UVOD

JEDNOSTAVNI C PROGRAM

Slijedeći program je napisan u C programskom jeziku.

```
#include <stdio.h>

main()
{
      printf("Programiranje u C-u  je lako.\n");
}
```

```
Ispis programa
Programiranje u C-u je lako.
_
```

Za pisanje C-programa koriste se mala i velika slova, ali sve naredbe u C-u moraju biti napisane malim slovima. Početak C programa označava se s:

```
main()
```

Zagrade koje se nalaze nakon ključne riječi main označavaju da program ne prima nikakve argumente. (što ce biti kasnije objašnjeno).

Dvije zagrade, { i }, označavaju početak i kraj dijelova programa. Svrha izraza

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

je da omogući uporabu printf naredbe za ispis. Tekst koji će ispisati printf() mora biti u navodnicima. Program ima samo jednu naredbu

```
printf("Programiranje u C-u je lako.\n");
```

printf() je zapravo funkcija (procedura) u C-u koja se koristi za formatirani ispis varijabli i teksta. Ondje gdje se tekst pojavljuje u navodnicima " ", ispisuje se bez promjene. Ipak postoje neke iznimke - označavaju se znakovima: \ i % . Ovi znakovi su modifikatori. U našem primjeru \ koji prethodi znaku n predstavlja oznaku novog reda. Tako program ispisuje

```
Programiranje u C-u je lako.
```

i kursor se postavlja na početak nove linije. Kao što cemo kasnije vidjeti, znak koji slijedi iza znaka \ odreduje što se ispisuje, tj. 8 praznih mjesta (tab), prazan ekran, prazna linija itd. Važno je zapamtiti da se sve naredbe u C-u završavaju točka-zarezom ;

Zakljucak

* C pravi razliku izmedu velikih i malih slova, koristite mala slova i pokušajte ne davati varijablama imena s velikim slovima
* ključne riječi se pišu malim slovima
* naredbe se završavaju točka-zarezom
* izvršavanje programa počima prvom naredbom od main()
* vitičaste zagrade {} oznacavaju početak i kraj programskog bloka
* niz znakova (literalni string) se označava navodnicima
* printf() se moze koristiti za prikaz teksta na ekranu
* \n označava položaj kursora na početku nove linije

O VARIJABLAMA

Programski jezik C ima ČETIRI osnovna tipa podataka. Varijable koje definira korisnik moraju biti deklarirane prije nego će se koristiti u programu.

Naviknite se deklarirati varijable koristeći mala slova. Zapamtite da C razlikuje mala i velika slova, tako da iako dvije varijable dolje navedene imaju isto ime, u C-u se smatraju različitim varijablama.

```
sum , Sum
```

Deklaracija varijabli vrši se nakon otvorene zagrade main(),

```
#include <stdio.h>

main()
{
        int sum;
        sum = 500 + 15;
        printf("Suma 500 i 15 iznosi %d\n", sum);
}
```

Ispis programa:

```
Suma 500 i 15 iznosi 515
```

Moguće je deklarirati varijable i na drugim mjestima u programu, ali započet ćemo polako i kasnije razmatrati varijacije.

Osnovni format deklariranja varijabli je

```
tip_pod   var, var, ... ;
```

gdje je tip_pod jedan od četiri osnovna tipa podataka: integer, character, float ili double .

Program deklarira varijablu sum kao tip INTEGER (int). Varijabli sum se onda pridružuje vrijednost 500 + 15 koristeći operator pridruživanja, znak = .

```
sum = 500 + 15;
```

Pogledajmo sada pobliže printf() funkciju. Ima dva argumenta, odvojena zarezom. Pogledajmo prvi argument,

```
"Suma 500 i 15 iznosi %d\n"
```

Znak % je posebni znak u C-u. Koristi se za ispis vrijednosti varijable. Kad se program izvršava, C ispisuje tekst dok ne naiđe na znak % . Kada ga pronađe, traži slijedeći argument (u ovom slučaju sum), ispisuje njegovu vrijednost, te nastavlja.

Znak d koji prati % označava da se očekuje integer. Tako, kada se naiđe na znak %d printf() funkcija traži slijedeći argument (u ovom slučaju varijablu sum, koja iznosi 515), i ispisuje ga. Znak \n se nakon toga izvršava što znači da se kursor prebacuje u novi red.

Tako ispis programa izgleda:

```
Suma 500 i 15 iznosi  515
```

Formatiranje ispisa za printf :

```
Specijalni znakovi
\n      prelazak u novi red
\t      tabulator
\r      postavljanje kursora na početak linije
```

\f postavljanje margine
\v vertikalni tabulator

Specifikarori formata
%d decimalni cjelobrojni (integer)
%c znak (character)
%s string ili niz znakova (character array)
%f realni brojevi s pomičnim zarezom (float)
%e realni brojevi s pomičnim zarezom dvostruke točnosti (double)

Slijedeći program ispisuje dvije cjelobrojne vrijednosti razdvojene tabulatorom:

To se postiže korištenjem specijalnog znaka \t

```
#include <stdio.h>

main()
{
	int sum, value;
	sum = 10;
	value = 15;
	printf("%d\t%d\n", sum, value);
}
```

Ispis programa

```
10	15
```

KOMENTARI

Dodavanje komentara programu je poželjno. Komentari se C programu mogu dodati kako je pokazano,

```
/* bla bla bla bla bla bla */
```

Primijetite da /* otvara polje komentara, a da ga */ zatvara. Komentari mogu obuhvaćati više linija. Međutim, ne mogu se gnijezditi jedan unutar drugog.

```
/* ovo je komentar. /* ovaj komentar je unutra */ pogrešno */
```

U gornjem primjeru, prva pojava znaka */ zatvara komentar za čitavu liniju, što znači da riječ se pogrešno interpretira kao C izraz ili varijabla, i u ovom primjeru, generira grešku.

Zašto se komentari koriste

* dokumentiranje varijabli i njihovo korištenje
* objašnjavanje složenih dijelova programa
* opis programa, autor, datum, promjene, revizije itd.

Osnovna struktura C programa
C programi su u osnovi građeni na slijedeći način, kao skup dobro definiranih dijelova.

```
/* ZAGLAVLJE                    */
/* Sadrži ime programa, ime autora, broj revizije...*/

/* INCLUDE dio                  */
```

```
/* sadrži  #include izraze          */


/* KONSTANTE I TIPOVI          */
/* sadrži tipove i  #define izraze          */


/* GLOBALNE  VARIJABLE             */
/* ako postoje globalne varijable deklariraju se ovdje    */


/* FUNKCIJE                    */
/* funkcije koje definira korisnik               */


/* main()                      */

int main()
{


}
```

Pridržavanje ove dobro definirane strukture učinit će vaše programe

* lakim za čitanje
* lakim za modificiranje

VIŠE O VARIJABLAMA

Imena varijabli moraju započinjati slovom ili znakom _, a nakon toga bilo kakvom kombinacijom slova, znaka _, ili znamenki 0 - 9. Slijede primjeri dozvoljenih imena varijabli,

```
zbroj
izlazna_zastavica
i
Jerry7
Broj_poteza
_važeća_zastavica
```

Osigurajte da koristite imena sa značenjem za vaše varijable. Razlozi za to su,

* imena sa značenjem na prvi pogled pokazuju što varijabla radi
* lakša su za razumijevanje
* nema povezanosti s korištenjem prostora u izvršnom kodu (.EXE file)
* čine program lakšim za čitanje

IMENA VARIJABLI I PREFIKSI KOD PISANJA WINDOWS ILI OS/2 PROGRAMA

Tijekom razvoja OS/2, postalo je uobičajeno dodavati prefikse imenima varijabli da bi se označio tip podatka.

To je omogućavalo programerima da identificiraju tip podatka bez gledanja u deklaraciju, tako da su lako mogli provjeriti izvode li dozvoljene operacije na tom tipu i po mogućnosti, smanjiti broj grešaka.

Prefiks	Svrha ili tip podatka
b	byte vrijednost
c	brojilo ili veličina
clr	varijabla koja označava boju
f	polja bitova ili zastavica (flag)
h	a handle
hwnd	a window handle
id	identifikator
l	long integer
msg	poruka
P	pointer
rc	povratna vrijednost
s	short integer
ul	unsigned long integer
us	unsigned short integer
sz	string varijabla završena nulom
psz	pointer na sz

U kodu napisanom za Windows ili OS/2, možete vidjeti varijable napisane po ovim pravilima.

TIPOVI PODATAKA I KONSTANTE

Tip označava način kodiranja i operacije koje je dozvoljeno primijeniti na programske varijable, čija je vrijednost smještena u memoriji računala. Položaj varijable u memoriji naziva se adresa ili memorijska referenca varijable.
Četiri osnovna tipa podataka su:

* Cjelobrojni decimalni (INTEGER)

U ovaj tip podataka spadaju pozitivni i negativni cijeli brojevi. Također imamo i unsigned (samo pozitivni cijeli brojevi). Postoje i short i long cijeli brojevi.

