Programiranje 1
5. predavanje — dodatak

Saša Singer

singer@math.hr

web.math.pmf.unizg.hr/~singer

PMF – Matematički odsjek, Zagreb

Sadržaj predavanja — dodatka

- Primjeri programa kroz Code::Blocks:
  - Prvi program — "Hello world".
  - Primjer 2 — učitaj, izračunaj, ispiši (int).
  - Primjer 3 — učitaj, izračunaj, ispiši (double).

Primjeri programa kroz Code::Blocks

Prvi program — "Hello world"

Primjer 1. Standardni prvi C program u većini knjiga izgleda (otprilike) ovako:

```
#include <stdio.h>

  /* Glavni program - funkcija main. */

int main(void)
{
    printf("Dobar dan.\n");
    return 0;
}
```

Što radi ovaj program?

Prvi program — svrha

Iskreno, ništa jako pametno:

- ispisuje tekst `Dobar dan.` na standardni izlazni uređaj.

Sjetite se, svaki program (algoritam) mora imati neki izlaz.

- Naš program ima samo to i ništa više!

Dakle, to je (skoro) "najmanji" mogući program:

- napiši zadani tekst.

Jedini "višak" u programu je komentar (v. malo kasnije).

Program je vrlo jednostavan, ali potpun, u smislu da se može

- korektno prevesti i izvršiti,

bez grešaka!

Prvi program — Unix okruženje

Pod Unixom, treba napraviti sljedeće:

- Utipkati tekst programa (u nekom editoru) i spremiti ga u neku datoteku — recimo, `prvi.c`,
- Pozvati C prevoditelj (recimo, `cc`) naredbom
  - `cc prvi.c`
- Prevoditelj prevodi program u objektni kôd, sam poziva linker koji uključuje standardnu biblioteku i kreira izvršni kôd u datoteci `a.out` (jer nismo drugačije rekli).
- Program izvršavamo tako da utipkamo naredbu
  - `./a.out`
- Rezultat izvršavanja je (prema očekivanju) ispis poruke
  - `Dobar dan.`

Prvi program — Code::Blocks

Na Windowsima, ako želimo raditi u Code::Blocks okolini,

- prvo treba startati Code::Blocks.

Zatim, treba redom:

- Odabrati File (na vrhu), pa New — Empty file, jer želimo utipkati tekst novog programa.
- Otvorit će se prozor za unos teksta programa, u kojeg treba utipkati tekst programa.
- Kad ste gotovi, vrlo je zdravo spremiti taj tekst u neku datoteku:
  - File, Save file, izaberite mapu i ime datoteke.

  Na primjer, `prog_1.c`.

Prvi program — Code::Blocks (nastavak)

Ako odmah želite "obojani" tekst (tzv. syntax highlighting), onda postupak ide ovako:

- Odabrati File (na vrhu), pa New — File . . . .
- U prozoru New from template treba redom:
  - Izabrati (kliknuti) C/C++ source, pa Go,
  - onda Next (ili isključite tu stranicu),
  - izabrati C, pa Next,
  - upisati puni put i ime datoteke, ili preko . . . izabrati mapu i upisati ime datoteke, pa Finish.
- Sad (na)pišete program, a spremite ga ovako:
  - File, Save file.

Prvi program — Code::Blocks (dodatak)

Ja neću tako, jer

- već imam gotov tekst programa u datoteci `prog_1.c`.

U tom slučaju, radim sljedeće:

- Idem na File (na vrhu), pa Open ... ,
- U prozoru Open file "prošetam" do mape (foldera) gdje je moj program,
- izaberem ("kvrčnem") moju datoteku `prog_1.c`
- i kažem Open.

Sad imam moj tekst programa u prozoru za editiranje.

Prvi program — Code::Blocks (dodatak)

Nakon toga, želim pozvati C prevoditelj. To radim tako da

- odem na Build (na vrhu), pa kažem Compile current file (može i Build, ili Build and run).

Ako ima grešaka, onda

- dobijem "što me ide" u obliku poruka o greškama.
- U prozoru na dnu, pod Build log, mogu pročitati detalje o greškama.

