

Programski jezik C za apsolutne pocetnike

Zasto uopste uciti C u 2020 godini?

Iako ljudi smatraju C programski jezik (u daljnjem tekstu samo C) komplikovanim za učenje i upotrebu, uskoro cete vidjeti da to ipak nije tako. Ako se pravilno pristupi učenju i primjeni ovog programskog jezika, shvaticete da je C neophodan za upoznavanje nacina na koji kompjuterski programi rade. Strucnjaci koji poznaju C su zaista trazeno zanimanje a samim učenjem C jezika stvarate odlicnu osnovu za postizanje vjestina koje ce vam otvoriti vrata za lakse učenje razvoja aplikacija, programiranja igrica ali i mnogih drugih programskih jezika.

Ako nikad u zivotu niste napisali liniju koda onda je ovaj kurs definitivno za vas, jer cu vas nauciti osnovnim konceptima programiranja, objasniti sta je program i shvaticete koliko je C ustvari mocan i lijep programski jezik.

Dakle nema brige, na vama je samo da imate volju za učenjem C programskog jezika a ja cu da vam predstavim ovo na najlaksi moguci nacin. Ukoliko budete pratili ovaj kurs i odradite sve vježbe, na odlicnom putu ste da postanete programer.

Vecina novih programskih jezika se koncentrise na produktivnost i pokusava da sto vise pojednostavi ... (ja ih ponekad zovem no brainer) (tbd.)

Sta je program?

Vjerovali ili ne, kompjuter kao masina je nista, uopste nije pametan, cak i u momentima kad je vas nivo inspiracije na minimumu, opet ste pametniji od kompjutera. Kompjuteri za razliku od ljudi, ne mogu samostalno da razmisljaju, odlucuju bez programera (ljudi koji "govore" kompjuterima sta da rade), koji daju detaljne instrukcije u vidu programskog koda. Bez ovih instrukcija kompjuteri su beskorisna hrpa metala i plastike. Sto su programeri inspirativniji, njihovi programi su napredniji, a samim tim kompjutere cine "pametnijim".

Dakle program predstavlja skup precizno i detaljno definisanih instrukcija koje govore kompjuteru sta da radi, opisuju kako da se ponasa.

Iako danas mozemo pronaci mnogo razlicitih programa na trzistu koji ispunjavaju vecinu nasih prohtjeva, cesto je potrebno napraviti program od nule, na zahtjev neke firme ili klijenta i tada nastupaju programeri i developeri sa svojim vjestinama da bi rijesili i obezbijedili ove specificne zahtjeve. Nakon sto naucite programski jezik C, bicete u mogucnosti da napisete program koji sadrzi instrukcije koje ce vas kompjuter da slijedi i cime cete odrediti njegovo ponasanje.

Priprema radnog okruzenja

Da bismo napisali instrukcija u C programskom jeziku moramo imati nakakav tekst editor i specijalan compiler koji služi za prevodjenje naseg izvornog koda u kod razumljiv kompjuteru. C compiler uzima nas C izvorni kod i vrsi operaciju build-anja ili compile-ranja (tehnicki naziv kako napraviti program razumljiv kompjuteru), koji nam omogucava da pokrenemo program na nasem kompjuteru. Iako ima dosta razlicitih pristupa, za potrebe ovog kursa mi cemo koristiti Code::Blocks (kompletno IDE okruzenje), jer se moze koristiti na sve tri platforme (Linux, MacOS i Windows), naravno vi ste slobodni da potrazite i druga radna okruzenja ali za da budemo potpuno sigurni da mozete pratiti kurs najbolje je da instalirate Code::Blocks kako je navedeno u nastavku.

Takodje na Windows masinama je potrebno instalirati i C compiler, najbolje prije instalacije Code::Blocks-a, jer ne dolazi preinstaliran sa operativnim sistemom, a neophodan je za prevodjenje izvornog koda u kod razumljiv kompjuteru. Za tu svrhu cemo koristiti MinGW, cija instalacija je objesnjena u sledececih nekoliko koraka.

MinGW instalacija (tbd.)

Code::Blocks instalacija (tbd.)

Proces/koraci prilikom programiranja

1. Odlucite sta vas program treba da uradi
2. Koristite "editor" da napisete i sacuvate vas kod
3. Compile program
4. Provjerite ako ima greski, popravite i ponovite korak 3.
5. Izvrsite vas program

Zdravo Svijete, vas prvi C program

Za pocetak se opustite i nemojte pokusavati da razumijete svaku liniju koda, ionako cu vam objasniti sve u detalje, a kroz sam kurs vise puta cemo se vratiti nekim osnovama kako bi vase znanje postalo temeljno i kako biste stekli samopouzdanje. Idemo se upoznati sa C-om i programskim okruzenjem Code::Blocks-a.

koraci kompajliranja i pokretanja u Code::Blocks (tbd.)

Izvorni kod: kod-10_zdravo-svijete.c

```
#include<stdio.h>

int main(){

    printf("Zdravo Svijete");

    return 0;

}
```

Rezultat

Zdravo Svijete

Napomena Vodite racuna prilikom copy/paste koda iz drugih editora kao recimo MS Word (tbd.)

Par pravila prilikom pisanja koda u C programskom jeziku

Komande i funkcije (naucicete funkcije u sledecoj sekciji) u C-u se pisu malim slovima, jedina situacija kada koristimo velika slova jeste sa direktivom **#define** i unutar stampanja poruka sa funkcijom **printf** i drugih funkcija za stmapanje. U slucaju da funkcije ili komande imaju vise rijeci u svom nazivu, tada je pravilo da se koristi veliko slovo na pocetku svake sledece rijeci u nazivu (**stampajIzvestaj()**, **mojaPrvaFunkcija()**), kako bi se povecala citljivost (razmaci u nazivima funkcija nisu dozvoljeni). Ovaj nacin zapisavanje se jos naziva camelCase. Najlaksi nacin da zapamtite jeste da imate na umu nazive iPhone ili eBay.

Svaka komanda ili funkcija u C-u mora imati tacku-zarez ; na kraju linije koda, nakon cega C zna da je linija zavrшена i pocinje nova. Tacka-zarez se ne stavlja nakon uglastih linija niti prve linije funkcije, zato sto se nista ne desava na ovim linijama.

Funkcija main()

Najvazniji dio C programa je upravo funkcija main(). Dakle, main() je funkcija u C programskom jeziku, ne komanda, zapmtite.

Funkcije nisu nista drugo, nego unaprijed definisan rutine za izvršavanje zadatka. Neke funkcije dolaze sa programskim jezikom C, dok ostale mozemo kreirati sami.

C program se sastoji od vise funkcija, ali svaki C program mora ukljucivati `main()` funkciju.

Kako da razlikujemo funkcije od komandi? Pa veoma lako, funkcije za razliku od komandi imaju otvorenu i zatvorenu zagradu `()` odmah nakon naziva funkcije.

Funkcije:

`main() printf() scanf() strlen()`

Komande:

`return int float while if for`

Dosta puta cete u C literaturi na objasnjenje funkcije kao recimo **`printf`**, gdje autor ne navodi zagrade, a trebao bi (**`printf()`**) bi bilo ispravno), mada je ovo nepisano pravilo i ustaljen nacin.

Mandatorna funkcija `main()`, je funkcija koju kompjuter prvu trazi i izcitava prilikom kompajliranja vases C koda. Cak iako se funkcija `main()` ne nalazi, po redosljedu pisanja na prvom mjestu u vasem kodu, ona i dalje odredjuje gdje vas progam pocinje. Zato je zbog citljivosti i preglednosti koda preporucljivo postaviti funkciju `main()` na prvo mjesto nakon `include` direktiva. U narednim lekcijama nasi programi ce imati samo jednu, jedinu funkciju `main()`, ali kako budemo napradovali i ucili nove stvari, shvaticete zasto se uvodjenjem dodatnih funkcija povecava produktivnost i efikasnost vasis programa.

Nakon funkcije `main()`, odnosno zagrada `()`, dolazi novi par viticastih zagrada `{}`, koje cine tijelo funkcije `main()`.

```
int main(){  
  
}
```

`int` nam govori da je funkcija `main()` tipa integer (cjelobrojna), o cemu ce detaljnije biti rijeci u sledecem poglavlju.

Direktiva **`#include <stdio.h>`** je neophodna u 99% posto slucajeva prilikom rada sa C programima i pomaze na da prihvatimo ili ispisemo informacije na ekranu, cime predstavlja neizostavni dio svakog naseg C programa u ovom kursu.

