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 ucenje i upotrebu, uskoro cete vidjeti da to ipak nije tako. Ako se pravilno pristupi ucenju 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 ucenjem C jezika stvarate odlicnu osnovu za postizanje vjestina koje ce vam otvoriti vrata za lakse ucenje 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 ucenjem C programskog jezika a ja cu da vam predstavim ovo na najlaksi moguci nacin. Ukoliko budete pratili ovaj kurs i odradite sve vjezbe, 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 odretiti njegovo ponasanje.

Priprema radnog okruzenja

Da bismo napisali instrukcija u C programskom jeziku moramo imati nakakav tekst editor i specijalan compiler koji sluzi 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 koritiiti 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 opustitite i nemojte pokusavati da razumijete svaku liniju koda,
ionako cu vam objasniti sve u detalje, a kroz sam kurs vise puta cemo se vracati 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 (**stampajIzvjestaj(), 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 zavrsena 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 izvrsavanje 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 vaseg 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 vasih 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 realini 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.

Karakteri u C-u

U C programskom jeziku, karakter predstavlja svaki pojedinacni znak koji se moze predstaviti u kompjuteru. Kompjuter poznaje 256 razlicitih karaktera, koje mozete pronaci u ASCII tabeli (tbd. skinuti ASCII tableu ili linkovati), koju naravno nema potrebe da pokusavate uciti napamet.
Primjeri karaktera
A b 1 7 % $? W + - {] .

Cak i razmak ** ** predstavlja karakter, a kao sto mozete vidjeti svako slovo, broj, specijalni znak predstavlja karakter u C-u.
Iako 1 ili 7 izgledaju kao brojevi, sto stvarno jesu, ako smo ih definisali kao karakter tip podatka, mi vise ne mozemo vrsiti osnovne matematicke operacije na njima. Naravno ako definisemo 1 ili 7 kao cjelobrojni tip podatka mi nad njima mozemo vrsiti matematicke operacije. Ista je stvar sa specijalnim znakovima kao sto su **+ - = / %**.

Da bi smo definisali tip podatatka karakter u C-u, potrebno je upotrijbiti 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 ucenje'
napravili smo gresku jer ovo ne moze biti karakter, zato sto ima vise od jednog znaka unutar apostrofa.

U slucaju da zelimo definisati vise od jednog karaktera, moramo koristiti navodnike **"**. Na ovaj nacin mozemo grupisati vise od jednog karaktera a ovakva struktura se naziva jednom rijecju **string** i ovo bi bio ispravan nacin definisanja string-a, za razliku od prethodnog:
"C je jednostavan za ucenje".

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 vaseg 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 vaseg 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 moguce. 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

```c
#include<stdio.h>

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

```
    printf("Upravo je zavrsio poglavlje 1.");
    printf("Goku ima 14.5 godina.");
    printf("Goku voli programiranje.");

    return 0;
}
```

Rezultat

```
Goku je poceo da uci C programski jezik.Upravo je zavrsio 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 poceo da uci C programski jezik. \n");
    printf("Upravo je zavrsio poglavlje 1. \n");
    printf("Goku ima 14.5 godina. \n");
    printf("Goku voli programiranje. \n");

    return 0;
}
```

Rezultat

```
Goku je poceo da uci C programski jezik.
Upravo je zavrsio 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 poceo da uci %c programski jezik \n", 'C');
    printf("Upravo je zavrsio 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 zavrsio poglavlje 1.
Goku ima 14.5 godina.
Goku voli programiranje.
```

%c predstavlja mjesto gdje cemo ubaciti **karakter veliko C** i sluzi nam za reprezentaciju karaktera (skraceno od characther). %c je fiksno dok moze 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 sluzi nam za reprezentaciju cijelih brojeva (skraceno od digit). %d je fiksno i moze 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 sluzi nam za reprezentaciju realnih ili brojeva sa pokretnim zarezom float (skraceno od float). %f je fiksno i moze poprimiti bilo koji broj iz opsega realinih brojeva. Sta biti ako umjesto realnog broja dodijelimo neki cio broj ili neko slovo?

Vjezbajte na nacin da ekperimentisete. Izostavite neke dijelove izvornog koda, sacuvajte i pokusajte 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 nesmijemo 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 iako 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 vaseg 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 potrebeno

```
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 jos jedan karakter unutar C-a, potrebno ga je staviti u navodnike kako ne bi doslo do greske prilikom izvrsavanja koda.