Onda moram popraviti program, pa iznova . . . .

Ako nema grešaka (a neće ih biti!), onda

- dobijem poruku da je sve "dobro prošlo": 0 errors, 0 warnings.

Prvi program — Code::Blocks (dodatak)

Prevoditelj je (u međuvremenu)

- preveo program u objektni kôd,
- pozivao linker koji uključuje standardnu biblioteku
- i kreirao izvršni kôd u datoteci `prog_1.exe`.

Ta datoteka se nalazi na istom mjestu gdje je i tekst programa (ako nismo drugačije rekli).

Sljedeći korak je izvršavanje mog programa.

To mogu napraviti na dva načina:

- brži način je direktno iz Code::Blocks okoline,
- ili preko posebnog komandnog prozora, kao na Unixu.

Prvi program — Code::Blocks (dodatak)

Izvršavanje programa direktno iz Code::Blocks ide tako da

- odem na Build (na vrhu), pa kažem Run.

Tad se dogodi sljedeće:

- otvori se komandni prozor (Command prompt) za komunikaciju između programa i nas, tj. za ulaz/izlaz,
- u prozoru piše izlaz našeg programa — tekst `Dobar dan.`
- i poruka o trajanju izvršavanja.

Komandni prozor se gasi bilo kojom tipkom.

Prevođenje i izvršavanje programa može i zajedno, u "paketu":

- Build (na vrhu), pa Build and run.

Prvi program — Code::Blocks (dodatak)

"Spori" način za izvršavanje programa (kao na Unixu), kojeg nitko ne koristi, osim "po kazni":

- Otvorim Command prompt, odem do mape (foldera) gdje je izvršni program i utipkam naredbu
  - `prog_1`
- Rezultat izvršavanja je (kako i treba) ispis poruke
  - `Dobar dan.`

Probajte sami da ovo korektno radi.

Opis prvog C programa (dodatak)

C program sastoji se od funkcija i varijabli. Funkcije sadrže instrukcije koje određuju koje će operacije biti izvršene. Varijable služe pamćenju podataka u memoriji računala.

- Izvršavanje programa počinje funkcijom `main`. Funkcija s tim imenom `main` ("glavna") mora biti prisutna u svakom programu.
- Svaki objekt (funkcija, varijabla) koji koristimo u programu mora biti korektno deklariran prije upotrebe.
- To vrijedi i za sve funkcije iz standardne C biblioteke koje trebamo u programu.
- Nama treba funkcija `printf` za izlaz — formatirano pisanje podataka, pa moramo nekako navesti pripadnu deklaraciju.

Opis prvog C programa (dodatak)

Zato program započinjemo tzv. makro naredbom

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

Ta naredba uključuje (engl. `include`) u program

- datoteku `stdio.h` koja sadrži deklaraciju funkcije `printf` (i mnogih drugih funkcija za ulaz/izlaz).
- Datoteke s nastavkom `.h` nazivaju se datoteke zaglavlja (engl. header files).
- Navođenje imena datoteke između znakova `<` i `>` kaže da se radi o standardnoj datoteci zaglavlja — koja dolazi uz C prevoditelj, a ne o "našoj".

Makro naredbe su naredbe tzv. pretprocesoru.

Opis prvog C programa (dodatak)

Striktno govoreći, makro naredbe nisu dio jezika C,

- iako su sastavni dio teksta C programa.

Zato za njih ne vrijede pravila pisanja naredbi u C-u.

- Počinju znakom `#` na početku reda,
- cijeli tekst do kraja reda smatra se makro naredbom
- i ne završavaju točka-zarezom `;`.

Makro naredbe "izvršava" pretprocesor,

- promjenama u tekstu programa, prije nego što taj tekst ode prevoditelju na prevođenje.

U ovom slučaju, makro naredba `include` se zamijenjuje stvarnim tekstom iz navedene datoteke zaglavlja `stdio.h`.

Opis prvog C programa (dodatak)

Druga i četvrta linija programa su prazne.