(tbd. pokazati funkciju `printf()` u `c:/mingw/include`)

Tipovi podatka

C koristi podatke kao sto su brojevi, karakteri/znakovi i rijeci, od kojih kreira upotrebljive informacije.

Iako imamo dosta razlicitih tipova podataka ovo su neki od najosnovnijih i najvise u upotrebi kada je u pitanju C programski jezik:

- Karakteri (Characters)
- Cijeli brojevi, cjelobrojni (Integer)
- Brojevi sa pokretnim zarezom ili realni brojevi (Floating points)

Znam sad da mislite da je za programiranje potrebno znanje matematike i kvantne fizike, ja cu vam tokom kursa pokazati da nije. Sto se tice matematike dovoljno je da znate sabrati $2 + 2$, sto nije uvijek 4 😊 i da nevjerujete u crne rupe, vanzemaljce i sta ti ja znam kad je fizika u pitanju.

Karakter u C-u

U C programskom jeziku, karakter predstavlja svaki pojedinačni znak koji se može predstaviti u kompjuteru. Kompjuter poznaje 256 različitih karaktera, koje možete pronaći u ASCII tabeli (tbd. skinuti ASCII tableu ili linkovati), koju naravno nema potrebe da pokušavate učiti napamet.

Primjeri karaktera

A b 1 7 % \$? W + - { } .

Čak i razmak **** **** predstavlja karakter, a kao što možete vidjeti svako slovo, broj, specijalni znak predstavlja karakter u C-u.

Iako 1 ili 7 izgledaju kao brojevi, što stvarno jesu, ako smo ih definisali kao karakter tip podatka, mi više ne možemo vršiti osnovne matematičke operacije na njima. Naravno ako definisemo 1 ili 7 kao cjelobrojni tip podatka mi nad njima možemo vršiti matematičke operacije. Ista je stvar sa specijalnim znakovima kao što su **+ - = / %**.

Da bi smo definisali tip podatka karakter u C-u, potrebno je upotrijebiti apostrof, dakle otvorimo apostrof umetnemo karakter zatvorimo apostrof.

'A' 'b' '1' '7' '-' '.'

Naravno ako to nismo uradili onda ovi znakovi ne predstavljaju karaktere

**** A b 1 7 - ****

Ako recimo napisemo

'C je jednostavan za učenje'

napravili smo grešku jer ovo ne može biti karakter, zato što ima više od jednog znaka unutar apostrofa.

U slučaju da želimo definisati više od jednog karaktera, moramo koristiti navodnike **"**. Na ovaj način možemo grupisati više od jednog karaktera a ovakva struktura se naziva jednom riječju **string** i ovo bi bio ispravan način definisanja string-a, za razliku od prethodnog:

"C je jednostavan za učenje".

Ovo bi za sad bilo sve sto vam je potrebno, a u jednom od narednih poglavlja cemo uciti kako se karakteri i stringovi dodjeljuju promjenjivima, prilikom cega ce vam ovo znanje biti potrebno.

Brojevi u C-u

U C-u (generealno u programiranju), brojevi dolaze u razlicitim oblicima i velicinama. Vas C program mora imati opciju da sacuva brojeve, bez obzira kako oni izgledali i koje su velicine.

Cijeli brojevi (integers)

Cijeli brojevi se nazivaju integers. Cijeli brojevi nemaju zarez, dakle bilo koji broj koji nema decimalni zarez je integer. Ovo su primjeri cjelih brojeva

100 300 23 0 -200 -10.

Nikad nemojte pisati integer **27** kao **027**, jer ako stavite **nulu** ispred vases broja, C ce pomisliti da ste pokusali definisati broj u **oktalnom** brojevnom sistemu ili ne daj Boze da dodate **x** nakon nule **0x27** u tom slucaju C ce misliti da zelite definisati broj u heksadecimalnom brojevnom sistemu. Dakle, drzite se naucenog iz realnog zivota 😊.

Brojevi sa pokretnim zarezom (float)

Brojevi sa pokretnim zarezom ili decimalni brojevi, koji se nazivaju **float**, su svi brojevi koji u svojoj reprezentaciji imaju decimalni zaraz, koji se u C-u predstavlja tackom .

Primjeri decimalnih brojeva

3.14 33.4 0.0 1.0 -200.3 .33 -.13

U slucaju decimalnih brojeva, nula na pocetku je dozvoljena, naravno sve dok broj sadrzi tacku.

Koju vrstu brojeva koristiti prilikom implementacije vases programa, ostaje na vama kao programeru da odlucite. Obicno se integer-i primjenjuju kada zelimo definisati vrijednosti kao sto su godine ili kolicinia, dok se brojevi sa pokretnim zarezom, ili float, primjenjuju kada radimo neka preciznija izracunavanja. Float zauzima vise mjesta u memoriji racunara, tako da je potrebno voditi racuna i teziti primjeni integer-a kada god je to moguće. Razliciti compiler-i imaju razlicite vrijednosti za velicine integer-a i float-a u memoriji, ali ovo cemo raditi prakticno uskoro, gdje cete nauciti kako se dobijaju ovakve vrijednosti.

Hajde da vidimo prakticno kako se nasi tipovi podataka ponasaju unutar koda, ali i da se blize upoznamo sa C programskom strukturom i samom main() funkcijom. Iako ce svi ovi pojmovi vise puta biti objasnjeni do kraja kursa, bitno je da ih stalno ponavljamo, eksperimentisemo i na taj nacin sticemo samopuzdanje u sebe i svoje vjestine programiranja.

Izvorni kod: kod-11_rad-sa-tipovima-podataka.c

```
#include<stdio.h>

int main(){
    printf("Goku je poceo da uci C programski jezik.");
}
```

```
printf("Upravo je završio poglavlje 1.");  
printf("Goku ima 14.5 godina.");  
printf("Goku voli programiranje.");  
  
return 0;  
}
```

Rezultat

Goku je počeo da uči C programski jezik. Upravo je završio poglavlje 1. Goku ima 14.5 godina. Goku voli programiranje.

Izvorni kod: kod-12_rad-sa-tipovima-podataka.c

```
#include<stdio.h>  
  
int main(){  
    printf("Goku je počeo da uči C programski jezik. \n");  
    printf("Upravo je završio poglavlje 1. \n");  
    printf("Goku ima 14.5 godina. \n");  
    printf("Goku voli programiranje. \n");  
  
    return 0;  
}
```

Rezultat

Goku je počeo da uči C programski jezik.
Upravo je završio poglavlje 1.
Goku ima 14.5 godina.
Goku voli programiranje.

Izvorni kod: kod-13_rad-sa-tipovima-podataka.c

```
#include<stdio.h>  
  
int main(){  
    printf("Goku je počeo da uči %c programski jezik \n", 'C');  
    printf("Upravo je završio poglavlje %d \n", 1);  
    printf("Goku ima %.1f godina \n", 14.5);  
    printf("Goku voli programiranje.");  
}
```



```
    return 0;  
}
```

Rezultat

```
Goku je poceo da uci C programski jezik.  
Upravo je zavrrio poglavlje 1.  
Goku ima 14.5 godina.  
Goku voli programiranje.
```

%c predstavlja mjesto gdje cemo ubaciti **karakter veliko C** i služi nam za reprezentaciju karaktera (skraćeno od character). %c je fiksno dok može da poprimi bilo koji karakter, sve dok imamo jedan karakter. Sta bi bilo ako umjesto 'C' dodijelimo 'Java'?

%d predstavlja mjesto gdje cemo ubaciti **broj 1** i služi nam za reprezentaciju cijelih brojeva (skraćeno od digit). %d je fiksno i može poprimiti vrijednost bilo kog cijelog broja. Sta bi bilo ako umjesto 1 dodijelimo neko slovo ili specijalan karakter, odnosno bilo koju vrijednost koja nije cio broj.?

%f predstavlja mjesto gdje cemo ubaciti **broj 14.5** i služi nam za reprezentaciju realnih ili brojeva sa pokretnim zarezom float (skraćeno od float). %f je fiksno i može poprimiti bilo koji broj iz opsega realnih brojeva. Sta biti ako umjesto realnog broja dodijelimo neki cio broj ili neko slovo?

Vježbajte na način da eksperimentisete. Izostavite neke dijelove izvornog koda, sacuvajte i pokušajte da kompajlirate (recimo izostavite tacku-zarec, kakav ce biti efekat?). Ucite na greskama. Svakako ce uskoro biti rijeci o takozvanim karakterima za konverziju.