Ključna riječ za definiranje cijelog broja je,

```
int
```

Primjer integer vrijednosti je 32. Primjer deklaracije integer varijable sum je,

```
int sum;
sum = 20;
```

* Realni brojevi s pomičnim zarezom (FLOATING POINT)

Obuhvaćeni su i pozitivni i negativni realni brojevi. Ključna riječ za definiranje realnog broja je,

```
float
```

Primjer float vrijednosti je 34.12. Primjer deklaracije float varijable money je,

```
float money;
money = 0.12;
```

* Realni brojevi s eksponentom (DOUBLE)

To su eksponencijalni brojevi, pozitivni i negativni. Ključna riječ koja se koristi za definiranje double varijabli je,

```
double
```

Primjer double vrijednosti je 3.0E2. Primjer deklariranja double varijable big je,

```
double big;
big = 312E+7;
```

* Znak (CHARACTER)

Ključna riječ za deklariranje znaka je,

```
char
```

Primjer znakovne vrijednosti je slovo A. Primjer deklariranja znakovne varijable letter je,

```
char letter;
letter = 'A';
```

Primijetite da se pridruživanje znaka A varijabli letter vrši stavljanjem znaka u jednostruke navodnike. Zapamtite zlatno pravilo: Jedan znak - jednostruki navodnici.

Primjer programa koji pokazuje uporabu svih tipova podataka

```
#include < stdio.h >

main()
{
        int   sum;
        float money;
        char  letter;
        double pi;

        sum = 10;               /* pridruživanje integer vrijednosti */
        money = 2.21;           /* pridruživanje  float vrijednosti */
        letter = 'A';          /* pridruživanje character vrijednosti */
        pi = 2.01E6;            /* pridruživanje double vrijednosti */

        printf("vrijednost sum = %d\n", sum );
        printf("vrijednost money = %f\n", money );
        printf("vrijednost letter = %c\n", letter );
        printf("vrijednost pi = %e\n", pi );
}
```

Ispis programa

```
vrijednost sum = 10
vrijednost money = 2.210000
vrijednost letter = A
vrijednost pi = 2.010000e+06
```

INICIJALIZIRANJE VARIJABLI PRI DEKLARIRANJU

Za razliku od PASCAL-a, C varijable se mogu inicijalizirati na neku vrijednost kada se deklariraju. Pogledajmo slijedeću deklaraciju koja deklarira integer varijablu count koja je inicijalizirana na vrijednost 10.

```
int  count = 10;
```

JEDNOSTAVNO PRIDJELJIVANJE VRIJEDNOSTI VARIJABLAMA

Operator = se koristi za pridruživanje vrijednosti varijablama. Pogledajmo slijedeći izraz, koji pridjeljuje vrijednost 32 integer varijabli count, i slovo A character varijabli letter

```
count = 32;
```

```
letter = 'A';
```

VRIJEDNOST VARIJABLI PRI DEKLARACIJI

Ispitajmo koja se vrijednost automatski pridjeljuje varijabli kada je deklariramo (default). Da bismo to učinili, razmotrimo slijedeći program, koji deklarira dvije varijable, count koja je cjelobrojna, i letter koja je znakovna.

Nijedna varijabla nije prije inicijalizirana. Vrijednost svake od varijabli se ispisuje korištenjem printf() funkcije.

```
#include <stdio.h>

main()
{
      int   count;
      char  letter;

      printf("Count = %d\n", count);
      printf("Letter = %c\n", letter);
}
```

Ispis programa

```
Count = 26494

Letter = f
```

Iz ispisa programa može se vidjeti da su vrijednosti koje se pridružuju varijablama pri deklaraciji različite od nule. U C-u, to je uobičajeno, i programeri moraju osigurati da su varijablama pridružene vrijednosti prije no što ih koriste.

Ako bi se program ponovno startao, ispis bi mogao imati i različite vrijednosti za svaku od varijabli. Nikad ne možemo pretpostaviti da će varijable deklarirane na ovaj način poprimiti neku određenu vrijednost.

Neki prevodioci (compileri) mogu generirati upozorenje kod ovakovih varijabli, pa Turbo C (Borland) generira slijedeće upozorenje,

```
possible use of 'count' before definition in function main
```

MIJENJANJE BROJEVNE BAZE

Brojčani podaci mogu se izraziti u bilo kojoj bazi jednostavnim mijenjanjem modifikatora, npr. decimalno, oktalno, ili heksadecimalno. To se postiže slovom koje prati znak % u printf funkciji.

```
#include <stdio.h>

main() /* Ispisuje istu vrijednost u decimalnoj, heksadecimalnoj i oktalnoj notaciji */

{
      int   number = 100;

      printf("Kao decimalan broj je zapisan sa %d\n", number);
      printf("Kao heksadecimalan broj je zapisan sa %x\n", number);
      printf("Kao oktalan broj je zapisan sa %o\n", number);
      /* što je s  %X\n kao argumentom?  */
}
```

Ispis programa

Kao decimalan broj je zapisan sa 100

Kao heksadecimalan broj je zapisan sa 64

Kao oktalan broj je zapisan sa 144

DEFINIRANJE VARIJABLI U OKTALNOJ I HEKSADECIMALNOJ NOTACIJI

Često kada piše programe, programeru je potrebno zapisati brojeve u nekoj drugoj notaciji osim decimalne.

Cjelobrojne konstante mogu se definirati u oktalnoj ili heksadecimalnoj notaciji korištenjem određenog prefiksa, npr., za definiranje cijelog broja u oktalnoj notaciji koristi se %o

```
int    sum = %o567;
```

Za definiranje cijelog broja u heksadecimalnoj notaciji koristi se %0x

```
int    sum  = %0x7ab4;
int    flag = %0x7AB4;     /* mogu se koristiti i velika i mala slova */
```

VIŠE O FLOAT I DOUBLE VARIJABLAMA

C ispisuje i float i double varijable na šest decimalnih mjesta. To se NE odnosi na preciznost (točnost) na koju je broj zapravo pohranjen, već samo na to koliko decimalnih mjesta printf() koristi za prikaz tog tipa varijabli.

Slijedeći program ilustrira kako se različiti tipovi podataka deklariraju i ispisuju,

```
#include <stdio.h>

main()
{
      int     sum = 100;
      char    letter = 'Z';
      float   set1 = 23.567;
      double  num2 = 11e+23;

      printf("Integer varijabla je %d\n", sum);
      printf("Znak je %c\n", letter);
      printf("Float varijabla je %f\n", set1);
      printf("Double varijabla je %e\n", num2);
}
```

Ispis programa

Integer varijabla je 100

Znak je Z

```
Float varijabla je 23.567000

Double varijabla je 11.000000e23
```

Da bismo promijenili broj decimalnih mjesta na koja se ispisuju float ili double varijable, mijenjamo %f ili %e tako da uvrstimo i preciznost, npr,

```
printf("Float varijabla je %.2f\n", set1 );
```

U ovom slučaju, uporaba %.2f ograničava ispis na dva decimalna mjesta, i sad on izgleda ovako

Ispis programa

```
Integer varijabla je 100
Znak je Z
Float varijabla je 23.56
Double varijabla je 11.000000e23
```

ZABILJEŠKA O KONVERZIJI TIPOVA PODATAKA

Pogledajmo slijedeći program,

```
#include <stdio.h>

main()
{
      int  value1 = 12, value2 = 5;
      float answer = 0;

      answer = value1 / value2;
      printf("Vrijednost %d podijeljena s %d iznosi %f\n",value1,value2,answer );

}
```

Ispis programa

```
Vrijednost 12 podijeljena s 5 iznosi 2.000000
```

Iako izgleda da je deklaracija u primjeru točna, nije uvijek 100% pouzdana. Zapazite kako answer ne sadrži točan decimalni dio (tj. sadrži sve nule).

Da osiguramo pojavljivanje točnog rezultata, tip podataka value1 i value2 trebao bi se konvertirati na float tip prije pridruživanja float varijabli answer. Pokazat ćemo kako to možemo učiniti,

```
answer = (float)value1 / (float)value2;
```

RAZLIČITI TIPOVI INTEGERA

Normalni integer je ograničen vrijednostima +-32767. Ova vrijednost se razlikuje od računala do računala. U C-u je moguće specificirati da integer pohranjujemo u četiri memorijske lokacije umjesto u uobičajene dvije. Ovo povećava raspon brojeva i omogućava pohranu vrlo velikih brojeva. To se radi na slijedeći način,

```
long int big_number = 245032L;
```

Za ispisati long integer, koristi se %l, npr,

```
printf("Veći broj je  %l\n", big_number );
```

Također imamo i short integer, npr,

```
short int   small_value = 114h;

printf("Vrijednost je %h\n", small_value);
```

Unsigned integer (samo pozitivne vrijednosti) također mogu biti definirani.

Veličina koju zauzima integer ovisi o samom računalu. ANSI C (American National Standards Institute) je pokušao standardizirati veličinu tipova podataka, i tako imamo sadašnje raspone.

Slijedeće informacije su iz Help-a Turbo C prevodioca,

Tip: int

Integer tip podatka

Varijable tipa int su duge jednu riječ.
Mogu biti signed (default) ili unsigned,
što znači da im je opseg vrijednosti od -32768 do
32767 , odnosno od 0 do 65535.

Modifikatori tipa: signed, unsigned, short, long

Modifikator tipa mijenja značenje osnovnog tipa
da bi se dobio novi tip. Svaki od navedenih modifikatora
može biti primijenjen na tip int. Modifikatori
signed and unsigned mogu se primijeniti na tip
char. K tome, long se može primijeniti
na double. Kada se osnovni tip ispusti iz deklaracije
pretpostavlja se int .