Takva linija sadrži samo znak za prijelaz u novi red, koji se interpretira kao razmak (bjelina, praznina).

Razmaci ili "praznine" (engl. blank) služe odvajanju pojedinih riječi ili drugih cjelina u jeziku.

- Razmaka smije biti i više, uključivo i prazne linije teksta.
- Svrha "viška" razmaka je povećanje čitljivosti programa — i treba ih koristiti!
- Prevoditelj preskače (ignorira) "višak" razmaka.

Opis prvog C programa (dodatak)

Treća linija programa

```
/* Glavni program - funkcija main. */
```

osim razmaka na početku reda, sadrži i komentar.

Komentar je bilo koji tekst zatvoren između susjednog para od po 2 znaka:

- `/*` — za početak komentara i
- `*/` — za kraj komentara.

Svrha komentara je (valjda) očita, a prevoditelj ih preskače (ignorira).

Opis prvog C programa (dodatak)

Sljedeća (peta) linija programa je deklaracija funkcije `main`

```
int main(void)
```

Slično kao i u matematici, funkcija

- može imati jedan ili više argumenata (ili parametara)
- i, obično, vraća neku vrijednost.

U C-u svaka funkcija ima svoje ime. Opis domene i kodomene:

- "Domenu" zadajemo deklaracijom argumenata unutar okruglih zagrada `(` i `)`, iza imena funkcije.
- "Kodomenu" zadajemo navođenjem tipa povratne vrijednosti funkcije, ispred imena funkcije.

Opis prvog C programa (dodatak)

U našem primjeru, funkcija `main` nema niti jedan argument.

- To deklariramo tako da u okrugle zagrade `(` i `)`, gdje inače deklariramo argumente, stavimo ključnu riječ `void` (engl. void = prazan).
- Zagrade `(` i `)` se moraju napisati i kad nema argumenata!

Tip povratne vrijednosti u našem primjeru je cijeli broj.

- To je označeno s `int` na početku, ispred imena funkcije.

Napomene.

- Funkcija `main` vraća operacijskom sustavu cjelobrojnu vrijednost koja ima značenje izlaznog statusa programa.
- Nula se interpretira kao uspješni završetak, a svaka druga vrijednost kao završetak usljed greške.

Opis prvog C programa (dodatak)

Iza deklaracije funkcije dolazi tzv. tijelo funkcije.

```
{
    printf("Dobar dan.\n");
    return 0;
}
```

Tijelo funkcije ima strukturu bloka (v. ranije). Sastoji se od

- deklaracija/definicija objekata (varijabli i funkcija),
- naredbi
- i neimenovanih blokova,
- zatvorenih unutar vitičastih zagrada `{` i `}`.

Savjet. Vitičaste zagrade pišite tako da budu dobro vidljive.

Opis prvog C programa (dodatak)

Dodatna pravila (ponavljanje):

- Svaka definicija/deklaracija i naredba mora završavati znakom točka–zarez `;` .
- Blok ne završava znakom `;` . Preciznije, kad `}` označava kraj bloka, onda se iza `}` ne piše `;` .

Napomena. Znak `}` može se pojaviti i s drugim značenjima (na pr. inicijalizacija polja). Tad se (katkad) piše `;` iza `}` .

U našem primjeru, nema deklaracija i neimenovanih blokova, već tijelo funkcije sadrži samo dvije naredbe:

- poziv funkcije `printf` — za ispis stringa,
- naredbu `return`.

Opis prvog C programa (dodatak)

Prva naredba

```
printf("Dobar dan.\n");
```

je poziv standardne funkcije `printf` za formatirani ispis.

Prvi, a u našem slučaju i jedini, argument funkcije je

- tzv. znakovni niz ili string.

Piše se kao

- niz znakova, zatvoren između dvostrukih navodnika " .

Svi znakovi u tom nizu

- koji nisu dio tzv. oznaka konverzije,

doslovno se prepisuju na izlazni uređaj.