Komentari prilikom programiranja u C-u

Kako smo vec naucili, C je precizan programski jezik i zahtijeva od nas kao programera da budemo veoma precizni prilikom naseg koda ako zelimo da se ponasa prema nasim zahtjevima. Ali nesmiijemo zaboraviti ni druge ljude prilikom pisanja naseg koda. Kao programer, cesto cete mijenjati programe, firme i bitno je da sve sto ostavite iza sebe u vidu napisanog koda bude citljivo i onima koji dolaze ili preuzimaju vas kod. Neke kompanije insistiraju da se na pocetku, u zaglavlju vaseg koda, ostave podaci o autoru, mail adresa pa cak i telefon u rijetkim situacijama, u slucaju da kasnije treba da vas kontaktiraju. Naravno, u danasnje vrijeme sa sve vecom upotrebom alata za verzionisanje koda, poput Git-a, ovaj pristup je sve redji u praksi. Dobro je ostaviti i naziv samog fajla u komentaru.

Komentari su dio koda koji compiler ignorise u potpunosti i kao takvi sluze da se prenese poruka ili da upustvo kako se neki dio koda koristi, sta neka funkcija radi i na koji nacin, zasto je nesto uradjeno kako je uradjeno.

Pisite komentare dok pisete vas program, cak i ako niko nece koristiti vas kod osim vas, ipak pisite kad god smatrate da postoji potreba za tim i nemojte cekati kasnije pisite ih zajedno sa vasim kodom i dajte im isti

znacaj kao i samom kodu. Komentarisanjem cete steci odlicnu naviku ali i obezbijediti citljivost i lakse odrzavanje vasesg koda kada vremenom naraste na nekoliko stotina ili hiljada linija.

Iako jos nismo dosli do ovog stepena, nije mi cilj da vas zbunjujem, prvenstveno hocu da pokazem koliko su komentari bitni, uzmimo za primjer:

```
return ((a < b) ? a : b);
```

Siguran sam da vam ovo ne znaci puno i trebalo bi vam vremena da shvatite sta prethodna linija koda radi. Ali ako to napisemo na nacin

```
return ((a < b) ? a : b); /* Vrati manji od zadana dva broja. */
```

Iz ovog vidite koliki je znacaj komentara. Pa cak i da vam nije jasno kako ovo radi imacete polaznu tacku, za dalju pretragu na internetu ili cete moci da se obratite nekom od kolega sa razumljivim pitanjem.

Naravno nemojte pisati komentare kada to nije potrebno

```
printf("Zdravo Svijete"); /* Ispisi poruku "Zdravo svijete" */
```

Mislim da ovo do sad i ptice na grani znaju 😊.

Komentare mozemo pisati na dva nacina:

1. kao jednolinijski komentar:

```
/* komentar u jednoj liniji */
```

ili

```
// komentar u jednoj liniji
```

2. kao viselinijski komentar:

```
/* komentar u  
vise linija */
```

Razmaci i prazna polja (Whitespace)

Kolekcije razmaka i praznih polja se jednom rijecju nazivaju Whitespace i veoma su vazan segment C programiranja, kada je u pitanju citljivost koda. Razmotrimo sledeci primjer izvornog koda u C-u:

Izvorni kod: kod-13_rad-sa-tipovima-podataka.c

```
#include <stdio.h>
main(){float a; float b; printf("Koliko imate kilograma?"
);scanf(" %f",&a); b = a - 1;
printf("Ako budete setali sledecih 5 dana,imacete jedan kilogram manje ili
%.1f", b);return 0;
}
```

Rezultat

```
Koliko imate kilograma?82.7
Ako budete setali sledecih 5 dana,imacete jedan kilogram manje ili 81.7
```

Izvorni kod: kod-13_rad-sa-tipovima-podataka.c

```
#include <stdio.h>
main(){
    float a;
    float b;

    printf("Koliko imate kilograma?");
    scanf(" %f",&a);

    b = a - 1;
    printf("Ako budete setali sledecih 5 dana,imacete jedan kilogram manje
ili %.1f", b);

    return 0;
}
```

Rezultat

```
Koliko imate kilograma?82.7
Ako budete setali sledecih 5 dana,imacete jedan kilogram manje ili 81.7
```

Igrajte se i eksperimentisite sa kodom!

printf() funkcija

Kako smo vec naucili funkcija printf() nam sluze za ispis karaktera, brojeva i teksta na ekran. Postoji dosta opcija koje nudi ova funkcija ali za nas je najbitnije da znamo osnovne koje su nam potrebne za svakodnevni rad sa C-om.

Format funkcije printf()

Funkcija printf(), moze doci u razlicitim formatima, ali za nas je najbitnija prilikom ispisa teksta na ekran, sto ce reci njegovog formatiranja.

printf(kontrolniString [, podaci])

Format svaka komande u nastavku kursa ce biti objasnjen na nacin kao sto smo uradili za printf(), gdje **kontrolniString** predstavlja minimalnu i potrebnu vrijednost koju moramo proslijediti funkciji, dok **[, podaci]** unutar uglastih zagrada predstavljaju opcione parametre. (tbc.)

```
printf ("Moj srecan broj je %d", 13); // ispisi "Moj srecan broj je 13"
```

Kako smo vec naucili, svaki string mora biti predstavljen unutar navodnika, samim tim "kontrolniString" se nalazi unutar navodnika, dok sve nakon "kontrolnogString"-a je opciono i zavisi od vrijednosti koje zelimo stampati, odnosno prikazati na ekranu.

Ispis stringa

```
printf("Zdravo Svijete!");  
printf("Moje ime je Goku.")
```

Kao sto vidite C nece automatski prebaciti vas tekst u novi rad, tacnije funkcija printf(). Potrebno je ubaciti escape sekvencu kako bi smo rekli funkciji printf() da prebaci nas tekst u novi red.

Escape sekvence

Kod	Opis
_____	_____
\n	New line
_____	_____
\a	Alarm
_____	_____
\b	Backspace
_____	_____
\t	Tab

—	—
\\	Backslash
—	—
\'	Apostrof
—	—
\"	Navodnici
—	—

```
printf("Zdravo Svijete! \n");
printf("Moje ime je Goku.")
```

Obzirom da `\n` predstavlja još jedan karakter unutar C-a, potrebno ga je staviti u navodnike kako ne bi doslo do greske prilikom izvršavanja koda.

```
printf("Zdravo Svijete!\nMoje ime je Goku.")
```

Jos jednom da utvrdimo znanje, u navodnike smijestamo stringove (niz karaktera), u apostrofe smijestamo pojedinačne karaktere i backslash `\` daje signal compiler-u da počinje nova escape sekvenca, koja ima svoje specifične komande.

Izvorni kod: `kod-14_rad-sa-escape-sekvencama.c`

```
#include <stdio.h>
main(){
    printf("Moje \t ime je \tGoku.\n");
    printf("Krillin je najbolji Gokuov prijatelj. \a\n"); // ovde cemo dobiti
    zvuk
    printf("Prva osoba koju je Gokuu\b upoznao nakon smrti svog djeda Gohana,
    je bila Bulma.");
    printf("\\"Goku u navodnicima\\"");
    printf("Picolo zna kako da ispise karakter \\");

    return 0;
}
```

Rezultat

```
Moje      ime je      Goku.
Krillin je najbolji Gokuov prijatelj.
Prva osoba koju je Goku upoznao nakon smrti svog djeda Gohana, je bila Bulma.
```

```
"Goku u navodnicima"  
Picolo zna kako da ispise karakter \
```

Moramo ukljuciti, ubaciti u nas program `<stdio.h>` uz pomoc directive `#include`, kako bi smo obezbijedili nesmetan rad funkcije `printf()`. U slucaju da zaboravimo compiler nece znati kako da izvrši funkciju `printf()` i dobicemo gresku. Primijetite takodje da se prilikom ispisa trece linije **"Prva osoba koju je Gokuu"...**, drugi karakter **u** je nestao usljed primjene escape sekvence `\b` koja je vratila kursor jedno polje nazad i na ovaj nacin prepisala karakter.

Rezultat

```
Goku ima 14 14 godina. Krilin ima 43.530000 kg
```

Primijetite da C kod stampanja brojeva sa pokretnim zarezmom, iako smo eksplicitno naveli vrijednost 43.53, stampa ukupno 6 decimalnih mjesta nakon zareza. Ovo mozemo kontrolisati na nacin da unutar kontrolnog stringa stavimo . i **broj** izmedju % i f. Na ovaj nacin definisemo broj decimalnih mjesta koji zelimo da se ispisu nakon zareza.