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

Jos jednom da utvrdimo znanje, u navodnike smijestamo stringove (niz karaktera), u apostrofe smijestamo pojedinacne karaktere i backslash **** daje signal compiler-u da pocinje nova escape sekvenca, koja ima svoje specificne komande.

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

```
#include <stdio.h>
main(){
    printf("Moje \t ime je \tGoku.\n");
    printf("Krilin 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.
Krilin 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 izvrsi 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.

Konverzioni karakteri ili karakteri konverzije

Prilikom ispisa brojeva i karaktera, moramo reci C-u na koji nacin se ovi elementi ispisuju. Imali ste priliku da ih vidite ali ajde da detaljnije objasnimo karaktere za konverziju.

```
Konverzioni |  Opis
karakter    |
____________|_____________________________________________
%d          |Cijeli broj - int (d - skraceno od Digit)
____________|_____________________________________________
%f          |Decimalni broj - float (f - skraceno od Float)
____________|_____________________________________________
%c          |Karakter - char (c - skraceno od Character)
____________|_____________________________________________
%s          |Niz karaktera - string (s - skraceno od String)
____________|_____________________________________________
```

Kada zelimo da ispisemo, stampamo neku vrijednost unutar stringa, koristimo konverzione karaktere unutar kontrolnog stringa, a zatim na desnoj strani od kontrolnog stringa, odvojeno zarezom, ubacujemo listu od jednog ili vise vrijednosti ciji broj odgovara upravo broju konverzionih karaktera unutar kontrolnog stringa.

```
                                        ____________________
                  ____________________________|_____________    |
                 |                            |             |    |
printf("Goku ima %d  godine. Krlin ima  %f  godina.", 14, 14.5)
       |________|  |  |___________________|  | |________|
           |       |          |            |      |
       ____|___    |  _________|_________   |    __|____
       |       |   |  |                 |  |   |       |
       Goku ima  14  godine. Krilin ima 14.5  godina.
```

Stringovi i karakteri imaju takodje svoje konverzione karaktere, iako unutar navodnika nije potrebno da ih koristimo, **%s** je potrebno koristiti prilikom kombinovanja karaktera i stringova sa drugim podacima.

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

```c
#include <stdio.h>
main(){
    printf("%s %d %s %s %f %c%c \n", "Goku ima 14 ", 14, "godina.", "Krilin
ima ", 43.53, 'k', 'g');

    return 0;
}
```

Rezultat

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

Primijetite da C kod stampanja brojeva sa pokretnim zarezom, iako smo eksplicitno naveli vrijednost 43.53, stampa ukupno 6 decimalnih mijesta 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

```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

```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\ni posalji signal za alarm \a
\n\n");

    printf("Uspjesno ste zavrsili 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 zavrsili prvo poglavlje ili 0.3333% kursa,
'Programski jezik C za apsolutne pocetnike'
Cestitam!
```

Ovde vidimo da u slucaju da zelimo stampati karaker **%** unutar kontrolonog stringa, moramo koristiti escape sekvencu, jer **%** predstavlja konverzacioni karakter.

Promjenjive ili varijable u C-u

Obzirom da se programiranje svodi na obradu podataka, raznih tipova, do sad smo naucili 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 manipulisali 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 sadrze cjelobrojne tipove podataka, float varijabla moze da sadrzi 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 sadrze definisane vrijednosti koje se mogu mijenjati po potrebi.