Opis prvog C programa (dodatak)

Oznake konverzije počinju znakom `%`, a završavaju nekim od dozvoljenih znakova — poput `d` ili `g`, i imaju posebno značenje (v. drugi program).

Naš string `"Dobar dan.\n"` ne sadrži oznake konverzije (nema `%`). To znači da se doslovno ispisuje sadržaj stringa

- niz znakova: `Dobar dan.\n`.

Napomena. Funkcija `printf` nakon završenog ispisa ne prelazi "sama" u novi red.

- To se postiže specijalnim znakom `\n` za prijelaz u novi red (engl. "newline" znak).
- Specijalni znakovi u C-u pišu se tako da počinju znakom `\`.

Opis prvog C programa (dodatak)

Funkcija `printf` može imati i više od jednog argumenta.

- Prvi argument je string (znakovni niz), kao i kod nas. Taj niz zove se kontrolni string.
- Ostali argumenti su, općenito, izrazi, a ispisuju se vrijednosti tih izraza (nakon što se izračunaju).

U tom slučaju,

- kontrolni string mora sadržavati oznake konverzije koje zadaju način (format) ispisa pojedinih vrijednosti.

Opis prvog C programa (dodatak)

Druga i zadnja naredba je

```
return 0;
```

Naredba return završava izvršavanje funkcije.

- Ako funkcija treba vratiti neku vrijednost, ta vrijednost navodi se u `return` naredbi — iza riječi `return`.

U našem primjeru, funkcija `main` vraća nulu, pa je zadnja naredba programa `return 0;` . Tom naredbom

- završava se izvršavanje cijelog programa,
- a povratna vrijednost `0` je signal operacijskom sustavu da je program uspješno završio.

Prvi program — još malo

Zadatak. Probajte što radi prvi program kad izbrišemo `\n` na kraju stringa u pozivu funkcije `printf`.

Zadatak. Sljedeći program radi isto kao i prvi. Probajte!

```
#include <stdio.h>

int main(void)
{
    printf("Dobar ");
    printf("dan.");
    printf("\n");
    return 0;
}
```

Primjer 2 — učitaj, izračunaj, ispiši (int)

Primjer 2. Napišite program koji

- učitava dva cijela broja a, b (tipa `int`),
- računa vrijednost izraza $3a^2 - b$ i sprema tu vrijednost u varijablu c,
- a zatim ispisuje vrijednost te varijable c.

Ovo je ponešto kompliciraniji program od prvog, jer sadrži ulaz podataka, računanje izraza i ispis rezultata.

Tekst programa spremljen je u datoteci `prog_2.c`.

Drugi program — tekst

```
#include <stdio.h>

int main(void) {
    int a, b, c;

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

    c = 3 * a * a - b;

    printf(" Rezultat = %d\n", c);

    return 0;
}
```

Drugi program — opis (dodatak)

Napomena. Poznate stvari iz prvog programa nećemo ponovno opisivati.

Prva nova stvar je čitanje podataka — ulaz u program.

Bilo koju učitanu vrijednost moramo negdje spremiti, da bismo ju kasnije mogli koristiti. Gdje?

- U memoriju računala, tj. na neku adresu u memoriji.

C ne dozvoljava da sami biramo adrese na koje spremamo podatke. Umjesto toga, možemo koristiti tzv. varijable.

Varijable su simbolička imena za lokacije u memoriji

- u koje možemo spremiti vrijednosti odgovarajućeg tipa.

Drugi program — opis (dodatak)

Takva imena moramo deklarirati (uvesti u program) na odgovarajući način.

```
int a, b, c;
```

Ova deklaracija uvodi tri nove varijable u funkciju `main`.

- Imena tih varijabli su `a`, `b` i `c`,
- a "u njih" možemo spremiti vrijednosti tipa `int`.

Kad "pročita" ovu deklaraciju, C prevoditelj

- sam rezervira potreban prostor u memoriji za svaku od ovih varijabli.
- Veličina potrebnog prostora određena je tipom varijable.