Izvorni kod: kod-15_rad-sa-escape-sekvencama.c

```
#include <stdio.h>
main(){
    printf("%f %.3f %.2f %.1f \n", 333.3333, 333.3333, 333.3333, 333.3333);

    return 0;
}
```

Rezultat

```
333.333300 333.333 333.33 333.3
```

Kontrolni string unutar funkcije printf() kontrolise u potpunosti ispis na nacin da nema slucajnog ispisa. Svaka anomalija je nasa greska prilikom pisanja koda.

Izvorni kod: kod-15_rad-sa-escape-sekvencama.c

```
#include <stdio.h>
main(){
    printf("Proizvod\tCijena\tKolicina\tSuma\n");
    printf("%s\t%f\t%d\t%.2f\n", "Mlijeko", 2.0, 10, (2.0 * 10));
    printf("%s\t%f\t%d\t%.2f\n", "Brasno", 10.0, 2, (10.0 * 2));
    printf("%s\t%f\t%d\t%.2f\n", "Secer", 1.5, 14, (1.5 * 15));

    printf("Preskoci nekoliko linija \n\n\n\n\ni posalji signal za alarm \n\n\n");

    printf("Uspjesno ste zavrшили prvo poglavlje ili ");
    printf("%.4f%c kursa, \n'Programski jezik C za apsolutne pocetnike'\n",
0.333333, '%');
    printf("Cestitam!\n");
}
```

```
    return 0;  
}
```

Rezultat

```
Proizvod      Cijena  Kolicina      Suma  
Mlijeko 2.000000      10      20.00  
Brasno 10.000000       2      20.00  
Secer 1.500000      14      22.50  
Preskoci nekoliko linija
```

i posalji signal za alarm

Uspjesno ste zavrшили prvo poglavlje ili 0.3333% kursa,
'Programski jezik C za apsolutne pocetnike'
Cestitam!

Ovde vidimo da u slucaju da zelimo stampati karakter % unutar kontrolnog stringa, moramo koristiti escape sekvencu, jer % predstavlja konverzioni karakter.

Promjenjive ili varijable u C-u

Obzirom da se programiranje svodi na obradu podataka, raznih tipova, do sad smo naučili da u C-u postoje karakteri, stringovi, cijeli i decimalni brojevi. Kao i u drugim programskim jezicima, tako i u C-u je potrebno da ove podatke sacuvamo i po potrebi pozovemo na obradu, kako bi smo manipulirali njima i dobijali nove. Za ovo su nam potrebne promjenjive ili varijable.

Varijable su nista drugo do lokacije u memoriji, zamislite ih kao kutije, kontejnere koji cuvaju broj ili karakter.

U C-u imamo nekoliko tipova promjenjivih, isto kao sto imamo nekoliko tipova podataka. Svaki tip podatka definise tip varijable, tako imamo da integer varijable sadrže cjelobrojne tipove podataka, float varijabla može da sadrži samo float tipove podataka i tako dalje.

Podaci, tipovi podakata, sa kojima smo se do sad susreli se jos zovu literarni ili konstante (literal data / constant data) i ovi podaci su ne promjenjivi. Broj 5 ili karakter 'a' uvijek ce imati vrijednost 5 i 'a'. Ali podaci sa kojima radimo u svakodnevnom zivotu, poput broja godina, naziva karaktera, visine plate, tezine moraju biti promjenjivi. Tu uskacu varijable, kako smo vec naveli lokacije u memoriji koje sadrže definisane vrijednosti koje se mogu mijenjati po potrebi.

Pored dosta tipova podataka, ovo su najosnovniji i nama potrebni za pracenje i uspjesno učenje C programiranja. Primijetite da vecina varijabli ima svoje odgovarajuće tipove podataka. Sa dodatnim tipovima i podjelama mozete se upoznati kroz samostalno istrazivanje.

Naziv	Opis
char	Sadrzi/prihvata karaktere kao 'x', '7', '^' reda 1 byte (-128 - 127 ili 0 - 255)
int	Sadrzi/prihvata cijele brojeve kao 1, 0, -3 reda 2 byte (-32,768 - 32,767) ili reda 4 (-2,147,483,648 to 2,147,483,647)
float	Sadrzi/prihvata decimalne brojeve 3.14 -2.3 reda 4 byte (2.3E-308 - 1.7E+308)
double	Sadrzi/prihvata ekstremno velike i male brojeve sa pokretnim zarezm reda 8 byte (2.3E-308 - 1.7E+308)

Iz prethodne tabele u polju naziv vidimo listu kljucnih rijeci koje koristimo prilikom konstrukcije promjenjive u C programskom jeziku. Drugim rijecima, ukoliko zelite definisati varijablu koja ce sadržati podatak tipa cijeli broj (integer), koristicemo kljucnu rijec **int**. Prije nego krenemo aktivno u primjenu promjenjivih, ajde da naucimo osnovna pravila prilikom kreiranja varijabli.

Naziv varijable, davanje imena varijabli

Varijabla mora imati naziv.

Naziv varijable mora biti jedinstven.

Preporučena dužina naziva varijable treba biti od 1 do 31 karakter. (manja mogućnost greske)

Naziv varijable mora početi slovom, nakon čega mogu ići druga slova, brojevi i donja crta `_`, u bilo kojoj kombinaciji.

C dozvoljava da naziv varijable počne donjom crtom `_`, ali se ne preporučuje, obzirom da C ima ugrađene funkcije koje počinju sa donjom crtom.

Ispravni nazivi varijabli:

```
varijabla promjenjivaIme karakterNaziv godine_karaktera brojKolicina Prezime
```

Pogresni nazivi varijabli:

```
2Godine 5_tezina 'promjenjiva Godine' ime,prezime
```

Definisanje varijable

Prije nego počnemo koristiti varijable moramo ih definisati. Definisanje ili deklaracija varijabli, nije ništa drugo nego informisanje C-a da nam je potreban prostor u memoriji računara, koji želimo rezervisati za pohranjivanje naših podataka. Gledajte na to kao rezervisanje stola u restoranu, prije nego imate večeru sa prijateljima, potrebno je prethodno nazvati restoran i rezervisati sto. Sto predstavlja prostor u restoranu ili kontejner koji ćemo koristiti za postavljanje podataka u obliku hrane, dok opet stolice su različite vrste varijabli koje će služiti za postavljanje podataka u obliku vaših prijatelja različite težine i oblika. Nadam se da ste stvorili sliku 😊.

```
main(){
    // moje varijable
    char odgovor;
    int kolicina;
    float cijena;
    /*ostali dio koda*/
}
```

Prethodni kod predstavlja način definisanja tri promjenjive: odgovor, kolicina i cijena. Ove promjenjive mogu prihvatiti/pohraniti/sadržati tri različite vrste tipova podataka: karakter, cijeli broj i decimalni broj respektivno. U slučaju da prethodno nismo definisali varijable, ne bi bili u mogućnosti koristiti kasnije u našem programu, dakle varijable se prvo moraju definisati da bi se koristile.

Uvijek možete definisati više od jedne varijable istog tipa u istom programu.

```
main(){
    // moje varijable
    char prvoSlovoImena;
    char prvoSlovoPrezimenaa;
    /*ostali dio koda*/
}
```

```
main(){
    /* moje varijable, napisane u jednoj liniji
       jer su istog tipa */
    char prvoSlovoImena, prvoSlovoPrezimenaa;
    /*ostali dio koda*/
}
```

Ako se varijable definisu unutar tijela funkcije, dakle izmedju viticastih zagrada {}, sto je slucaj u prethodnom primjeru sa funkcijom main(), ove varijable se nazivaju **lokalne varijable**. C takodje ima opciju definisanja **globalnih varijabli**, varijable definisane izvan viticastih zagrada {}. Programeri u C-u uglavnom preferiraju korištenje **lokalnih varijabli**, vise detalja u nastavku kursa.