Pored dosta tipova podataka, ovo su najosnovniji i nama potrebni za pracenje i uspjesno ucenje C programiranja. Primijetitie da vecina varijabli ima svoje odgovarajuce 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 zarezom
        |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 sadrzati 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.
Preporucena duzina naziva varijable treba biti od 1 do 31 karakter. (manja mogucnost greske)
Naziv varijable mora poceti slovom, nakon cega mogu ici druga slova, brojevi i donja crtu _, u bilo kojoj kombinaciji.
C dozvoljava da naziv varijable pocne donjom crtom _, ali se ne preporucuje, obzirom da C ima ugradjene funkcije koje pocinju 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 pocnemo koristiti varijable moramo ih definisati. Definisanje ili deklaracija varijabli, nije nista drugo nego informisanje C-a da nam je potreban prostor u memoriji racunara, koji zelimo rezervisati za pohranjivanje nasih podataka. Gledajte na to kao rezervisanje stola u restoranu, prije nego imate veceru sa prijateljima, potrebno je prethodno nazvati restoran i rezervisati sto. Sto predstavlja prostor u restoranu ili kontejner koji cemo koristiti za postavljanje podataka u obliku hrane, dok opet stolice su razlicite vrste varijabli koje ce sluziti za postavljanje podataka u obliku vasih prijatelja razlicite tezine i oblika. Nadam se da ste stvorili sliku 😃.

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

Prethodni kod predstavlja nacin definisanja tri promjenjive: odgovor, kolicina i cijena. Ove promjenjive mogu prihvatiti/pohraniti/sadrzati tri razlicite vrste tipova podataka: karakter, cijeli broj i decimalni broj respektivno.
U slucaju da prethodno nismo definisali varijable, ne bi bili u mogucnosti koristiti kasnije u nasem programa, dakle varijable se prvo moraju definisati da bi se koristile.
Uvijek mozete definisati vise od jedne varijable istog tipa u istom programu.

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

```
main(){
    /* moje varijable, napisane u jednoj liniji
       jer su istog tipa */
    char prvoSlovoImena, prvoSlovoPrezimena;
    /*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 koristenje **lokalnih varijabli**, vise detalja u nastavku kursa.

Dodjeljivanje vrijednosti varijabli

Konacno, operator dodjele **=** nam sluzi da dodijelimo/pohranimo vrijednost u nasu promjenjivu. Iako ovo zvuci 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

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

int main(){
    // definisanje varijabli
    char prvoSlovoImena, prvoSlovoPrezimena;
    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 uceniku
    prvoSlovoImena = 'G';
    prvoSlovoPrezimena = 'C';
    broj_olovki = 4;
    broj_sveski = 5;

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

    // informacije o drugom uceniku
    prvoSlovoImena = 'B';
    prvoSlovoPrezimena = 'V';
    broj_olovki = 3;
    broj_sveski = 2;

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

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

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

    printf("%c.%c. treba %d olovki, %d sveski i jedan obrok dnevno.\n",
           prvoSlovoImena, prvoSlovoPrezimena, broj_olovki, broj_sveski);
    printf("Ukupna cijena potrepstina za ucenika %c%c je %.2fKM\n\n",
           prvoSlovoImena,prvoSlovoPrezimena,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 komplena 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 intenzinvno 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 duzini stringa prilikom programiranja u C-u, kao rezultat dobicemo vrijednost duzine stringa koja ne ukljucuje string terminator. Iako C koristi **null zero** kako bi znao kada je kraj stringa, isto tako zna da u slucaju racunanja duzine stringa, prilikom ispisa, **null zero** nije ukljucen u konacan rezultat.

Svi navedeni stringovi su duzine 6 karaktera.

```
"Goku ."
"Bulma."
"Krilin"
```

Medjutim, kada planiramo i definisemo varijable u koje smjestamo stringove moramo voditi racuna i ukljuciti **string terminator** prilikom odredjivanja duzine varijable. Najlakse je zapamtiti na nacin kada definisemo jedan jedini karakter i smijestamo ga u varijablu, cak i tad ovaj karakter ima duzinu od dva karaktera u memoriji racunara 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 mnogin 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 koristenje 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

```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 napomenti 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

```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