Primjeri:

```
long            x;  /* int sepretpostavlja*/

unsigned char     ch;

signed int        i;  /* signed je default */

unsigned long int l;  /* int se može staviti, ali nije potrebno */
```

NAREDBE PREPROCESORU

Izraz define se koristi za lakše čitanje programa. Pogledajmo slijedeće primjere,

```
#define TRUE    1    /* Nemojte koristiti točka-zarez , # mora biti prvi znak u liniji */
#define FALSE   0
#define NULL    0
#define AND     &
```

```
#define OR      |
#define EQUALS  ==

game_over = TRUE;
while( list_pointer != NULL )
        .....
```

Zapamtite da naredbe preprocesoru započinju sa simbolom # , i da NISU završene točka-zarezom. Uobičajeno je naredbe preprocesoru navoditi na početku koda.

Preprocesorske naredbe izvršava prevodioc (ili preprocesor) prije no što je program stvarno preveden. Svi # izrazi se izvrše tako da se simboli (poput TRUE) koji se pojavljuju u C programu zamjenjuju njihovim vrijednostima (poput 1). Kad preprocesor izvrši ovu zamjenu, program se prevodi.

Opaćnito se preprocesorske konstante pišu VELIKIM SLOVIMA.

ZAMJENA SIMBOLIČKIH KONSTANTI KORIŠTENJEM #define

Pogledajmo sada nekoliko primjera korištenja simbolićkih konstanti u našim programima. Slijedeći program definira konstantu imena POREZNA_STOPA.

```
#include <stdio.h>

#define POREZNA_STOPA  0.10

main()
{
        float glavnica;
        float porez;

        glavnica = 72.10;
        porez = glavnica * POREZNA_STOPA;
        printf("Porez na %.2f iznosi %.2f\n", glavnica, porez );
}
```

Preprocesor prvo zamjenjuje sve simbolieke konstante prije no što je program preveden, tako nakon preprocesora (i prije prevođenja), program izgleda ovako,

```
#include <stdio.h>

#define POREZNA_STOPA  0.10

main()
{
        float glavnica;
        float porez;

        glavnica = 72.10;
        porez = glavnica * 0.10;
        printf("Porez na %.2f iznosi %.2f\n", glavnica, porez );
}
```

NE MOŽETE PRIDJELJIVATI VRIJEDNOSTI SIMBOLIČKIM KONSTANTAMA
Uzimajući gornji program kao primjer, pogledajte promjene koje smo sada napravili. Dodali smo izraz koji nastoji promijeniti POREZNU_STOPU na novu vrijednost.

```
#include <stdio.h>

#define POREZNA_STOPA  0.10

main()
{
      float glavnica;
      float porez ;

      glavnica = 72.10;
      POREZNA_STOPA = 0.15;
      porez = glavnica * POREZNA_STOPA;
      printf("Porez na %.2f iznosi %.2f\n", glavnica, porez  );
}
```

Ovo je nedozvoljeno. Simboličkoj konstanti se ne može pridružiti nova vrijednost.

ZAMJENA SE VRŠI PISANJEM, PA PAZITE NA GREŠKE
Kao što je pokazano, preprocesor izvodi pisanu zamjenu simboličkih konstanti. Izmijenimo malo naš program i unesimo grešku da pojasnimo problem.

```
#include <stdio.h>

#define POREZNA_STOPA  0.10

main()
{
      float glavnica;
      float porez;

      glavnica = 72.10;
      porez = (glavnica * POREZNA_STOPA)+ 10.02;
      printf("Porez na %.2f iznosi %.2f\n", glavnica, porez );
}
```

U ovom slučaju, greška je u tomu što smo #define završili točka-zarezom. Preprocesor izvodi supstituciju i pogreška se javlja u liniji (označava je prevodioc)

```
      porez = (glavnica * 0.10; )+ 10.02;
```

Međutim, vi ne vidite izlaz preprocesora. Ako koristite TURBO C, samo vidite

```
      porez = (glavnica * POREZNA_STOPA)+ 10.02;
```

označeno kao grešku, a vama to izgleda u redu (ali nije! nakon zamjene).

LAKO MIJENJANJE PROGRAMA KORIŠTENJEM #define

Cijela svrha korištenja #define u vašim programima je da ih učinite lakšima za čitanje i mijenjanje. Koristeći gornje programe kao primjere, koje promjene bi trebalo napraviti ako se POREZNA_STOPA promijenila na 20%?

Očito, odgovor je da treba promijeniti #define naredbu koja deklarira simbolieku konstantu. Pisali bismo

```
#define POREZNA_STOPA = 0.20
```

Bez korištenja simboličkih konstanti, morali bismo unijeti vrijednost 0.20 u program, a ona se može pojavljivati nekoliko puta (ili nekoliko desetaka puta).

To bi mijenjanje programa učinilo teškim, zato što bi morali tražiti i zamijeniti svaku pojavu te varijable u programu. Međutim, kako programi postaju veći, što bi se dogodilo ako bi koristili vrijednost 0.20 u računu koji nema ništa s POREZNOM_STOPOM!

ZAKLJUČAK O #define

* omogućuje upotrebu simboličkih konstanti u programu
* uobičajeno se simboli pišu velikim slovima
* ne završava se točka-zarezom
* općenito se piše na početku koda
* svaki put kada se simbol pojavi, zamjenjuje se vrijednošću
* čini program lakim za čitanje i mijenjanje

PRIDRUŽIVANJE DATOTEKA PROGRAMU (HEADER FILES)

Header files su datoteke koje sadrže definicije funkcija i varijabli koje se mogu uključiti u bilo koji C program korištenjem preprocesorske direktive #include. Standardne datoteke dolaze sa svakim prevodiocem i pokrivaju široko područje, operacije sa stringovima, matematičke operacije, konverziju podataka, ispis i učitavanje varijabli.

Da bismo koristili neku od standardnih funkcija, moramo uključiti odgovarajući header file. To se čini na početku koda. Naprimjer, da bismo koristili funkciju printf() u programu, linija

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

bi trebala biti na početku programa, jer se definicija printf() može naći u file stdio.h Sve datoteke imaju ekstenziju .h i obično se nalaze u /include subdirektoriju.

```
#include <stdio.h>

#include "mydecls.h"
```

Korištenje zagrada <> informira prevodioca da traži u include direktoriju određenu datoteku. Korištenje nazivnika "" oko imena datoteke informira prevodioca da traži u trenutnom direktoriju određenu datoteku.

ARITMETIČKI OPERATORI

Simboli aritmetičkih operatora su:

Operacija	Operator	Opis	Vrijednost sum prije	Vrijednost sum kasnije
Množenje	*	sum = sum * 2;	4	8
Dijeljenje	/	sum = sum / 2;	4	2
Zbrajanje	+	sum = sum + 2;	4	6
Oduzimanje	-	sum = sum -2;	4	2
Inkrement	++	++sum;	4	5
Dekrement	--	--sum;	4	3
Modul	%	sum = sum % 3;	4	1

Slijedeći dio programa zbraja varijable loop i count , ostavljajući rezultat u varijabli sum

```
sum = loop + count;
```

Zapamtite: Ako znak modula % morate ispisati kao dio stringa, koristite dva znaka, tj. %%

```
#include <stdio.h>

main()
{
	int sum = 50;
	float modulus;

	modulus = sum % 10;
	printf(" %% od %d s 10 iznosi %f\n", sum, modulus);
}
```

Ispis programa

```
 % od 50 s 10 iznosi 0.000000
```

PREFIX/POSTFIX INKREMENT/DEKREMENT OPERATORI

PREFIX znači da se prvo obavi operacija inkrementa/dekrementa, a zatim nekakva operacija pridruživanja. POSTFIX znači da se operacija vrši nakon pridruživanja. Pogledajmo slijedeće izraze

```
++count;	/* PREFIX Inkrement, znači povećaj count za jedan*/

count++;	/* POSTFIX Inkrement, znači povećaj count za jedan */
```

U gornjem primjeru, zato što se vrijednost varijable count ne pridružuje nijednoj varijabli, učinak PREFIX/POSTFIX operacija nije jasno vidljiv.

Ispitajmo što se događa kada koristimo operator zajedno s operacijom pridruživanja. Pogledajmo slijedeći program,

```
#include <stdio.h>

main()
{
	int count = 0, loop;
	loop = ++count;  /* isto kao count = count + 1; loop = count;  */
	printf("loop = %d, count = %d\n", loop, count);

	loop = count++;  /* isto kao loop = count;  count = count + 1;  */
	printf("loop = %d, count = %d\n", loop, count);

}
```

Ispis programa

```
loop = 1, count = 1

loop = 1; count = 2
```

Ako operator prethodi (ako je s lijeve strane) varijabli, operacija se izvodi prva, tako izraz

```
loop = ++count;
```

ustvari znači inkrementiraj prvo count , a onda novu vrijednost varijable count pridruži varijabli loop.

Kako ćete pisati?
Tamo gdje se inkrement/dekrement operacija koristi za promjenu vrijednosti varijable, i nije uključena ni u kakvu operaciju pridruživanja, što ćete koristiti,

```
++loop_count;
```

ili

```
loop_count++;
```

Odgovor je da u stvari nije bitno. Čini se da većina C programera koristi postfix formu.

Na što morate pripaziti
Nemojte da vam uđe u naviku koristiti razmak(e) između imena varijable i prefix/postfix operatora.

```
loop_count ++;
```

Pokušajte povezati operator ne koristeći praznine.

DOBRA FORMA

Možda bismo trebali reći stil programiranja ili čitljivost. Najčešće pritužbe koje stižu na račun C programera početnika mogu se navesti kao ,

* programi nemaju formu
* programi se teško čitaju

Vaši programi će biti brži za pisanje i lakši za otkrivanje grešaka ako vam pređe u naviku dobro ih formirati dok ih pišete.