Dodjeljivanje vrijednosti varijabli

Konacno, operator dodjele = nam služi da dodijelimo/pohranimo vrijednost u nasu promjenjivu. Iako ovo zvuči kao otkrivanje tople vode, radi se o jednostavnom znaku jednakosti i dodjeljivanje vrijednosti varijabli izgleda ovako

```
promjenjivaNaziv = promjenjivaVrijednost;
```

Gdje je **promjenjivaNaziv** naziv varijable u koju zelimo pohraniti vrijednost **promjenjivaVrijednost**. Naravno varijabla mora biti prethodno definisana kako smo naucili, da bi mogli da joj dodijelimo vrijednost.

promjenjivaVrijednost moze biti broj, karakter ili neka matematicka operacija koja daje konacan rezultat u vidu broja, dakle opet broj. U sledecem primjeru imamo dodjeljivanje vrijednosti varijabli za definisane varijable iz prethodnog primjera. Zapamtite kad god imamo operator dodjele = C program to cita na nacin: izracunaj sve sto je na desnoj strani od opratora dodjeli ili uzmi vrijednost koja se nalazi desno i dodijeli je promjenjivoj koja se nalazi na lijevoj strani od operatora dodjele =.

```
main(){
    // moje varijable
    odgovor = "C";
    kolicina = 15;
    cijena = 13.97;
    // nove varijable koje moraju biti prethodno definisane
```

```
zbir = 4 + 7;
suma = kolicina * cijena; // koristenjem prethodno definisanih varijabli

/*ostali dio koda*/
}
```

Izvorni kod: kod-15_rad-sa-varijablama.c

```
/* program koji izracunava potrepstine za
   za polazak u prvi razred osnovne skole*/
#include <stdio.h>

int main(){
    // definisanje varijabli
    char prvoSlovoImena, prvoSlovoPrezimana;
    int broj_olovki;
    int broj_sveski;
    // definisanje varijabli sa dodjelom vrijednosti
    float cijena_olovke = 0.50;
    float cijena_sveske = 1.68;
    float cijena_obroka = 2.50;

    // informacije o prvom učeniku
    prvoSlovoImena = 'G';
    prvoSlovoPrezimana = 'C';
    broj_olovki = 4;
    broj_sveski = 5;

    printf("%c.%c. treba %d olovki, %d sveski i jedan obrok dnevno.\n",
           prvoSlovoImena, prvoSlovoPrezimana, broj_olovki, broj_sveski);
    printf("Ukupna cijena potrepstina za učenika %c%c je %.2fKM\n\n",
           prvoSlovoImena, prvoSlovoPrezimana, broj_olovki * cijena_olovke
           + broj_sveski * cijena_sveske + cijena_obroka);

    // informacije o drugom učeniku
    prvoSlovoImena = 'B';
    prvoSlovoPrezimana = 'V';
    broj_olovki = 3;
    broj_sveski = 2;

    printf("%c.%c. treba %d olovki, %d sveski i jedan obrok dnevno.\n",
           prvoSlovoImena, prvoSlovoPrezimana, broj_olovki, broj_sveski);
    printf("Ukupna cijena potrepstina za učenika %c%c je %.2fKM\n\n",
           prvoSlovoImena, prvoSlovoPrezimana, broj_olovki * cijena_olovke
           + broj_sveski * cijena_sveske + cijena_obroka);

    // informacije o trecem učeniku
    prvoSlovoImena = 'P';
```

```
prvoSlovoPrezimana = 'F';
broj_olovki = 10;
broj_sveski = 3;

printf("%c.%c. treba %d olovki, %d sveski i jedan obrok dnevno.\n",
       prvoSlovoImena, prvoSlovoPrezimana, broj_olovki, broj_sveski);
printf("Ukupna cijena potrepstina za učenika %c%c je %.2fKM\n\n",
       prvoSlovoImena, prvoSlovoPrezimana, broj_olovki * cijena_olovke
       + broj_sveski * cijena_sveske + cijena_obroka);

return 0;

}
```

Rad sa stringovima u C-u

Vjerovatno se pitate zasto sve do sad nismo definisali niti jednu promjenjivu kojoj je dodijeljena kompletna rijec ili recenica. To je zato sto u C programskom jeziku ne postoji varijabla tipa string, sto opet ne znaci da nema nacina da se string sacuva unutar varijable.

Do sad smo naucili da da u slucaju ispisa stringa isti moramo staviti unutar navodnika. Takodje smo naucili kako koristiti printf() funkciju za ispis stringa. Jedino sto je ostalo jeste da naucimo na koji nacin mozemo definisati varijablu koja je u stanju da pohrani **string (niz karaktera)**. Prije nego vas upoznam sa ovim specijalnim ali jednostavnim nacinom definisanjem varijable koja moze da pohrani string, bitno je da razumijete **"String Terminator"**.

String Terminator

Kada su u pitanju stringovi, C u pozadini radi neke cudne stvari na sebi svojstven nacin 😊, ali nista da mi ne mozemo nauciti. O cemu se radi? U C-u postoji nesto sto se zove "String Terminator", a jednostavnim rijecima **C dodaje nulu na kraj svakog stringa koji definisemo**. Bukvalno nulu **0**, a ova nula se moze pojaviti pod jos par naziva:

- Null zero
- Binary zero
- String terminator
- ASCII 0
- \0

Kada god definisemo neki string (niz karaktera), C na kraju stringa dodaje null zero ili string terminator. Na ovaj nacin C zna (moze da odredi) kada se nalazi na kraju nekog stringa, dakle samo ako na kraju stringa pronadje null zero. Naravno, ovaj karakter nije vidljiv programerima ali ipak je tu u memoriji i C ga intenzivno koristi.

Ako na primjer imamo string

```
"Goku"
```

```
|G|o|k|u|
```

vidjecemo da ovaj string ima 4 karaktera, medjutim C ovaj string vidi kao

```
|G|o|k|u|\0|
```

```
|_____ string terminator, null zero, binary zero
```

dakle ukupno 5 karaktera ili 5 bajtova da budem precizniji.

Duzina stringa

Bez obzira na string terminator, zapamtite kada god se radi o dužini stringa prilikom programiranja u C-u, kao rezultat dobit ćemo vrijednost dužine stringa koja ne uključuje string terminator. Iako C koristi **null zero** kako bi znao kada je kraj stringa, isto tako zna da u slučaju računanja dužine stringa, prilikom ispisa, **null zero** nije uključen u konačan rezultat.

Svi navedeni stringovi su dužine 6 karaktera.

```
"Goku ."  
"Bulma."  
"Krillin"
```

Međutim, kada planiramo i definisemo varijable u koje smjestamo stringove moramo voditi računa i uključiti **string terminator** prilikom određivanja dužine varijable. Najlakše je zapamtiti na način kada definisemo jedan jedini karakter i smijestamo ga u varijablu, čak i tad ovaj karakter ima dužinu od dva karaktera u memoriji računara jer pored karaktera moramo smjestiti i **null zero** ili **terminator string**.

Niz karaktera: Lista karaktera ("definisanje string varijable")

Niz karaktera je upravo ono sto nam je potrebno za definisanje karaktera u memoriji racunara i dodjeljivanje varijabli u C programskom jeziku. Iako je ovo u pojednostavljeno u mnogim programskim jezicima, C je zadrzao ovaj pristup i vjerujte nakon nekoliko godina programiranja kada razmilite o ovom nacinu bicete zahvalni.

Niz (array) je specijalan tip promjenjive. Ostali tipovi podataka kao sto su int, float, char i drugi imaju svoje odgovarajuce array tipove podatka. Niz nije nista drugo, nego lista promjenjivih istog tipa.

Prije nego definimo i omogucimo korištenje niza karaktera za kreiranje stringa, potrebno je reci C-u da zelimo kreirati niz karaktera, na isti nacin kao sto bi smo rekli C-u da trebamo kreirati bilo koju drugu promjenjivu.

Prilikom kreiranja niza koristimo uglaste zagrade **[]**, odmah nakon naziva varijable (naziva niza) i u slucaju da zelimo unaprijed odrediti duzinu niza stavljamo broj unutar ovih zagrada (duzina niza od pet karaktera **[5]**). U slucaju da nismo sigurni kolika je duzina niza, mozemo dati neku vecu vrijednost ili ostaviti prazne uglaste zagrade, cime govorimo C-u da sam izracuna duzinu niza i na taj nacin definise duzinu varijable.

Izvorni kod: kod-17_rad-sa-stringovima.c

```
#include <stdio.h>

int main(){
    char naziv_karaktera [5] = "Goku"; /* definisanje niza karaktera,
                                         naziv_karaktera */

    printf("Zdravo %s.", naziv_karaktera);

    return 0;
}
```

Dakle sva prica da bih vam rekao da je definisanje niza karaktera sasvim jednostavno, isto kao kad smo radili definisanje jednog karaktera samo dodamo uglaste zagrade nakon naziva promjenjive i po potrebi dodamo broj unutar zagrada koji definise duzinu karaktera. Ali ovo je zaista bitno da razumijete, jer sa ovim temeljima nastavljamo dalje i vjerujte bilo koji programski jezik nakon ovoga ce biti maciji kasalj.

Bitno je napomeniti da prilikom definisanja duzine promjenjive uvijek trebamo uzeti u obzir maksimalnu vrijednost i dodati jos jedan karakter kako bi smo pohranili i null zero.