```
September
Mart
```

Svaki pojedinacni dio niza karaktera se naziva elemntom. Niz mjesec ima ukupno deset elemenata, koji su precizno odredjeni subscripts-ima. Subscripts su brojevi koji odredjuju na kojoj poziciji se nalazi element niza. Svaki niz pocinje o 0, sto znaci da se i nula broji iz perspektive C-a. Svaki element naseg niza je karakter, sto znaci da niz ili lista karaktera cini string.

Stampanje niza je jednostavno kao i stampanje pojedinacnog stringa

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

Inicijalizacija stringa (strcpy())

Kako smo vec naucili prilikom definisanja stringa dovoljno je reci naziv i nakon toga u uglastu zagradu staviti broj koji odredjuje ukupnu duzinu stringa. Takodje, ako nismo sigurni kolika je duzina naseg stringa unaprijed, mozemo ostaviti uglaste zagrede prazne. U oba slucaja konacan rezultat ce biti isti.

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

U drugom slucaju C racuna broj karaktera u stringu i na kraju dodaje null zero. Naravno posto je ovo inicijalna kreacija od C-a, racunajuci i null zero karakter nasa promjenjiva mjesec ce do kraja programa moci da pohrani ukupno 7 karaktera u nizu (ukljucujuci null zero).

Ali sta ako ne zelimo vrsiti inicijalizaiju stringa u jednoj liniji kao do sad? Recimo da zelimo uraditi nesto slicno kao

```
char mjesec[10];      <-- definisanje varijble mjesec,
                          niz karaktera  ili string

mjesec = "Januar";    <-- inicijalizaija varijble,
                          dodjeljivanje vrijednosti
                          NIJE DOZVOLJENO NA OVAJ NACIN
```

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

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

U slucaju da ipak zelimo kasnije kroz nas program izmijeniti ovu vrijednost, odnosno definisati novu ili inicjalizirati novu vrijednost imamo dvije opcije. Jednu smo vec vidjeli u primjeru kada smo preko subscripts-a (indeksa) pristupili svakom karakteru posebno, sto je priznacete naporno. Drugi, elegantniji nacin, jeste funkcija koju nam nudi C, a koja se zove **strcpy()** (skraceno od string copy). Ova funkcija prihvata dva parametra, i prilikom pozivanja dodijeljuje, kopira, mapira, ono sto se nalazi u drugom parametru na desnoj strani u prvi parametar na lijevoj strani. strcpy() automatski dodaje null zero na kraju stringa, ali takodje moze smjestiti samo onu velicinu stringa u memoriju koja je prethodno definisana.

Izvorni kod: kod-19_inicijalizacija-stringa.c

```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 ukljucili novi header fajl (**#include <string.h>**), kako bi smo omogucili koristenje funkcije strcpy().

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

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

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


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

        // karakter_3 moze pohraniti naziv karaktera
        // od ukupno 29 karaktera, 30. sluzi 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 zavrsava

    strcpy(karakter_3, "Picolo");
        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.
Krilin dolazi sa planete Zemlja.
Picolo 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 subscripts pocinje sa nulom, dakle C pocinje brojanje od nule, ukjlucujuci 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 inicijalizaiju
2. element po element
3. koristenjm 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 sluzi za dodavanje, ukljucivanje (include ili ukljuci, uvezi iz vana) eksternih datoteka koje sadrze prethodno definisane funkcije neophodne za rad sa nasim programom. Kada god C, odnosno compiler, naidje na znak taraba (#) ovo je za njega okidac da se radi o pre-procesorkoj direktivi i u skladu sa tim se i ponasa. Pre-procesorska direktiva se ne izvrsava za vrijeme pokretanja programa (runtime), vec za vrijeme kompajliranja programa.

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

```
#include
#define
```

Ukljucivanje 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 uljucivanje sadrzaja nekog drugog fajla unutar naseg programa. Ovo ukljucivanje se desava samo za vrijeme kompajliranja naseg programa. Naravno ukljuceni kod je sastavljen od komentara ili opet od samog C koda.
Na ovaj nacin mozemo ukljuciti vec predefinisane/ugradjene funkcije (built-in funkcije) u C-u, kao