Naprimjer, pogledajmo donji program

```
#include<stdio.h>

main()
  {
   int sum,loop,kettle,job;
   char Whoknows;

       sum=9;
  loop=7;
 whoKnows='A';

printf("Whoknows=%c,Kettle=%d\n",whoknows,kettle);

}
```

Naša je namjera da program bude teško čitljiv, i zbog toga, neiskusnom programeru bit će teško i ispraviti greške. Program također sadrži nekoliko namjernih pogreški!

Stoga, napišimo program ponovno koristeći dobru formu.

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

```
main()
{

        int sum, loop, kettle, job;
        char whoknows;

        sum = 9;
        loop = 7;
        whoknows = 'A';

        printf( "Whoknows = %c, Kettle = %d\n", whoknows, kettle );

}
```

Također smo i ispravili pogreške. Najveće razlike su

* vitičaste zagrade su postavljene točno jedna ispod druge

* Ovo nam omogućava praćenje dijelova programa i osigurava da izrazi pripadaju točnim blokovima programa. Ovo je nužno kada su programi složeniji za lakšu čitljivost unosimo razmake

* Ljudi pišu rečenice koristeći razmake među riječima. To pomaže našem razumijevanju onoga što čitamo (ako ne vjerujete, pokušajte pročitati slijedeću rečenicu). kadbihimaodolarzasvakiputkadsamnapraviopogrešku. Unos razmaka također nam pomaže u bržem otkrivanju grešaka. dobro razmicanje

Nivoi razmicanja (tab) se koriste u blokovima programa, tako lako uočavamo funkcije, i koji izrazi pripadaju kojem dijelu programa { } .

UNOS S TIPKOVNICE

U C-u postoji funkcija koja omogućava programeru da prihvati unos s tipkovnice. Slijedeći program ilustrira upotrebu ove funkcije,

```
#include <stdio.h>

main()     /* program s primjerom unosa s tipkovnice */
{
        int  number;
        printf("Unesite broj \n");
        scanf("%d", &number);
        printf("Broj koji ste unijeli je %d\n", number);

}
```

Ispis programa

```
Unesite broj

23

Broj koji ste unijeli je 23
```

Definira se integer varijabla number . Zatim se ispisuje obavijest o unosu broja funkcijom

```
printf("Unesite broj \n:");
```

scanf funkcija, koja prihvaća odgovor, ima dva argumenta. Prvi ("%d") specifira koji tip podatka se očekuje (npr. char, int, ili float). Lista specifikatora formata za scanf .

Drugi argument (&number) specifira varijablu u koju će se smjestiti odgovor s tipkovnice. U ovom slučaju odgovor će biti smješten na memorijsku lokaciju pridruženu varijabli number.

Ovo objašnjava poseban značaj znaka & (koji znači adresa od).

Primjer programa koji pokazuje upotrebu scanf() za učitavanje integer, character i float varijabli

```
#include < stdio.h >

main()
{
        int sum;
        char letter;
        float money;

        printf("Molimo unesite integer ");
        scanf("%d", &sum );

        printf("Molimo unesite character ");
        /* prazno mjesto ispred znaka %c zanemaruje razmake u unosu */
        scanf("  %c", &letter );

        printf("Molimo unesite float varijablu ");
        scanf("%f", &money );

        printf("\nVarijable koje ste unijeli su\n");
        printf("vrijednost sum = %d\n", sum );
        printf("vrijednost letter = %c\n", letter );
        printf("vrijednost money = %f\n", money );

}
```

Ispis programa

Molimo unesite integer

34

Molimo unesite character

W

Molimo unesite float varijablu

32.3

Varijable koje ste unijeli su

vrijednost sum = 34

vrijednost letter = W

vrijednost money = 32.300000

Program pokazuje nekoliko važnih točaka.

* C jezik ne pruža nikakav oblik provjere unosa korisnika. Od korisnika se očekuje da unese točan tip podatka. Kao primjer, ako je korisnik unio character kada se očekivala integer vrijednost, program može ući u beskonačnu petlju ili se nenormalno prekinuti.
* na programeru je da ocijeni podatke za točnost tipa i njihov raspon vrijednosti.

RELACIJSKI OPERATORI
Omogućavaju usporedbu dviju ili više varijabli.

Operator	Značenje
==	jednako
!=	različito
<	manje od
<=	manje ili jednako
>	veće od
>=	veće ili jednako

Na slijedećih nekoliko stranica, ovi operatori će biti korišteni u for petljama i if izrazima.

Operator

```
<>
```

je dozvoljen u Pascalu, ali nije dozvoljen u C-u.

ITERACIJE, FOR PETLJA
Osnovni format for petlje je,

```
for( početni uvjet; uvjet nastavka; ponovna procjena )
        naredbe kontrolirane petljom;

/* primjer programa s for petljom */
#include <stdio.h>

main()   /* Program uvodi for petlju, broji do deset */
{
        int  count;

        for( count = 1; count <= 10; count = count + 1 )
                printf("%d ", count );

        printf("\n");
}
```

Ispis programa

```
1 2 3 4 5 6 7 8 9 10
```

Program deklarira integer varijablu count. Prvi dio izraza

```
for( count = 1;
```

inicijalizira vrijednost count na 1. For petlja se nastavlja dok je zadovoljen uvjet

```
count <= 10;
```

tj. dok je on istinit ili TRUE. Kako je varijabla count upravo inicijalizirana na 1, uvjet je TRUE pa se i naredba

```
printf("%d ", count );
```

vrši, ispisuje se vrijednost count na ekran, praćena razmakom.

Zatim se vrši slijedeća naredba u for petlji

```
count = count + 1 );
```

što dodaje jedan trenutnoj vrijednosti count. Vraćamo se opet na uvjet,

```
count <= 10;
```

koji je opet istinit pa se vrši naredba

```
printf("%d ", count );
```

Count se opet inkrementira, uvjet provjerava itd. sve dok se ne dođe do vrijednosti 11.

Kada se to dogodi, uvjet

```
count <= 10;
```

postaje FALSE, for petlja se završava i program prelazi na naredbu

```
printf("\n");
```

koja ispisuje novu liniju, i program završava jer više nema naredbi za izvršavanje.

```
/* primjer programa s  for petljom */
#include <stdio.h>

main()
{
      int  n, t_number;

      t_number = 0;
      for( n = 1; n <= 200; n = n + 1 )
            t_number = t_number + n;

      printf("Suma brojeva od 1 do 200 iznosi %d\n", t_number);
}
```

Ispis programa

```
Suma brojeva od 1 do 200 iznosi 20100
```

Gornji program koristi for petlju za računanje sume brojeva od 1 do 200 .

Slijedeći dijagram pokazuje kako se izvršavaju dijelovi for petlje.

Primjer korištenja for petlje za ispis znakova

```
#include <stdio.h>

main()
{
```

```
        char letter;
        for( letter = 'A'; letter <= 'E'; letter = letter + 1 ) {
                printf("%c ", letter);
        }
}
```

Ispis programa
A B C D E

Primjer korištenja for petlje za sumu brojeva, korištenjem dvije inicijalizacije

```
#include <stdio.h>

main()
{
        int total, loop;
        for( total = 0, loop = 1; loop <= 10; loop = loop + 1 ){
                total = total + loop;
        }
        printf("Total = %d\n", total );
}
```

Ispis programa

Total = 55

U gornjem primjeru, varijabla total se inicijalizira na 0 u prvom dijelu for petlje. Dva izraza,

```
for( total = 0, loop = 1;
```

su dio inicijalizacije. Ovo pokazuje da je dozvoljeno više izraza, ali moraju biti odvojeni zarezom.

WHILE PETLJA
Petlja while omogućava ponavljanje C naredbi dok je neki uvjet istinit. Njen format je,

```
while( uvjet )

        programske naredbe;
```

Negdje u tijelu while petlje mora postojati izraz koji mijenja vrijednost uvjeta tako da petlja može završiti.

```
/* Primjer programa s petljom while  */

#include <stdio.h>

main()
{
        int  loop = 0;

        while( loop <= 10 ) {
                printf("%d\n", loop);
                ++loop;
        }
}
```

Ispis programa

```
0

1

...

10
```

Gornji program koristi while petlju za ponavljanje naredbe

```
printf("%d\n", loop);
++loop;
```

dok je vrijednost varijable loop manja ili jednaka 10.

Zapamtite kako je varijabla o kojoj je petlja while ovisna inicijalizirana prije same while petlje (u ovom slučaju u prethodnoj liniji), i da se vrijednost varijable mijenja unutar petlje, tako da će jednom uvjet postati neistinit i while petlja će završiti.

Ovaj program je ustvari ekvivalentan prethodnom for programu koji je brojao do deset.

DO WHILE PETLJA

do { } while petla omogućava da se naredbe izvršavaju dok je uvjet istinit - TRUE (različit od nule). Petlja se izvršava barem jednom.

```
/* Demonstracija petlje DO...WHILE    */

#include <stdio.h>

main()
{
        int  value, r_digit;

        printf("Unesite broj koji će se ispisati od kraja.\n");
        scanf("%d", &value);
        do {
                r_digit = value % 10;
                printf("%d", r_digit);
                value = value / 10;
        } while( value != 0 );

        printf("\n");

}
```

Gornji program broj koji unese korisnik ispisuje od kraja. To čini koristeći ostatak dijeljenja (% operator) da bi krajnje desnu znamenku upisao u varijablu r_digit. Početni broj se zatim dijeli sa 10, i operacija se ponavlja dok broj ne postane jednak 0.