Dobar primjer je duzina mjeseci u godini

```
Januar
Februar
Mart
April
Maj
Juni
```

```

Juli
Avgust
Septembar
Oktobar
Novembar
Decembar

```

Ovde mozemo da vidimo kako mjesec Septembar ima najvise karaktare i za njegovo pohranjivanje ce biti potrebno 9 karaktera + 1 null zero karakter, sto je ukupno 10 karaktera.

Dakle prilikom rezervacije mjesta za pohranjivanje stringa u memoriju, uvijek je potrebno gledati na vrijednost najduzeg stringa i racunati na null zero ili string terminator. Preporuka je uvijek definisati niz karaktera koji je malo veci od najveceg stringa.

Kako to izgleda iz perspektive C-a.

```

char mjesec[10] = "Septembar";      <-- definisanje niza karaktera(string)

|S|e|p|t|e|m|b|a|r|\0|              <-- niz karaktera
[0][1][2][3][4][5][6][7][8][9]      <-- subscripts (indeksi)

char mjesec[10] = "Januar"           <-- definisanje niza karaktera(string)

|J|a|n|u|a|r|\0|?|?|?|              <-- niz karaktera
[0][1][2][3][4][5][6][7][8][9]      <-- subscripts (indeksi)

```

Jos jedna bitna stvar koju trebate zapamtiti jeste da C uvijek pocinje brojanje od nule **[0][1]...[9]...[100]...**

Izvorni kod: kod-18_rad-sa-stringovima.c

```

#include <stdio.h>

char mjesec[10] = "Septembar";

int main(){
    printf("%s\n", mjesec);
    mjesec[0] = 'M';
    mjesec[1] = 'a';
    mjesec[2] = 'r';
    mjesec[3] = 't';
    mjesec[4] = '\0'; // null zero, u slucaju da nismo definisali
                      // rezultat bi bio 'Martembar'
    printf("%s\n", mjesec);
}

```

Rezultat

```
Septembar  
Mart
```

Svaki pojedinačni dio niza karaktera se naziva elementom. Niz mjesec ima ukupno deset elemenata, koji su precizno određeni subscriptima. Subscripts su brojevi koji određuju na kojoj poziciji se nalazi element niza. Svaki niz počinje o 0, što znači da se i nula broji iz perspektive C-a. Svaki element našeg niza je karakter, što znači da niz ili lista karaktera čini string.

Stampanje niza je jednostavno kao i stampanje pojedinačnog stringa

```
printf("Mjesec je %s", mjesec);
```

Inicijalizacija stringa (strcpy())

Kako smo već naučili prilikom definisanja stringa dovoljno je reći naziv i nakon toga u uglastu zagradu staviti broj koji određuje ukupnu dužinu stringa. Takođe, ako nismo sigurni kolika je dužina našeg stringa unaprijed, možemo ostaviti uglaste zagrade prazne. U oba slučaja konačan rezultat će biti isti.

```
char mjesec[10] = "Januar";  
char mjesec[] = "Januar";
```

U drugom slučaju C računa broj karaktera u stringu i na kraju dodaje null zero. Naravno postoje i druge inicijalne kreacije od C-a, računajući i null zero karakter naša promjenjiva mjesec će do kraja programa moći da pohrani ukupno 7 karaktera u nizu (uključujući null zero).

Ali šta ako ne želimo vršiti inicijalizaciju stringa u jednoj liniji kao do sad? Recimo da želimo uraditi nešto slično kao

```
char mjesec[10];      <-- definisanje varijable mjesec,  
                      niz karaktera ili string  
  
mjesec = "Januar";    <-- inicijalizacija varijable,  
                      dodjeljivanje vrijednosti  
                      NIJE DOZVOLJENO NA OVAJ NAČIN
```

Nazalost, ovo je pogrešno i neće raditi kao u prethodnim primjerima kada smo definisali karaktere i brojeve, jer C ne dozvoljava da se string dodijeli nizu karaktera na navedeni način.

Operatorom dodjele, znakom = je jedino moguće dodijeliti string nizu u jednoj liniji prilikom definisanja stringa.

U slučaju da ipak želimo kasnije kroz naš program izmijeniti ovu vrijednost, odnosno definisati novu ili inicijalizirati novu vrijednost imamo dvije opcije. Jednu smo već vidjeli u primjeru kada smo preko subscripta-a (indeksa) pristupili svakom karakteru posebno, što je priznate naporno. Drugi, elegantniji način, jeste funkcija koju nam nudi C, a koja se zove **strcpy()** (skraćeno od string copy). Ova funkcija prihvata dva parametra, i prilikom pozivanja dodjeljuje, kopira, mapira, ono što se nalazi u drugom parametru na desnoj strani u prvi parametar na lijevoj strani. strcpy() automatski dodaje null zero na kraju stringa, ali također može smjestiti samo onu veličinu stringa u memoriju koja je prethodno definisana.

Izvorni kod: kod-19_inicijalizacija-stringa.c

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

char mjesec[10] = "Septembar";

int main(){
    printf("%s\n", mjesec);
    mjesec[0] = 'M';
    mjesec[1] = 'a';
    mjesec[2] = 'r';
    mjesec[3] = 't';
    printf("%s\n", mjesec);

    strcpy(mjesec, "Decembar");

    printf("%s\n", mjesec);
}
```

Rezultat

```
Septembar
Martembar
Decembar
```

Primijetite da samo uključili novi header fajl (**#include <string.h>**), kako bi smo omogućili korištenje funkcije strcpy().

Izvorni kod: kod-20_definisanja-i-inicijalizacija-karaktera.c

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

```
int main(){
    // karakter_1 moze pohraniti naziv karaktera
    // od ukupno 7 karaktera, 8. služi za null zero
    char karakter_1[8];

    // karakter_2 moze pohraniti naziv karaktera
    // od ukupno 6 karaktera, 7. služi za null zero
    char karakter_2[] = "Krillin";

    // karakter_3 moze pohraniti naziv karaktera
    // od ukupno 29 karaktera, 30. služi za null zero
    char karakter_3[30];

    char planeta_1[10] = "Vegeta";
    char planeta_2[25];
    char planeta_3[20];

    // karakter_1 je inicijaliziran karakter po karakter
    // sto i nije bas efikasno
    karakter_1[0] = 'G';
    karakter_1[1] = 'o';
    karakter_1[2] = 'k';
    karakter_1[3] = 'u';
    karakter_1[4] = '\\0'; // obavezno vodite racuna da
                          // ne zaboravite dodati null zero
                          // kako bi ste rekli C-u kada se niz završava

    strcpy(karakter_3, "Picollo");
    strcpy(planeta_2, "Zemlja");
    strcpy(planeta_3, "Namek");

    printf("%s dolazi sa planete %s. \\n", karakter_1, planeta_1);
    printf("%s dolazi sa planete %s. \\n", karakter_2, planeta_2);
    printf("%s dolazi sa planete %s. \\n", karakter_3, planeta_3);

    return 0;
}
```

Rezultat

```
Goku dolazi sa planete Vegeta.
Krillin dolazi sa planete Zemlja.
Picollo dolazi sa planete Namek.
```

Da zakljucimo, ako imate potrebu da sacuvate rijeci u varijable, C vam ne nudi string tip podataka, vec iste morate da sacuvate kao niz karaktera. Uvijek racunajte na "nevidljivi" null zero. Kada definisete vas niz, racunajte da subsripts pocinje sa nulom, dakle C pocinje brojanje od nule, uklucujuci i nulu a ne od 1 kako smo to navikli u svakodnevnom zivotu. Postoje tri nacina kako mozemo staviti string u niz karaktera:

1. prilikom definisanja niza u istoj liniji vrsimo inicijalizaciju
2. element po element
3. koristenjem funkcije `strcpy()`, koja zahtijeva ukljucivanje dodatnog header-a `#include <string.h>`

Direktive #include i #define

Sve vrijeme do sad koristimo #include a nismo posteno sjeli i porazgovarali o cemu se radi, pa #include na tebe je red.

#include predstavlja pre-procesorku direktivu, koja nam služi za dodavanje, uključivanje (include ili ukljuci, uvezi iz vana) eksternih datoteka koje sadrže prethodno definisane funkcije neophodne za rad sa našim programom. Kada god C, odnosno compiler, nađje na znak taraba (#) ovo je za njega okidač da se radi o pre-procesorskoj direktivi i u skladu sa tim se i ponasa. Pre-procesorska direktiva se ne izvršava za vrijeme pokretanja programa (runtime), već za vrijeme kompajliranja programa.

Najcesce pre-procesorke direktive koje cemo koristiti tokom kursa jesu:

```
#include  
#define
```

Uključivanje eksternih fajlova (#include)

#include ima dva formata, koji su na prvu skoro pa identicni