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

za potrebe koristenja funkcije printf() ili

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

za potrebe koristenja strcpy(), ali i more drugih. Built-in funkcije dolaze sa instalacijom naseg kompajlera. Ovi fajlovi se jos zovu i header fajlovi (header zaglavlje, jer se nakon ukljucivanja nalaze na pocetku naseg fajla, znaci u samom zaglavlju) i pravilo je da im se daje ekstenzija **.h**. Naravno #include se moze koristiti bilo gdje u nasem C kodu ali je preporuka da se koristi na pocetku pisanja izvornog koda.
Ako zelimo ukljuciti funkcije koje smo samostalno kreirali sintaksa ove direktive umjesto znakova manje vece,

koristi navodnike. Dakle, ispravno ukljucivanje naseg header fajla u program, koji se nalazi u istom direktoriju gdje i nas izvroni kod bi i zgledalo 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 sluzi za definisanja konstanti. Konstanta je isto sto i literalna vrijednost. 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 vrijednoscu 99
#define PI 3.14             <-- gdje god se pojavai konstanta PI
                                zamijeni vrijednoscu 3.14
#define KARAKTER_IME "Goku" <-- gdje god se pojavi konstanta KARAKTER_IME
                                zamijeni vrijednoscu "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 izvrsava 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 ukljucivanje u program

Kreirajmo sada program primjenjuci sve do sad nauceno.

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 ukljucuje
   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] = "Krilin";

    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, naucili 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 naucili 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 nastavljanje izvrsavanja 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 sluzi 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

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

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

```
    char prvoSlovoImena;
    char prvoSlovoPrezimena;
    int godine_karaktera;
    int omiljeni_broj;

    printf("Prvo slovo vaseg imena: ");
    scanf(" %c", &prvoSlovoImena);

    printf("Prvo slovo vaseg prezimena: \n");
    scanf(" %c", &prvoSlovoPrezimena);

    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, prvoSlovoPrezimena, godine_karaktera);
    printf("Vas omiljeni broj je: %d\n", omiljeni_broj);
}
```

Rezultat

```
Prvo slovo vaseg imena: G
Prvo slovo vaseg prezimena:
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 vavijabli 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 zvucalo sebicno, C je napisan na takav nacin i radi od kad znamo za kompjutere, tako da je svaka implementacija uradjena sa namjenom a na name je da se prilagodimo i naucimo kako se koristi.

Prvi od problema sa kojim se susrecemo prilikom koristenja scanf() funkcije, jeste da korisnik mora unijeti tacno ono sto scanf() od njega ocekuje. U slucaju da korisnik unese realan broj (float) na mjestu gdje funkcija scanf() po definiciji ocekuje cijeli broj (integer) doci ce problema prilikom izvrsavanja ili u drugim slucajevima do netacnog rezultata. Ovo moze biti veoma opasno kada su financijski proracuni u pitanju. Za sad cemo predpostaviti da ce korisnik svaki put unijeti tacno ono sto se od njega ocekuje. O prevazilazenju ovog problema ce biti rijeci u nokoj od narednih lekcija.

Takodje, postoji izuzetak pravilu prilikom koristenja ampersand **&** znaka, kada je u pitanju scanf() funkcija, a to je kada trazimo od korisnika da unese niz karaktera ili string (%s), recimo u slucaju da trazimo unos imena ili prezimena. Ovo za sad zapamtite na nacin, u sljucaju da trazite od korisnika da unese cijeli broj (integer), decimalni broj (float ili double), pojedinacni karakter (char) **moramo staviti ampersand & ispred varijable** prilikom definisanja scanf() funkcije. U slucaju da trazimo 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 slucajno, ali o ovome ce biti rijeci kasnije kada budemo demistifikovali pointer-e u C programskom jeziku 😄.

Jos jedan od problema na koje nailazimo prilikom koristenja funkcije scanf(), jeste da scanf() prestaje da cita unos stringa nakon sto naidje na prvi razmak ili prazno polje. Na ovaj nacin smo ograniceni unosom samo jedne rijeci koju ce scanf() da prihvati sve nakon razmaka ce biti odsijeceno ili ignorisano. Dakle ako korisnik unese ime i prezime, scanf() ce procitati samo ime i smjestiti u promjenjivu koju smo prethodno definisali. Za sad problem mozemo prevazici na nacin da nakon funkcije print(), gdje trazimo unos imena i prezimena, definisemo dvije nove funkcije scanf(), jednu za ime drugu za prezime, ali kako predpostavljate ovo se rijetko kad dobro zavrsi 😉.

Pitate se zasto uopste koristimo funkciju scanf() kada ima toliko "nedostatka"? Zato sto je jednao 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

```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 mjeces, 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 mjeces, 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.
```