvornog C fajla u programu Notepad*
- *Kompajliranje i izvršavanje programa*

09

OSNOVI RADA U KOMANDNOJ LINIJI

Visual Studio obuhvata i C kompajler koji možete koritititi da kreirate bilo šta, od osnovne konzolne aplikacije do Windows Desktop aplikacije

Visual Studio obuhvata i C kompajler koji možete koritititi da kreirate bilo šta, od osnovne konzolne aplikacije do Windows Desktop aplikacije. U nastavku je opisan postupak kreiranja osnovnog C programa korišćenjem tekst editora, a zatim i postupak kompajliranja iz komandne linije. Moguće je koristiti bilo koji vaš C program umesto primera koji je opisan u okviru ovog uputstva.

Prema podrazumevanim podešavanjima, Visual C++ kompajler tretira sve fajlove sa ekstenzijom .c kao fajlove sa izvornim kodom C jezika, i sve fajlove sa ekstenzijom .cpp kao fajlove sa izvornim kodom C++ jezika.

Da bi kreirali C izvorni fajl i iskokompajlirali ga korišćenjem komandne linije, treba ispratiti sledeće korake:

Korak 1. Otvoriti konzolu (command prompt) razvojnog okruženja. U OS Windows 8, kliknuti na **Start** ekran, otvoriti direktorijum **Visual Studio Tools** izabrati ikonicu programa **Developer Command Prompt**. U prethodnim verzijama Windows operativnog sistema, izabrati dugme **Start**, zatim proširiti dugme **All Programs**, **Microsoft Visual St**udio, i **Visual Studio Tools**, i izabrati **Developer Command Prompt**.

U zavisnosti od verzije operativnog sistema na računaru i sistemskih podešavanja (system security configuration), moguće je da je neophodno otvoriti meni za program **Developer Command Prompt** (desni klik miša na ikonicu) i zatim izabrati opciju **Run as Administrator** da bi uspešno kreirali i pokrenuli aplikaciju korišćenjem koraka koji slede.

Korak 2. U okviru konzole (command prompt-a), kreirati direktorijum za vaš izvorni fajl i postaviti trenutni radni direktorijum (current working directory). Na primer, otkucati md c:\simple i pritisnuti Enter da kreirate direktorijum naziva simple, a zatim otkucati cd c:\simple i pritisnuti Enter da bi taj direktorijum postavili kao radni direktorijum.

KREIRANJE IZVORNOG C FAJLA U PROGRAMU NOTEPAD

Kada hoćete da kompajlirate program iz komandne linije, jedino što je potrebno je da kod aplikacije pišete u notepad-u i to snimite kao običan fajl

Korak 3. U okviru konzole otkucati notepad i pritisnuti taster Enter. Otvoriće se program Notepad za kucanje teksta.

Korak 4. U programu Notepad, otkucati sledeće linije koda:

```
#include <stdio.h>
int main()
{
    printf("This is a native C program.\n");
    return 0;
}
```

Korak 5. U meniju programa Notepad izabrati **File, Save** da bi se otvorio **Save As** dijalog. Pronađite i izaberite direktorijum koji ste prethodno kreirali. U polju za unos **File name**, ukucati naziv pod kojim želite da zapamtite vaš fajl sa izvornim kodom (source file), na primer simple.c — i zatim u padajućoj listi **Save as type**, izabrati tip **All Files (*.*)**. Izabrati zatim dugme **Save** da kreirate C izvorni fajl u radnom direktorijumu.

Korak 6. U konzoli, otkucati dir i pritisnuti Enter. Na ekranu konzole će se pojaviti tekst sličan kao na sledećoj slici 1:

```
C:\simple>dir
 Volume in drive C has no label.
 Volume Serial Number is CC62-6545

 Directory of C:\simple

10/02/2012  03:46 PM    <DIR>          .
10/02/2012  03:46 PM    <DIR>          ..
10/02/2012  03:36 PM               102 simple.c
               1 File(s)            102 bytes
               2 Dir(s)  514,900,566,016 bytes free
```

Slika 1. Struktura radnog direktorijuma u kojem smo kreirali C izvorni fajl

Detalji prethodne slike će se razlikovati od računara do računara. Ako ne vidite fajl u direktijumu, proverite poziciju gde ste sačuvali fajl iz Notepad programa.