```
#include <naziv_fajla>  <-- include built-in definisane funkcije  
#include "naziv_fajla"  <-- include korisnicki definisane funkcije
```

Dakle #include direktiva nije nista drugo nego uključivanje sadržaja nekog drugog fajla unutar našeg programa. Ovo uključivanje se desava samo za vrijeme kompajliranja našeg programa. Naravno uključeni kod je sastavljen od komentara ili opet od samog C koda.

Na ovaj način možemo uključiti već predefinisane/ugradjene funkcije (built-in funkcije) u C-u, kao

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

za potrebe korištenja funkcije printf() ili

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

za potrebe korištenja strcpy(), ali i more drugih. Built-in funkcije dolaze sa instalacijom našeg kompajlera. Ovi fajlovi se još zovu i header fajlovi (header zaglavlje, jer se nakon uključivanja nalaze na početku našeg fajla, znači u samom zaglavlju) i pravilo je da im se daje ekstenzija **.h**. Naravno #include se može koristiti bilo gdje u našem C kodu ali je preporuka da se koristi na početku pisanja izvornog koda.

Ako želimo uključiti funkcije koje smo samostalno kreirali sintaksa ove direktive umjesto znakova manje vece,

koristi navodnike. Dakle, ispravno uključivanje naseg header fajla u program, koji se nalazi u istom direktoriju gdje i nas izvrni kod bi i izgledalo kao

```
#include "mojaFunkcija.h"
```

Hajde da vidimo kako to izgleda iz perspektive samog kompajlera.

```
/* dio glavnog koda*/

#include "promjenjive.h"
char naziv_karaktera[8] = "Goku";
#include "mojaFunkcija.h"
printf("%s dolazi sa planete %s\n", naziv_karaktera, naziv_planete);
printf("Zbir dva broja a + b = %d\n", zbir);

/* ostatak glavnog koda*/
```

```
/* fajl: promjenjive.h */
int a = 10;
int b = 11;
char naziv_planete[20] = "Vegeta";
```

```
/* fajl: mojaFunkcija.h*/
int zbir = a+b;
```

Konacan kod koji kompajler vidi za vrijeme build-anja izgleda ovako

```
/* dio glavnog koda*/

/* fajl: promjenjive.h */
int a = 10;
int b = 11;
char naziv_planete[20] = "Vegeta";

char naziv_karaktera[8] = "Goku";
/* fajl: mojaFunkcija.h*/
int zbir = a+b;

printf("%s dolazi sa planete %s\n", naziv_karaktera, naziv_planete);
printf("Zbir dva broja a + b = %d\n", zbir);
```

```
/* ostatak glavnog koda*/
```

U slucaju da koristimo navodnike prilikom pozivanja `#include` direktive, C ce prvo traziti fajl koji smo mi definisali a nakon toga built-in funkcije koje dolaze predefinisane od C-a, cak i ako damo isto ime kao sto je built-in ali ovo se ne preporucuje. Pozeljno je da se `#include` direktiva nalazi prije `main()` funkcije.

Definisanje konstanti (`#define`)

#define pre-procesorska direktiva nam služi za definisanja konstanti. Konstanta je isto sto i literalna vrijdnost. Kako smo vec naucili literlna vrijednost se ne mijenja, recimo broj 5 ce uvijek biti broj 5, slovo/karakter 'A' ce uvijek biti slovo 'A'. `#define` pre-procesorska direktiva nam sluza da imenujemo, damo nazive, literalnim vrijednostima. Kada se literlana vrijednost imenaju na ovaj nacin, to se u C-u jos naziva konstanta ili definisana konstanta.

Kontanta nije promjenjiva ili varijabla, iako izgledaju slicno razlika je sto kad jednom u zaglavlju definisemo konstantu ona ostaje takva, ne mijenja se kroz dalji tok programa za razliku od promjenjive.

Format konstante izgleda ovako

```
define KONSTANTA definicijaKonstante
```

Primjeri definisanja konstante

```
#define GODINE_LIMIT 99      <-- gdje god se pojavi konstanta GODINE_LIMIT  
                             zamijeni vrijdnoscu 99  
#define PI 3.14             <-- gdje god se pojavai konstanta PI  
                             zamijeni vrijdnoscu 3.14  
#define KARAKTER_IME "Goku" <-- gdje god se pojavi konstanta KARAKTER_IME  
                             zamijeni vrijdnoscu "Goku"
```

Slicno kao sa direktivom `#include`, C ce kada god naidje na direktivu `#define` zamijeniti vrijednost `KONSTANTA` sa vrijednoscu definicijaKonstante. Prilikom davanja naziva konstantama, postoji izuzetak u C-u i preporucuje se davati nazive velikim slovima. Na ovaj nacin kroz nas kod, uvijek mozemo razlikovati konstante od promjenjivih. Takodje prilikom definisanja konstanti ne koristimo operator dodjele `=`. Kada se konstante jednom definise, nije dozvoljeno njeno mijenjenje kasnije u toku programa.

```
PI = 3.15 /* nije dozvoljeno */
```

Direktiva `#define` nije komanda u C-u i ne izvsava se za vrijeme pokretanja programa (u runtime-u) vec samo za vrijeme kompajliranja. Sve dok definisete konstantu prije funkcije `main()`, kompletan program ce imati ovu informaciju i moci cete ukljuciti vrijednost definisane konstante u vas kod.

Kreiranje header-a i uključivanje u program

Kreirajmo sada program primjenjujući sve do sad naučeno.

Izvorni kod: konstante.h

```
/* header sa definisanim konstantama */
#define KARAKTER_NAZIV "Goku"
#define KARAKTER_GODINE 14
#define KARAKTER_PLANETA "Vegeta"
#define PI 3.14
#define KONSTANTA_KOJU_NECEMO_KORISTITI 13
```

Izvorni kod: kod-21_program-sa-ukljucivanjem-zaglavlja.c

```
/* glavni program koji uključuje
korisnicki kreiran fajl konstante.h
*/
#include <stdio.h>
#include <string.h>
#include "konstante.h"

int main(){
    int trenutna_godina;

    int poluprecnik;
    char karakter_naziv[10] = "Krillin";

    printf("%s dolazi sa planete %s\n",
           KARAKTER_NAZIV, KARAKTER_PLANETA);
    printf("%s ima najboljeg prijatelja koji se zove %s\n",
           KARAKTER_NAZIV, karakter_naziv);
    printf("%s ima %d godina.\n",
           KARAKTER_NAZIV, KARAKTER_GODINE);

    poluprecnik = 2;

    printf("Obim kruga poluprecnika vrijednosti %d iznosi %.2f\n",
           poluprecnik, 2 * poluprecnik * PI);

    return 0;
}
```

Rezultat

Goku dolazi sa planete Vegeta
Goku ima najboljeg prijatelja koji se zove Krilin
Goku ima 14 godina.
Obim kruga poluprecnika vrijednosti 2 iznosi 12.56

Interakcija sa korisnikom - scanf()

Konacno, dosli smo do tacke kada konacno uz sve prethodno nauceno mozemo dodati i interakciju sa korisnikom u C-u. Dakle, sve do sad su nasi programi bili staticni na neki nacin, uglavnom smo definisali promjenjive, naučili smo kako se definisu konstante i konacno smo sve te vrijednosti ispisivali uz pomoc funkcije `printf()`. Sta ako zelimo da nas program bude malo dinamicniji, na nacin da mi kao korisnik mozemo unijeti vrijednosti svakim novim pokretanjem programa i umjesto koristenja operatora dodjele svaki put direktno pisuci u kod, dodijelimo vrijednost direktno koju proslijedi korisnik?

Za ovu potrebu C nam obezbjedjuje funkciju **`scanf()`**. Kako vec znamo **`printf()`** salje ispis na ekran, dok **`scanf()`** funkcija ceka na unos podataka sa tastature. Mozda ce vam se funkcija `scanf()` ciniti malo konfuzna na pocetku, ali ne brinite. Svi principi koji vrijede za `printf()`, u vecini slucajeva su primjenjivi na `scanf()` i ako ste dobro naučili `printf()` veoma lako cete usvojiti sintaksu `scanf()` funkcije, da ne pricam o prednosti i prosirenju mogucnosti koje cete obezbijediti svom C programu.

`scanf()` je ugradjena (built-in) funkcija koja dolazi sa C kompajlerom. Kako bi smo omogucili koristenje `scanf()` funkcije, kao i kod `printf()`, moramo uvesti odredjen header fajl direktivom `#include`. Na svu srecu `scanf()`, kao i `printf()`, koristi isti header `<stdio.h>`, dakle nema dodatnog specijalnog header-a. Pored slicne sintakse `scanf()` i `printf()` funkcija, `scanf()` funkcija takodje koristi iste konverzacione karaktere poput **`%s`**, **`%d`**, **`%f`**.

Format funkcije scanf()

Funkcija `scanf()` ima sledeci format

`scanf(kontrolniString [, podaci])`

Kada C program naidje na funkciju `scanf()`, on se jednostavno zaustavi i ceka unos sa tastature. Varijable koje se nadju unutar `scanf()` ce biti prihvacene i procesurina na osnovu definisanog karaktera konverzije. Prekid funkcije `scanf()`, odnosno nastavljajanje izvorsavanja C programa, se desava sve dok korisnik ne pritisne **ENTER** ili **"Return key"**. Nikad nemojte definisati escape sekvence (`\t`, `\a`, `\n`) unutar `scanf()` funkcije, jer dolazi do zbunjivanja funkcije. Konkretno `\n` ce se ponasati kao novi red, sto ce zbuniti `scanf()` i trazice nam unos dva puta.

Prije koristenja funkcije `scanf()`, obicno uvijek ide funkcija `printf()` koja nam služi da informisemo korisnika sta zelimo, koju vrijednost promjenjive, da nam proslijedi. U slucaju da izostavimo `printf()` funkciju, program ce se automatski zaustaviti na funkciji `scanf()` i cekace dok nesto ne unesemo sa tastature a zatim udarimo ENTER. Ovo moze biti pomalo zastrasujuce iskustvo za korisnika 😊.

Izvorni kod: `kod-22_rad-sa-funkcijom-scanf.c`