Naša je namjera pokazati da je ovakvo programiranje nepravilno i da bi se trebalo izbjegavati. Ima potencijalne probleme, kojih morate biti svjesni.

Jedan od takvih problema jest nedostatak kontrole. Pogledajmo slijedeći dio programa,

```
do {

    r_digit = value % 10;

    printf("%d", r_digit);

    value = value / 10;

} while( value != 0 );
```

NEMA nikakvog izbora hoće li se petlja izvršavati ili ne. Ulazak u petlju je automatski, jer je vaš jedini izbor nastavak.

Još jedan problem je što se petlja uvijek bar jednom izvrši. Ovo ide uz nedostatak kontrole. To znači da je moguće ući u do { } while petlju s nevaljanim podacima.

Programeri početnici lako mogu zapasti u brdo nevolja, pa je naša preporuka da ovu petlju izbjegavate koristiti. Ovo je jedini put da ćete je ovdje susresti. Uporabu je lako izbjeći korištenjem slijedećeg algoritma,

```
inicijaliziraj varijablu za kontrolu petlje

while( varijabla kontrole petlje je valjana ) {

    obrađuj podatke

    prilagodi kontrolnu varijablu ako je potrebno

}
```

Sada napišimo ponovno gornji program bez upotrebe do { } while petlje.

```
/* ponovljeni program bez do-while */

#include <stdio.h>

main()
{
    int  value, r_digit;

    value = 0;
    while( value <= 0 ) {
        printf("Unesite broj koji će se ispisati od kraja.\n");
        scanf("%d", &value);
        if( value <= 0 )
            printf("Broj mora biti pozitivan\n");
    }

    while( value != 0 )
    {
        r_digit = value % 10;
        printf("%d", r_digit);
        value = value / 10;
    }
    printf("\n");

}
```

Ispis programa

Unesite broj koji će se ispisati od kraja.

-43

Broj mora biti pozitivan

Unesite broj koji će se ispisati od kraja.

423

324

DONOŠENJE ODLUKA

SELEKCIJA (IF NAREDBA)
if naredba omogućava grananje (donošenje odluka) ovisno o vrijednosti ili stanju varijabli. To omogućava da se naredbe izvrše ili preskoče, ovisno o odluci. Osnovni format je,

```
if( izraz )

        programske naredbe;
```

Primjer;

```
if( studenti < 65 )

        ++student_count;
```

U gornjem primjeru, varijabla student_count se inkrementira samo ako je vrijednost varijable studenti manja od 65.

Slijedeći program koristi if naredbu da ocijeni je li korisnikov upisani broj između 1 i 10.

```
#include <stdio.h>

main()
{
        int number;
        int valid = 0;

        while( valid == 0 ) {
                printf("Unesite broj između 1 i 10 -->");
                scanf("%d", &number);
                /* pretpostavka da je broj valjan */
                valid = 1;
                if( number < 1 ) {
                        printf("Broj je manji od 1. Unesite ponovno\n");
                        valid = 0;
                }
                if( number > 10 ) {
                        printf("Broj je veći od  10. Unesite ponovno\n");
                        valid = 0;
                }
        }

        printf("Broj je %d\n", number );
```

```
}
```

Ispis programa

```
Unesite broj između 1 i 10 --> -78

Broj je manji od 1. Unesite ponovno

Unesite broj između 1 i 10 --> 4

Broj je 4
```

Pogledajmo slijedeći program koji utvrđuje je li znak unesen s tipkovnice u rasponu od A do Z.

```
#include <stdio.h>

main()
{
        char letter;
        printf("Unesite znak-->");
        scanf(" %c", &letter );

        if( letter >= 'A' ) {
                if( letter <= 'Z' )
                        printf("Znak je između A i Z\n");
        }
}
```

Ispis programa

```
Unesite znak --> C

Znak je između  A i Z
```

Program ne ispisuje ništa ako znak nije između A i Z. O ovomu ćemo govoriti na slijedećim stranicama kada budemo razmatrali if else naredbu.
Molimo zapamtite korištenje preznog mjesta u izrazu (ispred %c)

```
scanf(" %c", &letter );
```

Ovo omogućava preskakanje TABULATORA, razmaka i ENTER tipke. Ako se ne koristi vodeće prazno mjesto, koristio bi se prvi unešeni znak, i scanf ne bi ignorirala navedene znakove.

USPOREDBA JEDNAKOSTI float tipova
Zbog načina na koji se pohranjuju float tipovi, vrlo je teško usporediti jesu li jednaki. Stoga to izbjegavajte činiti, jer možete dobiti nepredviđene rezultate.

if else
Opći format ove naredbe je,

```
if( uvjet 1 )

  izraz1;

else if( uvjet 2 )

  izraz2;

else if( uvjet 3 )

  izraz3;

else

  izraz4;
```

else omogućava da se izvrši naredba kada je uvjet neistinit (jednak nuli).

Slijedeći program koristi if else naredbu da ocijeni je li upis korisnika između 1 i 10.

```
#include <stdio.h>

main()
{
        int number;
        int valid = 0;

        while( valid == 0 ) {
                printf("Unesite broj između 1 i 10 -->");
                scanf("%d", &number);
                if( number < 1 ) {
                        printf("Broj je manji od 1. Unesite ponovno\n");
                        valid = 0;
                }
                else if( number > 10 ) {
                        printf("Broj je veći od  10. Unesite ponovno\n");
                        valid = 0;
                }
                else
                        valid = 1;
        }
        printf("Broj je %d\n", number );
}
```

Ispis programa

```
Unesite broj između 1 i 10 --> 12

Broj je veći od 10. Unesite ponovno

Unesite broj između 1 i 10 --> 5

Broj je 5
```

Ovaj program se malo razlikuje od prethodnog primjera u tome da se else koristi za

postavljanje varijable valid na 1. U ovom programu lakše je pratiti logiku.

```
/* Ilustrira ugniježđene if else i više argumenata za scanf funkciju.  */

#include <stdio.h>

main()
{
      int    invalid_operator = 0;
      char   operator;
      float  number1, number2, result;

      printf("Unesite dva broja i operator\n");
      printf(" number1 operator number2\n");
      scanf("%f %c %f", &number1, &operator, &number2);

      if(operator == '*')
            result = number1 * number2;
      else if(operator == '/')
            result = number1 / number2;
      else if(operator == '+')
            result = number1 + number2;
      else if(operator == '-')
            result = number1 - number2;
      else
            invalid_operator = 1;

      if( invalid_operator != 1 )
            printf("%f %c %f iznosi %f\n", number1, operator, number2, result );
      else
            printf("Nedozvoljeni operator.\n");

}
```

Ispis programa

```
Unesite dva broja i operator
number1 operator number2

23.2 + 12

23.2 + 12 iznosi 35.2
```

Gornji program radi kao jednostavni kalkulator.

LOGIČKI OPERATORI U SLOŽENIM RELACIJAMA (AND, NOT, OR, EOR)
Uspoređivanje više od jednog uvjeta
Logički operatori omogućavaju uspoređivanje više od jednog uvjeta. Ti uvjeti su dio nekog izraza kojeg testiramo. Simboli ovih operatora su:

LOGIČKO I (AND) &&

Logičko I zahtijeva da svi uvjeti budu istiniti - TRUE (različiti od nule).

LOGIČKO ILI (OR) ||

Logičko ILI će se izvršiti ako je bilo koji (ili svi) od uvjeta istinit (različit od nule).

LOGIČKA NEGACIJA (NOT) !

Logička negacija negira uvjet (mijenja ga iz istinitog u neistinitog i obrnuto).

LOGIČKO EKSKLUZIVNO ILI (EOR) ^

Logičko ekskluzivno ILI će se izvršiti ako je jedan od uvjeta istinit, ali NE ako su svi istiniti.

Slijedeći program koristi if naredbu s logičkim I - AND da ocijeni je li korisnikov upis u rasponu između 1 i 10.

```
#include <stdio.h>

main()
{
	int number;
	int valid = 0;

	while( valid == 0 ) {
		printf("Unesite broj između 1 i 10 -->");
		scanf("%d", &number);
		if( (number < 1 ) || (number > 10) ){
			printf("Broj nije između 1 i 10. Unesite ponovno\n");
			valid = 0;
		}
		else
			valid = 1;
	}
	printf("Broj je %d\n", number );
}
```

Ispis programa

```
Unesite broj između 1 i 10 --> 56

Broj nije između 1 i 10. Unesite ponovno

Unesite broj između 1 i 10 --> 6

Broj je 6
```

Program se malo razlikuje od prethodnog primjera u tome što LOGIČKO I eliminira jedan od else izraza.

LOGIČKI OPERATORI U SLOŽENIM RELACIJAMA (AND, NOT, OR, EOR)
NEGACIJA

```
#include <stdio.h>

main()
{
	int flag = 0;
```

```
        if( ! flag ) {
                printf("Zastavica nije postavljena.\n");
                flag = ! flag;
        }
        printf("Vrijednost zastavice je %d\n", flag);
}
```

Ispis programa

Zastavica nije postavljena.

Vrijednost zastavice je 1

Program testira je li zastavica jednaka nuli, tj. nije (!) postavljena. Tada ispisuje određenu poruku, mijenja vrijednost zastavice (flag); flag postaje jednaka negaciji flag; tj. jednaka 1. Naposljetku se vrijednost flag ispisuje.