KOMPAJLIRANJE I IZVRŠAVANJE PROGRAMA

U komandnoj liniji otkucati komandu cl zajedno sa nazivom vašeg izvornog fajla—na primer, cl simple.c—i pritisnuti taster Enter da bi kompajlirali program

Korak 7. U konzoli (command prompt), specificirati komandu cl zajedno sa nazivom vašeg izvornog fajla—na primer, cl simple.c—i pritisnuti taster Enter da bi kompajlirali program. Kompajler cl.exe generiše jedan fajl sa ekstenzijom .obj koji sadrži iskompajlirani kod, a zatim pokreće i linker da bi napravio izvršni program koji ima isto ime kao i ime vašeg izvornog fajla, ali sa ekstenzijom .exe— na primer, simple.exe.

Nakon pokretanja kompajlera odgovarajućom komandom (cl) na ekranu konzole se prikazuje listing uspešnog / bezuspešnog kompajliranja i generisanja izvršnog (executable) fajla. Izlaz bi mogao da izgleda kao na slici 2:

```
Microsoft (R) C/C++ Optimizing Compiler Version 17.00.50727.1 for x86
Copyright (C) Microsoft Corporation.  All rights reserved.

simple.c
Microsoft (R) Incremental Linker Version 11.00.50727.1
Copyright (C) Microsoft Corporation.  All rights reserved.

/out:simple.exe
simple.obj
```

Slika 2. Listing u komandnoj liniji nakon kompajliranja programa korišćenjem Visual Studio kompajlera.

Broj verzije alata za kompajliranje zavisi od verzije razvojnog okruženja Visual Studio ili od ažuriranih dodataka koji su instalirani.

Korak 8. Da bi pokrenuli program, jednostavno otkucati naziv programa bez ikakvih ekstenzija—na primer, simple—i pritisnuti taster Enter. Program će ispisati sledeći tekst i prestati sa radom:

```
This is a native C program.
```

Dodatna objašnjenja za kompajliranje C programa korišćenje komandne linije možete pronaći na linku - **Compiling a C Program on the Command Line** https://msdn.microsoft.com/en-us/library/bb384838(v=vs.120).aspx, dok za opis kompajliranja C++ programa korišćenjem komandne linije možete posetiti sledeći link: **Compiling a Native C++ Program on the Command** Line - https://msdn.microsoft.com/en-us/library/ms235639(v=vs.140).aspx.

Vežba - Ulaz i izlaz različitih podataka

Ulazne funkcije, izlazne funkcije, brojevi, karakteri, ASCII kodovi

- *Primeri 1*
- *Primeri 2*
- *Primeri 3*
- *Primeri 4*
- *Primeri 5*

10

PRIMERI 1

Primer: Ulaz / izlaz sa ASCII kodovima

U nekom od prethodnih primera, uneli smo karakter u program ali on nije zapamćen u obliku ASCII koda jer smo koristili konverziju %c. Prilikom štampanja smo takođe koristili konverziju %c, pa je izvršeno štampanje istog tog karaktera. Ukoliko prilikom štampe koristimo konverziju %d biće oštampan ASCII kod karaktera:

```
#include <stdio.h>
int main(){
    char var1;
    printf("Enter character: ");
    scanf("%c",&var1);
    printf("You entered %c.\n",var1);
/* \n prints the next line(performs work of enter).
*/
    printf("ASCII value of %d",var1);
    return 0;
}
```

Izlaz programa je

```
Enter character:
g
103
```

Kada unosemo *'g'* čuva se ASCII kod 103 umesto karaktera *g*. U zavisnosti od tipa konverzije vrši se ili štampanje karaktera ili ASCII koda. U nastavku je dat primer konverzije ASCII koda u karakter prilikom štampe.

```
#include <stdio.h>
int main(){
    int var1=69;
    printf("Character of ASCII value 69: %c",69);
    return 0;
}
```

Izlaz prethodnog programa je:

```
Character of ASCII value 69: E
```

ASCII vrednost karaktera 'A' je 65, 'B' je 66 i tako dalje do 'Z' čiji je ASCII kod 90. Slično tome, ASCII vrednost karaktera 'a' je 97, 'b' je 98, …, a slova 'z' je 122.