```
/* program koji služi za demonstraciju funkcije scanf */
#include<stdio.h>

int main(){
    // definisanje promjenjivih za koje cemo
    // traziti unos od korisnika
```

```
char prvoSlovoImena;  
char prvoSlovoPrezimana;  
int godine_karaktera;  
int omiljeni_broj;  
  
printf("Prvo slovo vaseg imena: ");  
scanf(" %c", &prvoSlovoImena);  
  
printf("Prvo slovo vaseg prezimana: \n");  
scanf(" %c", &prvoSlovoPrezimana);  
  
printf("Unesite vase godine: ");  
scanf(" %d", &godine_karaktera);  
  
printf("Unesite omiljeni broj: \n");  
scanf(" %d", &omiljeni_broj);  
  
printf("Vasi inicijali su %c.%c. i imate %d godina.\n",  
      prvoSlovoImena, prvoSlovoPrezimana, godine_karaktera);  
printf("Vas omiljeni broj je: %d\n", omiljeni_broj);  
}
```

Rezultat

```
Prvo slovo vaseg imena: G  
Prvo slovo vaseg prezimana:  
K  
Unesite vase godine: 14  
Unesite omiljeni broj:  
13  
Vasi inicijali su G.K. i imate 14 godina.  
Vas omiljeni broj je: 13
```

Kao sto vidite scanf() ide ruku pod ruku sa printf() funkcijom. Da vidite koliko je zbunjujuce raditi ako uklonimo bilo koji printf() prije scanf() funkcije u prethodnom programu, pokusajte sami.

Iz prethodnog primjera mozemo vidjeti da prve dvije scanf() funkcije prihvataju vrijednost tipa karakter (zato koristimo %c konverzacioni karakter), dok treca i cetvrta funkcija scanf() prihvataju vrijednosti tipa broj/digit (digit - %d), a zatim ih dodijeljuju odgovarajucim promjenjivim, odnosno varijablama.

Takodje dobra navika je prilikom definisanja konverzacionih karaktera unutar scanf() funkcije, staviti jedan razmak ispred, jer kasnije recimo kad unosimo vise od jednog broja ili karaktera ovo dolazi do izrazaja.

Najvaznije, primijetite znak ampersand & ispred svake od definisanih varijabli unutar scanf() funkcije. Za sad je bitno da znate, kako je ovaj znak & (ampersand) nophodan da bude ispred svake promjenjive u funkciji scanf()

kako bi funkcija nesmetano funkcionisala. U slucaju da izostavimo ili zaboravimo **&**, scanf() nece prihvatiti unos od strane korisnika u promjenjvu i vjerovatno cemo dobiti gresku za vrijeme kompajliranja naseg programa.

Nedostaci i "problemi" funkcije scanf()

Da se razumijemo, u C-u ne postoje problemi, ma koliko to zvučalo sebično, C je napisan na takav način i radi od kad znamo za kompjutere, tako da je svaka implementacija urađena sa namjenom a na nama je da se prilagodimo i naučimo kako se koristi.

Prvi od problema sa kojim se susrećemo prilikom korištenja scanf() funkcije, jeste da korisnik mora unijeti tačno ono što scanf() od njega očekuje. U slučaju da korisnik unese realan broj (float) na mjestu gdje funkcija scanf() po definiciji očekuje cijeli broj (integer) doći će problema prilikom izvršavanja ili u drugim slučajevima do netačnog rezultata. Ovo može biti veoma opasno kada su finansijski proračuni u pitanju. Za sad ćemo pretpostaviti da će korisnik svaki put unijeti tačno ono što se od njega očekuje. O prevazilaženju ovog problema će biti riječi u nekoj od narednih lekcija.

Takođe, postoji izuzetak pravilu prilikom korištenja ampersand **&** znaka, kada je u pitanju scanf() funkcija, a to je kada tražimo od korisnika da unese niz karaktera ili string (%s), recimo u slučaju da tražimo unos imena ili prezimena. Ovo za sad zapamtite na način, u slučaju da tražite od korisnika da unese cijeli broj (integer), decimalni broj (float ili double), pojedinačni karakter (char) **moramo staviti ampersand & ispred varijable** prilikom definisanja scanf() funkcije. U slučaju da tražimo od korisnika da unese string ili niz karaktera, **ne stavljamo ampersand & ispred varijable**.

Generalno znak ampersand **&** se nikad ne stavlja ispred **pointer-a**. Kako je niz (array) sam po sebi **pointer**, odatle dolazi ovo pravilo, koje dakle nije slučajno, ali o ovome će biti riječi kasnije kada budemo demistifikovali pointer-e u C programskom jeziku 😊.

Jos jedan od problema na koje nailazimo prilikom korištenja funkcije scanf(), jeste da scanf() prestaje da čita unos stringa nakon što naiđe na prvi razmak ili prazno polje. Na ovaj način smo ograničeni unosom samo jedne riječi koju će scanf() da prihvati sve nakon razmaka će biti odsijeceno ili ignorisano. Dakle ako korisnik unese ime i prezime, scanf() će pročitati samo ime i smjestiti u promjenjivu koju smo prethodno definisali. Za sad problem možemo prevazići na način da nakon funkcije printf(), gdje tražimo unos imena i prezimena, definisemo dvije nove funkcije scanf(), jednu za ime drugu za prezime, ali kako pretpostavljate ovo se rijetko kad dobro završi 😊.

Pitate se zašto uopšte koristimo funkciju scanf() kada ima toliko "nedostatka"? Zato što je jedina od osnovnih funkcija u C-u koja ide ruku pod ruku sa funkcijom printf(), najjednostavnija je i pogodna za demonstraciju unosa i prihvatanja podataka od korisnika.

Izvorni kod: kod-15_rad-sa-escape-sekvencama.c

```
#include<stdio.h>

int main(){
    char naziv_karaktera[30];
    int karakter_godine;
```

```
int mjesec,dan,godina;
float tezina;

//

printf("Unesite vase ime: \n");
scanf(" %s", naziv_karaktera);

printf("Unesite vase godine: \n");
scanf(" %d", &karakter_godine);

printf("Unesite trenutni mjesec, dan i godinu\n");
printf("u formatu XX/XX/XXXX\n");
scanf(" %d/%d/%d", &mjesec, &dan, &godina);

printf("Unesite tezinu u formatu xx.xx\n");
scanf(" %f", &tezina);

printf("Danas je %d/%d/%d godine.\n", mjesec,dan,godina);
printf("Vi imate %d godine i zovete se %s.\n",
       karakter_godine, naziv_karaktera);
printf("Image %.2f kg. ", tezina);

return 0;
}
```

Rezultat

```
Unesite vase ime:
Goku
Unesite vase godine:
14
Unesite trenutni mjesec, dan i godinu
u formatu XX/XX/XXXX
03/05/2020
Unesite tezinu u formatu xx.xx
89.94
Danas je 3/5/2020 godine.
Vi imate 14 godine i zovete se Goku.
Image 89.94 kg.
```