LOGIČKI OPERATORI U SLOŽENIM RELACIJAMA (AND, NOT, OR, EOR)
Provjeravanje raspona korištenjem logičkih operatora
Razmatrat ćemo slučaj kada korisnik upisuje neku vrijednost, a onda se provjerava pripada li ona zadanom rasponu, npr. između 1 i 100.

```
#include <stdio.h>

main()
{
        int number;
        int valid = 0;

        while( valid == 0 ) {
                printf("Unesite broj između 1 i 100");
                scanf("%d", &number );
                if( (number < 1) || (number > 100) )
                        printf("Broj je izvan zadanog raspona\n");
                else
                        valid = 1;
        }
        printf("Broj je %d\n", number );
}
```

Ispis programa

Unesite broj između 1 i 100

203

Broj je izvan zadanog raspona

Unesite broj između 1 i 100

-2

Broj je izvan zadanog raspona

Unesite broj između 1 i 100

37

Broj je 37

Program koristi varijablu valid kao zastavicu koja označava je li uneseni podatak unutar dozvoljenog raspona vrijednosti. While petlja se vrši sve dok je valid jednak 0.

Izraz

```
if( (number < 1) || (number > 100) )
```

provjerava je li korisnikov upis unutar granica zadanog raspona, a ako jest, postavlja valid na 1, omogućavajući izlaz iz petlje.

Sada pogledajmo program koji provjerava je li znak kojeg upisuje korisnik između A-Z, drugim riječima alfabetski.

```
#include <stdio.h>

main()
{
	char ch;
	int valid = 0;

	while( valid == 0 ) {
		printf("Unesite znak A-Z");
		scanf(" %c", &ch );
		if( (ch >= 'A') && (ch <= 'Z') )
			valid = 1;
		else
			printf("Znak je izvan dozvoljenog raspona\n");
	}
	printf("Znak je %c\n", ch );
}
```

Ispis programa

Unesite znak A-Z

a

Znak je izvan dozvoljenog raspona

Unesite znak A-Z

0

Znak je izvan dozvoljenog raspona

Unesite znak A-Z

R

Znak je R

U ovom slučaju, logičko I koristimo jer želimo ocijeniti je li znak unutar raspona, tj. sve vrijednosti između donje i gornje granice. U prethodnom slučaju, koristili smo logičko ILI jer smo željeli provjeriti je li unos iznad gornje granice ili ispod donje granice raspona.

switch() naredba:
Switch naredba je bolji način pisanja programa kada se pojavljuje više naredbi if else. Opći format je,

```
switch ( izraz ) {

    case  vrijednost1:

        programske naredbe;

        programske naredbe;

        ......

        break;

    case  vrijednostn:

        programske naredbe;

        .......

        break;

    default:

        .......

        .......

        break;

}
```

Ključna riječ break mora biti uključena na kraju svake alternative. Default označava alternativu koja će biti izabrana ako nijedna od prethodnih ne odgovara vrijednosti izraza. Default može biti i ispušten. Desna zagrada na kraju označava kraj izbora alternativa.

Pravila za switch naredbu

vrijednosti 'case' moraju biti integer ili character konstante

poredak 'case' izraza nije važan

default se može pojaviti kao prva alternativa (uobičajeno je posljednja)

ne mogu se koristiti rasponi ili izrazi kao vrijednosti 'case'

```
#include <stdio.h>

main()
{
    int menu, numb1, numb2, total;
```

```
        printf("unesite dva broja -->");
        scanf("%d %d", &numb1, &numb2 );
        printf("unesite izbor\n");
        printf("1=zbrajanje\n");
        printf("2=oduzimanje\n");
        scanf("%d", &menu );
        switch( menu ) {
                case 1: total = numb1 + numb2; break;
                case 2: total = numb1 - numb2; break;
                default: printf("Nedozvoljen izbor\n");
        }
        if( menu == 1 )
                printf("%d plus %d je %d\n", numb1, numb2, total );
        else if( menu == 2 )
                printf("%d minus %d je%d\n", numb1, numb2, total );
}
```

Ispis programa

```
unesite dva broja --> 37 23

unesite izbor

1=zbrajanje

2=oduzimanje

2

37 minus 23 je 14
```

Gornji program koristi switch da izabere između korisnikovog upisa simulirajući jednostavan izbor.

UPISIVANJE ZNAKA S TIPKOVNICE

getchar

Slijedeći program pokazuje uporabu getchar,

```
#include <stdio.h>

main()
{
        int  i;
        int ch;

        for( i = 1; i<= 5; ++i ) {
                ch = getchar();
                putchar(ch);
        }
}
```

Ispis programa

AACCddEEtt

Program učitava pet znakova (po jedan iz svakog ponavljanja for petlje) s tipkovnice. Zapamtite da getchar() prima jedan znak s tipkovnice, a putchar() piše jedan znak (u ovom slučaju, ch) na ekran.

Datoteka ctype.h osigurava funkcije za rad sa znakovima.

UGRAĐENE FUNKCIJE ZA RAD SA STRINGOVIMA
string.h
Možda prvo želite pogledati dio o nizovima !. Slijedeći makroi su uključeni u datoteku string.h

strcat	dodaje string
strchr	traži prvo pojavljivanje određenog znaka
strcmp	uspoređuje dva stringa
strcmpi	uspoređuje dva stringa bez obzira na mala i velika slova
strcpy	kopira jedan u drugi string
strlen	pronalazi dužinu stringa
strlwr	prebacuje string u mala slova
strncat	dodaje n znakova stringu
strncmp	uspoređuje n znakova dva stringa
strncpy	kopira n znakova iz jednog stringa u drugi
strnset	postavlja n znakova stringa u određeni znak
strrchr	pronalazi zadnje pojavljivanje nekog znaka u stringu
strrev	obrće string
strset	postavlja sve znakove u stringu na određeni znak
strspn	pronalazi prvi podstring iz skupa znakova u stringu
strupr	prebacuje string u velika slova

Prebacivanje stringa u velika slova

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

main()
{
      char name[80];  /* deklarira niz znakova 0-79 */
```

```
        printf("Upišite ime malim slovima\n");
        scanf( "%s", name );
        strupr( name );
        printf("Ime velikim slovima: %s", name );
    }
```

Ispis programa

Upišite ime malim slovima

samuel

Ime velikim slovima: SAMUEL

UGRAĐENE FUNKCIJE ZA RAD SA ZNAKOVIMA
Slijedeće funkcije za rad sa znakovima su definirane u datoteci ctype.h

isalnum	testira je li znak alfanumerički
isalpha	testira je li znak alfabetski
isascii	testira je li znak iz ASCII
iscntrl	testira je li znak kontrolni
isdigit	testira je li znak znamenka
isgraph	testira može li se znak ispisati
islower	testira je li znak malo slovo
isprint	testira može li se znak ispisati
ispunct	testira je li znak interpunkcija
isspace	testira je li znak razmak
isupper	testira je li znak veliko slovo
isxdigit	testira je li znak heksadecimalan
toascii	prebacuje znak u ASCII
tolower	prebacuje znak u mala slova
toupper	prebacuje znak u velika slova

Prebacivanje stringa u velika slova znak po znak korištenjem toupper()

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

main()
{
    char name[80];
    int loop;
```

```
        printf("Unesite ime malim slovima\n");
        scanf( "%s", name );
        for( loop = 0; name[loop] != 0; loop++ )
            name[loop] = toupper( name[loop] );

        printf("Ime velikim slovima: %s", name );

    }
```

Ispis programa

```
Unesite ime malim slovima

samuel

Ime velikim slovima: SAMUEL
```

Vrednovanje korisničkog upisa u C-u
Osnovna pravila

* Ne propuštajte naprijed nevaljane podatke.
* Provjeravajte podatke kada se upisuju.
* Uvijek pružite korisniku značajne informacije.
* Obavijestite korisnika kakav upis očekujete.

```
/* prvi primjer, jednostavan upit o nastavku */

#include <stdio.h>
#include <ctype.h>

main()
{
    int     valid_input;    /* kada je 1, podatak je valjan i petlja završava */
    char    user_input;     /* označava korisnikov upis, izbor jednog znaka */

    valid_input = 0;
    while( valid_input == 0 ) {
        printf("Nastavak (D/N)?\n");
        scanf("  %c", &user_input );
        user_input = toupper( user_input );
        if((user_input == 'D') || (user_input == 'N') )  valid_input = 1;
        else  printf("\007Greška.Nepostojeći izbor\n");
    }
}
```

Ispis programa

```
Nastavak (D/N)?

b
```

```
	Greška.Nepostojeći izbor

	Nastavak (D/N)?

	N
```

```
/* drugi primjer, primanje i provjeravanje izbora */

#include <stdio.h>
#include <ctype.h>

main()
{
	int	exit_flag = 0, valid_choice;
	char	menu_choice;

	while( exit_flag == 0 ) {
		valid_choice = 0;
		while( valid_choice == 0 ) {
			printf("\nC = Copy File\nE = Exit\nM = Move File\n");
			printf("Unesite izbor:\n");
			scanf("	%c", &menu_choice );
			if((menu_choice=='C') || (menu_choice=='E') || (menu_choice=='M'))
				valid_choice = 1;
			else
				printf("\007Greška.Nepostojeći izbor\n");
		}
		switch( menu_choice ) {
			case 'C' : .....();	break;
			case 'E' : exit_flag = 1;  break;
			case 'M' : .....();  break;
			default : printf("Greška---Ne bi se trebalo pojaviti.\n"); break;
		}
	}
}
```

```
	Ispis programa

	C = Copy File

	E = Exit

	M = Move File

	Unesite izbor:

	X

	Greška.Nepostojeći izbor

	C = Copy File

	E = Exit

	M = Move File

	Unesite izbor:
```