PRIMERI 2

Varijacije štampanja realnih i celobrojnih konstanti u C-u:

Primer 1. Varijacije štampanja realnih i celobrojnih konstanti u C-u:

```
#include<stdio.h>
int main(){
    printf("Case 1:%6d\n",9876);
/*  Prints the number right justified within 6 columns  */
    printf("Case 2:%3d\n",9876);
/* Prints the number to be right justified to 3 columns
but, there are 4 digits so number is not right justified
*/
    printf("Case 3:%.2f\n",987.6543);
/* Prints the number rounded to two decimal places */
    printf("Case 4:%.f\n",987.6543);
/* Prints the number rounded to 0 decimal place, i.e,
rounded to integer */
    printf("Case 5:%e\n",987.6543);
/* Prints the number in exponential notation(scientific
notation) */
    return 0;
}
```

Izlaz programa je:

```
Case 1:  9876
Case 2:9876
Case 3:987.65
Case 4:988
Case 5:9.876543e+002
```

Primer 2. Razne varijacije ulazno izlaznih operacija sa celobrojim i realnim brojevima

```
#include <stdio.h>
int main(){
    int a,b;
    float c,d;
    printf("Enter two intgers: ");
/*Two integers can be taken from user at once as
below*/
    scanf("%d%d",&a,&b);
    printf("Enter intger and floating point numbers:
");
/*Integer and floating point number can be taken at
once from user as below*/
    scanf("%d%f",&a,&c);
    return 0;
```

PRIMERI 3

Primer sa ulazno izlaznim operacijama nad karakterima

Primer upotrebe putchar i getchar funkcija za rad sa karakterima

```
#include <stdio.h>
void main()
{
            int c1, c2;
            c1 = getchar();
            printf("------------\n");
            c2 = getchar();
            printf("c1 = %d, c2 = %d\n",c1, c2);
            printf("c1 = %c, c2 = %c\n",c1, c2);
            putchar(c1); /* isto je kao i
printf("%c",c1); */
            putchar(c2); /* isto je kao i
printf("%c",c2); */
            putchar('\n');
            /* Za ispisivanje karaktera a */
            putchar('a');
            /* dozvoljeno je : printf("abc");
printf("a"); */
            /* nije dozvoljeno : printf('a');
putchar('abc'); putchar("abc"); */
}
```

Primer: Kopiranje sa ulaza na izlaz korišćenjem getchar i putchar (toupper prebacuje mala slova u velika).

```
#include <stdio.h>
#include <ctype.h>
/* kopiranje ulaza na izlaz */
int main(void) {
      int c;
      c=getchar();
      while(c!=EOF) {
          putchar(toupper(c));
          c=getchar();
      }
     return 0;
}
```

Primer: korišćenje getchar i putchar funkcija, učitavanje karaktera dok se ne dođe do kraja datoteke.

```
#include <stdio.h>
int main()
{
     int c;
     c = getchar();
     while (c != EOF) {
         putchar(c);
         c = getchar();
     }
}
```

PRIMERI 4

Formatiranje izlaza za različite tipove podataka

U nastavku je dat još jedan primer formatiranja

```
#include<stdio.h>
main()
{
          printf("The color: %s\n", "blue");
          printf("First number: %d\n", 12345);
          printf("Second number: %04d\n", 25);
          printf("Third number: %i\n", 1234);
          printf("Float number: %3.2f\n", 3.14159);
          printf("Hexadecimal: %x\n", 255);
          printf("Octal: %o\n", 255);
          printf("Unsigned value: %u\n", 150);
          printf("Just print the percentage sign %%\n", 10);
}
```

Izlaz prethodnog primera biće:

```
The color: blue
First number: 12345
Second number: 0025
Third number: 1234
Float number: 3.14
Hexadecimal: ff
Octal: 377
Unsigned value: 150
Just print the percentage sign %
```

Napomena: U poslednjem printf izrazu biće oštampan samo znak procenta. Broj 10 u ovom izrazu biće zanemaren i neće biti oštampan. Da bi oštampali znak procenta to možemo uraditi na sledeći način:

```
printf("%2d%%\n", 10);
```

Izlaz ce biti 10%.

PRIMERI 5

Do sada je pokazan najveći deo načina za formatiranje podatka kod izlaznih operacija, ali postoji jedan tip kod kojih se formatira razlikuje od ostalih, a to su stringovi

Pogledajmo sledeći primer:

```
#include<stdio.h>
void main()
{
          printf(":%s:\n", "Hello, world!");
          printf(":%15s:\n", "Hello, world!");
          printf(":%.10s:\n", "Hello, world!");
          printf(":%-10s:\n", "Hello, world!");
          printf(":%-15s:\n", "Hello, world!");
}
```

Izlaz programa je:

```
:Hello, world!:
:  Hello, world!:
 :Hello, wor:
:Hello, world!:
 :Hello, world!   :
```

Kao što se može primetiti, formatiranje teksta pri ispisu stringa se veoma različito ponasa u zavisnosti od broja koji koristimo za konverziju formata.