E

UVJETNI OPERATORI

Uvjetni izraz uzima TRI operatora. Dva simbola koja se koriste da bi se označio ovaj operator su upitnik (?) i dvotočka (:). Prvi operand se postavlja prije ?, drugi operand između ? i :, i treći nakon :. Opći format je,

```
uvjet ? izraz1 : izraz2
```

Ako je rezultat uvjeta TRUE (različit od nule), izraz1 se računa i rezultat računanja postaje rezultat operacije. Ako je uvjet FALSE (nula), onda se izraz2 računa i njegov rezultat postaje rezultatom operacije. Pojasnit ćemo primjerom,

```
s = ( x < 0 ) ? -1 : x * x;
```

Ako je x manji od nule onda je s = -1

Ako je x veći od nule onda je s = x * x

Primjer programa koji ilustrira uporabu uvjetnog izraza

```
#include <stdio.h>

main()
{
        int input;

        printf("Reći ću vam je li broj pozitivan, negativan ili nula!"\n");
        printf("molim sad unesite vaš broj--->");
        scanf("%d", &input );
        (input < 0) ? printf("negativan\n") : ((input > 0) ? printf("pozitivan\n") : printf("nula
\n"));
}
```

Ispis programa

```
Reći ću vam je li broj pozitivan, negativan ili nula!

molim sad unesite vaš broj---> 32

pozitivan
```

NIZOVI

Nizovi su složeni tip podataka koji se sastoji od više podataka istog tipa. Razmotrimo situaciju kada programer mora voditi evidenciju određenog broja ljudi unutar neke organizacije. Sve do sada, naš početni pokušaj bi bio stvoriti posebnu varijablu za svakog korisnika. To bi moglo ovako izgledati,

```
int name1 = 101;
```

```
int name2 = 232;

int name3 = 231;
```

Naravno da postaje sve teže voditi evidenciju kako raste broj varijabli. Nizovi nude rješenje ovog problema.

Niz je poput kutije s više elemenata, slično arhivu, koji koristi princip indeksiranja da bi pronašao svaku varijablu koja je u njemu pohranjena. U C-u, indeksiranje počinje od nule.

Nizovi, poput drugih varijabli u C-u, moraju biti deklarirani prije nego što se mogu koristiti.

Zamjena gornjeg primjera uz pomoć nizova izgleda ovako,

```
int names[4];

names[0] = 101;
names[1] = 232;
names[2] = 231;
names[3] = 0;
```

Stvorili smo niz imena names, koji može sadržavati četiri integer varijable. Također možete vidjeti da smo na posljednjem mjestu niza pohranili nulu. Ovo je česta tehnika koju koriste C programeri da bi označili kraj niza.

Nizovi imaju slijedeću sintaksu, koriste uglate zagrade da bi pristupili svakoj pohranjenoj vrijednosti (zovemo je element).

```
x[i]
```

Tako se x[5] odnosi na šesti element u nizu imena x. U C-u, elementi niza počinju od 0. Pridjeljivanje vrijednosti elementima niza se radi pomoću

```
x[10] = g;
```

a pridjeljivanje elemenata niza vrijednosti pomoću

```
g = x[10];
```

U slijedećem primjeru, deklariramo niz znakova word , i svakom elementu pridružujemo znak. Zadnji element niza je nula, da bi označili kraj stringa (u C-u, string nije tip podatka, tako se za rad sa stringovima koriste nizovi znakova). Printf funkcija se koristi za ispis elemenata niza.

```
/*  Predstavljamo nizove, 2  */

#include <stdio.h>

main()
{
      char word[20];

      word[0] = 'H';
      word[1] = 'e';
      word[2] = 'l';
      word[3] = 'l';
      word[4] = 'o';
      word[5] = 0;

      printf("Sadržaj word[] je -->%s\n", word );
```

```
}
```

Ispis programa

```
Sadržaj word[] je Hello
```

DEKLARIRANJE NIZOVA

Nizovi mogu sadržavati bilo koji dozvoljeni tip podataka. Nizovi se deklariraju zajedno sa svim drugim varijablama u dijelu programa za deklaraciju.

```
/*  Predstavljamo nizove  */

#include <stdio.h>

main()
{
        int   numbers[100];
        float  averages[20];

        numbers[2] = 10;
        --numbers[2];
        printf("Treći element niza je %d\n", numbers[2]);
}
```

Ispis programa

```
Treći element niza je 9
```

Gornji program deklarira dva niza, pridružuje 10 vrijednosti trećeg elementa niza numbers, dekrementira tu vrijednost (--numbers[2]), i naposljetku je ispisuje. Broj elemenata koji svaki od nizova može imati je upisana između uglatih zagrada.

DODJELJIVANJE POČETNIH VRIJEDNOSTI NIZOVIMA

Prije deklaracije pišemo riječ static. Početne vrijednosti se pišu u vitičastim zagradama,

```
#include <stdio.h>

main()
{
        int x;
        static int  values[] = { 1,2,3,4,5,6,7,8,9 };
        static char  word[] = { 'H','e','l','l','o' };
        for( x = 0; x < 9; ++x )
                printf("Values [%d] je %d\n", x, values[x]);
}
```

Ispis programa

Values[0] je 1

Values[1] je 2

....

Values[8] je 9

Prethodni program deklarira dva niza, values i word. Primijetite da između uglatih zagrada nema varijable koja označava koliko velik će niz biti. U ovom slučaju, C inicijalizira niz na broj elemenata koji se pojavljuje unutar vitičastih zagrada. Tako se values sastoji od 9 elemenata (označenih od 0 do 8), a niz znakova word ima 5 elemenata.

Slijedeći program pokazuje kako inicijalizirati sve elemente cjelobrojnog niza na vrijednost 10, koristeći for petlju da bi pristupili svakom elementu.

```
#include <stdio.h>

main()
{
        int count;
        int  values[100];
        for( count = 0; count < 100; count++ )
                values[count] = 10;
}
```

VIŠEDIMENZIONALNI NIZOVI
Višedimenzionalni nizovi imaju dvije ili više indeksnih vrijednosti koje određuju elemente niza.

multi[i][j]

U gornjem primjeru, prva indeksna vrijednost i određuje indeks retka, dok j određuje indeks stupca.

Deklaracija i računanje

```
int       m1[10][10];

static int m2[2][2] = { {0,1}, {2,3} };



sum = m1[i][j] + m2[k][l];
```

ZAPAMTITE način na koji se pridjeljuju početne vrijednosti dvodimenzionalnom nizu m2. Unutar zagrada nalaze se,

{ 0, 1 },

{ 2, 3 }

Zapamtite da se nizovi dijele na retke i stupce. Prvi je redak, drugi je stupac. Promatrajući početne vrijednosti pridružene m2, vidimo da su one,

m2[0][0] = 0

m2[0][1] = 1

m2[1][0] = 2

m2[1][1] = 3

NIZOVI ZNAKOVA [STRINGOVI]
Pogledajmo slijedeći program,

```
#include <stdio.h>

main()
{
        static char name1[] = {'H','e','l','l','o'};
        static char name2[] = "Hello";
        printf("%s\n", name1);
        printf("%s\n", name2);
}
```

Ispis programa

```
Helloxghifghjkloqw30-=kl``
Hello
```

Razlika između ova dva niza je u tome što name2 ima nulu postavljenu na kraj niza, tj. u name2[5], dok name1 nema. Ovo često može rezultirati u znakovima viška koji se ispisuju na kraju. Da postavimo nulu na kraj niza name1 , inicijalizaciju možemo promijeniti na,

```
static char name1[] = {'H','e','l','l','o','\0'};
```

Pogledajmo slijedeći program koji inicijalizira niz znakova word tijekom programa, koristeći funkciju strcpy, koja zahtijeva korištenje include datoteke string.h

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

main()
{
        char word[20];
        strcpy( word, "hi there." );
        printf("%s\n", word );
}
```

Ispis programa

hi there.

VARIJACIJE U DEKLARIRANJU NIZOVA

```
int  numbers[10];

static int numbers[10] = { 34, 27, 16 };

static int numbers[] = { 2, -3, 45, 79, -14, 5, 9, 28, -1, 0 };

static char text[] = "Welcome to New Zealand.";

static float radix[12] = { 134.362, 1913.248 };

double  radians[1000];
```

UČITAVANJE STRINGOVA S TIPKOVNICE

Nizovi znakova se često u C-u nazivaju stringovi. C ne podržava tip string, tako da se nizovi znakova koriste umjesto stringa. Modifikator %s u funkcijama printf() i scanf() se koristi za ispis i unos niza znakova. Ovo pretpostavlja da je nula pohranjena kao zadnji element niza. Promotrimo slijedeće izraze koji učitavaju string znakova (uključujući razmake) s tipkovnice.

```
char string[18];

scanf("%s", string);
```

ZAPAMTITE da znak & ne morate pisati ispred imena varijable kada se koristi modifikator % s ! Ako bi korisnikov upis bio

```
Hello<enter>
```

tada

```
string[0] = 'H'

string[1] = 'e'

....

string[4] = 'o'

string[5] = '\0'
```

Primijetite da tipku enter scanf() ne uzima kao string i da je string završen nulom ('\0') nakon posljednjeg znaka pohranjenog u nizu.

FUNKCIJE

Funkcija u C-u može izvoditi određenu zadaću i podupire koncept modularnog programiranja.

Već smo radili sa funkcijama. Tijelo C programa, identificirano ključnom riječju main, i zatvoreno između vitičastih zagrada je funkcija. Poziva je operacijski sustav kada se program

učitava, a kada je završena, vraća se operacijskom sustavu.

Funkcije imaju osnovnu strukturu. Njihov format je

```
tip_povratnog_podatka  ime_funkcije ( argumenti, argumenti )

deklaracija_tipa_podataka_argumenata;

{

    tijelo_funkcije

}
```

Vrijedno je napomenuti da se tip_povratnog_podatka pretpostavlja da je int ako nije ništa navedeno, tako da programi koje smo imali do sada impliciraju da main() vraća integer operacijskom sustavu.

ANSI C se lagano razlikuje u tomu kako se funkcije deklariraju. Njihov format je

```
tip_povratnog_podatka  ime_funkcije (tip_podatka ime_varijable, tip_podatka
ime_varijable, .. )

{

    tijelo_funkcije

}
```

Ovo omogućava provjeru tipa korištenjem funkcijskih prototipova da bi se kompajler obavijestio o tipu i broju parametara koje funkcija prima. Kada se funkcija poziva, ove informacije se koriste da se obavi provjera tipa i parametara.

ANSI C također zahtijeva da tip_povratnog_podatka za funkciju koja ništa ne vraća bude void. Default tip_povratnog_podatka se pretpostavlja integer ako nije naveden, ali se mora podudarati s tipom koji specificira deklaracija funkcije.

Jednostavna funkcija je,

```
void print_message( void )
{
    printf("Ovo je modul imena print_message.\n");
}
```

Primijetite da je ime funkcije print_message. Funkcija ne prima nikakve argumente što se označava ključnom riječju void u dijelu za parametre funkcijske deklaracije. Tip_povratnog_podatka je void, tako da funkcija ne vraća nikakve podatke.

ANSI C funkcijski prototip za print_message() je,

```
void print_message( void );
```

Prototipovi funkcija se navode na početku koda. Često mogu biti smješteni u korisničkim datotekama .h (header) .

FUNKCIJE
Uključimo sada ovu funkciju u program.

```
/* Program koji ilustrira jednostavni funkcijski poziv */
```

```
#include <stdio.h>

void print_message( void );    /* ANSI C prototip funkcije */
void print_message( void )     /* kod funkcije */
{
      printf("Ovo je modul imena print_message.\n");
}

main()
{
      print_message();
}
```

Ispis programa

Ovo je modul imena print_message.

Da bismo pozvali funkciju, dovoljno je napisati njeno ime. Kod povezan sa imenom funkcije se tada izvršava. Kada funkcija završi izvršavanje se nastavlja sa izrazom koji je prvi nakon imena funkcije.

U gornjem programu, izvršavanje počinje sa main(). Jedini izraz unutar glavnog tijela programa je poziv kodu funkcije print_message(). Ovaj kod se izvršava, i kada je završen vraćamo se u main().

Kako program nema više naredbi, završava vraćanjem u operacijski sustav.

FUNKCIJE

U slijedećem primjeru, funkcija prima jednu varijablu, ali ne vraća nikakvu informaciju.

```
/* Program za računanje faktorijela  */

#include <stdio.h>

void calc_factorial( int );    /* ANSI prototip */
void calc_factorial( int n )
{
      int  i, factorial_number = 1;

      for( i = 1; i <= n; ++i )
            factorial_number *= i;
      printf("Faktorijela od %d je %d\n", n, factorial_number );
}

main()
{
      int  number = 0;

      printf("Unesite broj\n");
      scanf("%d", &number );
      calc_factorial( number );
}
```

Ispis programa

Unesite broj

3

Faktorijela od 3 je 6

Pogledajmo funkciju calc_factorial(). Deklaracija funkcije

```
void calc_factorial( int n )
```

označava da nema povratne vrijednosti i da funkcija prima jedan integer, unutar tijela funkcije poznat kao n. Zatim dolazi deklaracija lokalnih varijabli,

```
int  i, factorial_number = 0;
```

U C-u je pravilnije pisati,

```
auto int  i, factorial_number = 0;
```

jer ključna riječ auto označava prevodiocu da su varijable lokalne. Program radi prihvaćajući varijablu s tipkovnice i predajući je funkciji. Drugim riječima, varijabla number unutar main tijela se kopira u varijablu n u funkciji, koja tada računa točno rješenje.

VRAĆANJE REZULTATA FUNKCIJE

To se postiže ključnom riječju return koju prati varijabla ili konstantna vrijednost čiji tip podatka se mora podudarati sa deklariranim tipom_povratnog_podatka za funkciju.

```
float add_numbers( float n1, float n2 )

{

      return n1 + n2;    /* dozvoljeno*/

      return 6;          /* nedozvoljeno, nije isti tip */

      return 6.0;        /* dozvoljeno */

}
```

Moguće je da funkcija ima više izraza return.

```
int validate_input( char command )
{
      switch( command ) {
            case '+' :
            case '-' : return 1;
            case '*' :
            case '/' : return 2;
            default  : return 0;
      }
}
```

Evo još jednog primjera

```
#include <stdio.h>

int calc_result( int, int );          /* ANSI prototip */

int calc_result( int numb1, int numb2 )
{
      auto int result;
      result = numb1 * numb2;
      return result;
}

main()
{
      int  digit1 = 10, digit2 = 30, answer = 0;
      answer = calc_result( digit1, digit2 );
      printf("%d pomnoženo sa %d je %d\n", digit1, digit2, answer );
}
```

Ispis programa

10 pomnoženo sa 30 je 300

ZAPAMTITE da vrijednost koju vraća funkcija (rezultat) mora biti deklarirana u funkciji.

ZAPAMTITE: Formalna deklaracija imena funkcije mora biti nakon tipa podatka koji funkcija vraća,

```
int  calc_result ( numb1, numb2 )
```

LOKALNE I GLOBALNE VARIJABLE

Lokalne varijable

Ove varijable postoje samo unutar određene funkcije koja ih kreira. Nepoznate su drugim funkcijama i glavnom programu. Kao takve, uobičajeno se implementiraju koristeći stog (stack). Lokalne varijable prestaju postojati kada se izvrši funkcija koja ih je kreirala. Ponovno se kreiraju svaki put kada se funkcija poziva ili izvršava.

Globalne varijable

Ovim varijablama može pristupiti svaka funkcija iz programa. Implementiraju se asociranjem memorije imenima varijabli. Ne kreiraju se ponovno ako se funkcija ponovno poziva.

DEFINIRANJE GLOBALNIH VARIJABLI

```
/* Demonstriranje globalnih varijabli  */

#include <stdio.h>

int add_numbers( void );              /* ANSI prototip funkcije */

/* Ovo su globalne varijable i može im pristupiti svaka funkcija */

int  value1, value2, value3;

int add_numbers( void )
{
      auto int result;
```

```
        result = value1 + value2 + value3;
        return result;
}

main()
{
        auto int result;
        value1 = 10;
        value2 = 20;
        value3 = 30;
        result = add_numbers();
        printf("Suma %d + %d + %d iznosi %d\n",
                value1, value2, value3, final_result);
}
```

Ispis programa

Suma10 + 20 + 30 iznosi 60

Vidljivost globalnih varijabli se može ograničiti pažljivim odabirom mjesta deklaracije.
Globalne varijable su vidljive od deklaracije do kraja koda.

```
#include <stdio.h>

void no_access( void ); /* ANSI prototip funkcije */

void all_access( void );


static int n2;          /* n2 se vidi od ove točke nadalje */


void no_access( void )
{
        n1 = 10;        /* nedozvoljeno, n1 još nije poznat */
        n2 = 5;         /* dozvoljeno */
}

static int n1;          /* n1 se vidi od ove točke nadalje */

void all_access( void )
{
        n1 = 10;        /* dozvoljeno */
        n2 = 3;         /* dozvoljeno */
}
```

AUTOMATSKE I STATIČKE VARIJABLE
C programi imaju više područja (dijelova) gdje su smješteni podaci. Ti segmenti su tipično,

_DATA Statički podaci

_BSS Neinicijalizirani statički podaci, postavljeni na nulu prije poziva main()

_STACK Automatski podaci, smješteni na programskom stogu, lokalne varijable

_CONST konstantni podaci, korištenjem ANSI C ključne riječi const

Korištenje prikladne ključne riječi omogućava pravilno smještanje varijable u željeni segment

podataka.

```
/* primjer programa koji pokazuje razliku između statičkih i automatskih varijabli */

#include <stdio.h>

void demo( void );              /* ANSI prototip funkcije */

void demo( void )
{
        auto int avar = 0;
        static int svar = 0;

        printf("auto = %d, static = %d\n", avar, svar);
        ++avar;
        ++svar;
}

main()
{
        int i;

        while( i < 3 ) {
                demo();
                i++;
        }
}
```

Ispis programa

```
auto = 0, static = 0

auto = 0, static = 1

auto = 0, static = 2
```

Ispis programa

Statičke varijable se stvaraju i inicijaliziraju jednom, pri prvom pozivu funkcije. Slijedeći pozivi funkcije ne stvaraju ih i ne inicijaliziraju ih ponovno. Kada funkcija završava, varijable još uvijek postoje u _DATA segmentu, ali im ne mogu pristupiti vanjske funkcije.

Automatske varijable su suprotne. Stvaraju se i inicijaliziraju ponovno svaki put kada se funkcija pozove. Nestaju (dealociraju se) kada funkcija završava. Kreiraju se u _STACK segmentu.