- Izraz printf(“:%s:\n”, “Hello, world!”); štampa string u slobodnom formatu
- Izraz printf(“:%15s:\n”, “Hello, world!”); štampa string, ali samo na 15 pozicija. Ukoliko je dužina stringa manja od 15, ostale pozicije će biti ispunjene prazninama, poravnavanje teksta je sa desne strane.
- Izraz printf(“:%.10s:\n”, “Hello, world!”); štampa samo prvih 10 karaktera stringa.
- Izraz printf(“:%-10s:\n”, “Hello, world!”); štampa najmanje 10 karaktera stringa. Ukoliko je dužina teksta manje, dodaju se praznine na kraju
- Izraz printf(“:%-15s:\n”, “Hello, world!”); štampa najmanje 15 karaktera stringa. Tekst je u ovom slučaju manji od 15 tako da se dodaju praznine sa desne strane (definisano znakom minus.)

Vežba - Tipovi podataka, promenljive, operatori

podatak, promenjiva, enum, extern, static, operator, pokazivač

- *Osnovni primeri sa tipovima podataka*
- *Upotreba različitih tipova podataka i operatora*
- *Upotreba adresnog i indirektnog operatora*
- *Upotreba binarnih operatora*
- *Upotreba binarnih operatora u kombinaciji sa operatorom dodele*

11

OSNOVNI PRIMERI SA TIPOVIMA PODATAKA

Da bi smo odredili veličinu memorije koju neki tip podatka zauzima koristimo operator sizeof()

Primer: Štampanje realnih brojeva u različitim izlaznim formatima

```
#include <stdio.h>
void main()
{
          int a=21,d=5;
          float c=8.953, b=6.0;
          printf("d=%d c=%f\n",d,c);
          printf("a + d/c + b=%f\n",a+ d/c + b);
          printf("|%5f |%5.3f| %5.0f|%5.1f\n",c,c,c,c);
}
```

Zadatak: Napisati C program koji na standardni izlaz ispisuje broj bajtova koji zauzimaju određeni tipovi podataka:

```
#include <stdio.h>
void main()
{
          printf("char: %ld bajt\n", sizeof(char));
          printf("short: %ld bajta\n", sizeof(short));
          printf("int : %ld bajta\n", sizeof(int));
          printf("long= %ld bajta\n", sizeof(long));
          printf("unsigned= %ld bajta\n", sizeof(unsigned));
          printf("float= %ld bajta\n", sizeof(float));
          printf("double= %ld bajta\n", sizeof(double));
}
```

UPOTREBA RAZLIČITIH TIPOVA PODATAKA I OPERATORA

U nastavku su dati razni primeri aritmetičkih operacija nad promenljivama istog ili različitog tipa, kao i primeri sa konverzijom karaktera u njegov ASCII kod

Primer1: Zbir dva broja istog tipa

```
#include <stdio.h>
void main()
{
   int x,y;   // deklaracija ide na pocetku
   x=23;    // inicijalizacija vrednosti
   y=56; // inicijalizacija vrednosti
   printf("Njihov zbir je %d\n",x+y);  // stampanje
vrednosti
}
```

Primer2: Zbir dva broja različitog tipa

```
#include <stdio.h>
void main()
{
  int x=23;float y=5.6;
  printf("Njihov zbir je %f\n",x+y);
}
```

Primer3: Množenje dva velika broja

```
#include <stdio.h>
void main()
{
  long x=2345678,y=67890;
  printf(" Njihov proizvod je %ld\n",x*y);
}
```

Primer4: Štampanje karaktera i njegovog ASCII koda

```
#include <stdio.h>
void main()
{
            char c='A';
            printf(" %c %d\n",c,c);
}
```

Primer5: Štampanje realnog broja u različitim formatima

```
#include <stdio.h>
void main()
{
            int a=21,d=5;
            float c=8.953, b=6.0;
            printf("d=%d c=%f\n",d,c);
            printf("a + d/c + b=%f\n",a+ d/c + b);
            printf("|%5f |%5.3f|
%5.0f|%5.1f\n",c,c,c,c);
}
```

UPOTREBA ADRESNOG I INDIREKTNOG OPERATORA

*Adresni operator & određuje adresu promenljive, dok indirektni operator * odredjuje podatak koji se nalazi na adresi na koju pokazuje pokazivač*

Vec smo spomenuli da se svaka promenljiva čuva na memorijskoj lokaciji koju određujemo korišćenjem adresnog operatora &. U nastavku je dat primer koji štampa adresu definisanih promenljivih:

```
#include <stdio.h>

int main()
{
   int  var1;
   char var2[10];

   printf("Address of var1 variable: ");
   printf("%x\n",&var1);

   printf("Address of var2 variable: ");
   printf("%x\n",&var2);

   return 0;
```

Nakon kompajliranja i izvršavanja programa, na ekranu se dobija sledeći rezultat:

```
Address of var1 variable: 0xbfebd5c0
Address of var2 variable: 0xbfebd5b6
```

Primer: Definisati pokazivač na celobrojnu promenljivu kojoj je dodeljena neka proizvoljna vrednost. Oštampati adresu promenljive i njenu vrednost korišćenjem pokazivača:

```
#include <stdio.h>

int main ()
{
   int  var = 20;   /* actual variable declaration
*/
   int  *ip;        /* pointer variable declaration
*/

   ip = &var;  /* store address of var in pointer
variable*/

   printf("Address of var variable: %x\n", &var  );

   /* address stored in pointer variable */
   printf("Address stored in ip variable: %x\n", ip
);

   /* access the value using the pointer */
   printf("Value of *ip variable: %d\n", *ip );

   return 0;
}
```

Nakon izvršavanja programa dobija se sledeći rezultat:

```
Address of var variable: bffd8b3c
Address stored in ip variable: bffd8b3c
Value of *ip variable: 20
```

UPOTREBA BINARNIH OPERATORA

Operatori nad bitovima su specijalni tipovi operatora koji se najčešće koriste za programiranje procesora, jer ubrzavaju procesiranje i smanjuju snagu

U nastavku je dat primer u cilju boljeg razumevanja i korišćenja binarnih operatora:

```
#include <stdio.h>

void main()
{
   unsigned int a = 60; /* 60 = 0011 1100 */
   unsigned int b = 13; /* 13 = 0000 1101 */
   int c = 0;
   c = a & b;       /* 12 = 0000 1100 */
   printf("Line 1 - Value of c is %d\n", c );
   c = a | b;       /* 61 = 0011 1101 */
   printf("Line 2 - Value of c is %d\n", c );
   c = a ^ b;       /* 49 = 0011 0001 */
   printf("Line 3 - Value of c is %d\n", c );
   c = ~a;          /*-61 = 1100 0011 */
   printf("Line 4 - Value of c is %d\n", c );
   c = a << 2;     /* 240 = 1111 0000 */
   printf("Line 5 - Value of c is %d\n", c );
   c = a >> 2;     /* 15 = 0000 1111 */
   printf("Line 6 - Value of c is %d\n", c );
}
```

Izlaz:

```
Line 1 - Value of c is 12
Line 2 - Value of c is 61
Line 3 - Value of c is 49
Line 4 - Value of c is -61
Line 5 - Value of c is 240
Line 6 - Value of c is 15
```

UPOTREBA BINARNIH OPERATORA U KOMBINACIJI SA OPERATOROM DODELE

*Kada su u kombinaciji sa operatorima dodele, operatori nad bitovima imaju sličan efekat kao i operatori dodele nad osnovnim tipovima podataka (+=, -=, *=, ...)*

U nastavku je dat primer korišćenja različitih binarnih operatora u kombinaciji sa operatorom dodele:

```
#include <stdio.h>
void main()
{
   int a = 21;
   int c ;
   c <<=  2;
   printf("Line 7 - <<= Operator Example, Value of c = %d\n", c );
   c >>=  2;
   printf("Line 8 - >>= Operator Example, Value of c = %d\n", c );
   c &=  2;
   printf("Line 9 - &= Operator Example, Value of c = %d\n", c );
   c ^=  2;
   printf("Line 10 - ^= Operator Example, Value of c = %d\n", c );
   c |=  2;
   printf("Line 11 - |= Operator Example, Value of c = %d\n", c );
}
```

Rezultat:

```
Line 7 - <<= Operator Example, Value of c = 44
Line 8 - >>= Operator Example, Value of c = 11
Line 9 - &= Operator Example, Value of c = 2
Line 10 - ^= Operator Example, Value of c = 0
Line 11 - |= Operator Example, Value of c = 2
```

Zadaci za samostalan rad

C, programiranje, tipovi podataka, sintaksa C jezika

- *Zadaci za samostalno vežbanje*

12

ZADACI ZA SAMOSTALNO VEŽBANJE

Korišćenjem materijala sa predavanja i vežbi za ovu nedelju uraditi sledeće zadatke:

1. Napisati program koji ispisuje poruku "Zdravo /*\" (svi karakteri su vidljivi prilikom ispisa).

2. Napisati C program koji će da ispisuje “Dobrodošli na C/C++” 5 puta.

3. Napraviti program koji za unete stranice A i B, računa površinu i obim pravougaonika i ispisuje ih na standardnom izlazu.

4. Napraviti program koji za unet poluprečnik, računa površinu i obim kruga.

5. Napisati program koji učitava podatke za poluprečnik r osnove kupe i izvodnicu s. Prikazuju se podaci za B, M, P i V prave kupe

6. Napisati program koji za uneti karakter ispisuje njegovu numeričku vrednost. Uraditi ovo koriščenjem scanf i printf.

7. Definisati PI kao konstantu preprocesora; Napisati program koji za uneti poluprečnik izračunava i prikazuje obim i povšinu kruga korišćenjem konstante PI.

8. Napisati program koji ispisuje vrednost broja Pi sa 5 decimalnih mesta. (Pomoc: pogledati sadrzaj zaglavlja math.h pomocu man komande)

9. Napisati program koji računa godišnju kamatu na iznos depozita u banci. Unosimo količinu novca i kamatnu stopu na mesečnom nivou, a aplikacija nam kaže koliko smo novca zaradili od kamate za godinu dana. Razmisliti koje tipove podataka je optimalno koristiti za ovaj problem. Oštampati rezultat na 3 i 5 decimala, kao i u eksponencijalnom obliku.

10. Napraviti program u koji se unosi broj od 1 do 7, a on ispisuje na konzoli dan u nedelji. Ako je unet neki drugi broj, program ispisuje grešku.

11. Kreirati nabrojivi tip za dane u nedelji i ispitati da li je u pitanju radni dan ili vikend.

12. Napisati program u kojem se prvo deklarise promenljiva slovo, a zatim se njena vrednost inicijalizuje na A i ispisuje na ekran.

Zaključak

REZIME

Na osnovu svega obrađenog možemo izvesti sledeći zaključak:

Instrukcije programa moraju biti u kompjuterskoj memoriji da bi program radio. Postoje tri glavne memorijske lokacije na računaru CPU, RAM i rezervna memorija. Programi obično koriste RAM memoriju za skladištenje instrukcija i podataka. Instrukcije i podaci su smešteni na odgovarajućim adresama. Računar skladišti informacije u serijama koje se sastoje iz nula i jedinica, gde svaka nula i jedinica predstavljaju jedan bit.

Svaki C/C++ program mora imati jednu i samo jednu glavnu funkciju, funkciju main(). Instrukcije programa se pišu između dve velike zagrade, a instrukcije se u principu završavaju sa znakom tačka-zarez.

Neke informacije su numeričke, druge su tekstualne. Osnovni tipovi podataka su celi brojevi, decimalni brojevi i tekstovi. Međutim, svi tipovi podataka imaju zajedničku karakteristiku a to je veličina tipa podatka koja se meri u bajtovima. Veličina tipa podatka istovremeno predstavlja i njegov opseg, odnosno najmanji i najveći broj koji može da bu smešten u taj tip podatka.

Promenljive se koriste u dve svrhe. One obezbedjuju pristup odgovarajućoj informaciji, a takođe rezervišu deo memorije koja je neophodna da se smesti neka informacija. Neophodno je deklarisati promenljivu pre njenog korišćenja. Možete koristiti adresni operator &, da bi ste odredili adresu promenljive. Svrha promenljive je da se pomoću nje skladišti neki podatak u memoriji. Stoga je prvi logični korak nakon kreiranja promenljive, da se specificira informacija koja će biti smeštena na adresu rezervisanu od strane promenljive.

Najveći deo računara ima za cilj da obavlja složene proračune i operacije. Računar, osim mogućnosti da skladišti ogromnu količinu podataka, takođe ima mogućnost da izračuna operacije mnogo brže nego što to mi možemo. C podržava rad sa svim aritmetičkim operacijama.

C, za razliku od nekih programskih jezika, nema operator stepenovanja, ali zato ima ugrađenu funkciju *pow* koja se nalazi u standardnoj biblioteci *math.h.*