Drugi program — opis (dodatak)

Zato se prvo piše tip (jer određuje veličinu prostora), a zatim imena varijabli — odvojena zarezima, ako ima više imena.

Standardno, tip `int` zauzima 4 bajta, tj. prevoditelj rezervira po 4 bajta za svaku od varijabli `a`, `b`, `c`.

Napomene.

- Prevoditelj sam "bira" i dodjeljuje adrese za pojedine varijable.
- To znači da adrese tih varijabli ne znamo unaprijed.
- Međutim, nakon deklaracije bilo koje varijable, možemo saznati i koristiti njezinu adresu — korištenjem tzv. adresnog operatora `&`.

Na primjer, `&a` je adresa varijable `a`.

Drugi program — opis (dodatak)

Vrijednost varijable je

- trenutni sadržaj spremljen na pripadnoj adresi,

s tim da se sadržaj, kao niz bitova, interpretira kao vrijednost zadanog tipa.

Do trenutne vrijednosti varijable dolazimo navođenjem imena varijable. Na primjer, kad nadalje koristimo ime `a`,

- to je sinonim za trenutnu vrijednost varijable `a`,

tj. za trenutni sadržaj na pripadnoj adresi.

Bitno. Kod deklaracije varijable, prevoditelj samo rezervira prostor,

- ali ne sprema nikakvu vrijednost (sadržaj) u taj prostor,

osim ako ne zatražimo drugačije.

Drugi program — opis (dodatak)

To "drugačije" moguće je napraviti tako da inicijaliziramo varijablu prilikom deklaracije (v. kasnije).

U našem slučaju, varijable nisu inicijalizirane, jer nismo zadali neku "početnu vrijednost" u deklaraciji.

To praktično znači da

- vrijednosti varijabli `a`, `b` i `c` (još) nisu definirane!

Sadržaj na tim adresama je "neko smeće" ostalo od ranije u memoriji.

Uočite da namjerno nismo zadali neke fiksne početne vrijednosti za `a` i `b`, jer te vrijednosti

- želimo učitati prilikom izvršavanja programa.

Drugi program — opis (dodatak)

Naredba

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

je poziv standardne funkcije `scanf` za formatirano čitanje podataka.

Ova naredba

- učitava dvije cjelobrojne vrijednosti,
- po oznakama konverzije `%d` (`d` = decimal),
- i učitane vrijednosti dodjeljuje, redom, varijablama `a` i `b`.

Funkcija `scanf` deklarirana je u datoteci zaglavlja `stdio.h` (kao i funkcija `printf`).

Drugi program — opis (dodatak)

Detaljniji opis rada funkcije `scanf` za formatirano čitanje.

Funkcija stvarno čita neki niz znakova s ulaza,

- pretvara (ili konvertira) određene dijelove tog niza u vrijednosti odgovarajućih tipova,
- i dodjeljuje te vrijednosti odgovarajućim argumentima.

Prvi argument funkcije `scanf` je kontrolni string koji opisuje

- tzv. "format" učitavanja znakova i način pretvaranja tih znakova u vrijednosti odgovarajućih tipova.

Svi argumenti iza toga moraju biti pokazivači, tj.

- adrese varijabli na koje treba spremiti učitane vrijednosti (prema zadanom formatu).

Drugi program — opis (dodatak)

Način pretvaranja znakova u vrijednost odgovarajućeg tipa zadaje se tzv. oznakom konverzije u kontrolnom stringu.

- Svaka oznaka konverzije počinje znakom `%`,
- a završava nekim od dozvoljenih znakova konverzije, poput `d` ili `g`.

U našem pozivu funkcije `scanf`, kontrolni string `"%d%d"` sadrži samo dvije oznake konverzije `%d`.

- Znak konverzije `d` služi za "decimalno" čitanje (i pisanje) cjelobrojne vrijednosti s predznakom.
- Prva oznaka konverzije odgovara prvom sljedećem argumentu iza kontrolnog stringa, druga oznaka drugom sljedećem argumentu, i tako redom.

Drugi program — opis (dodatak)

Dakle, precizniji opis naredbe `scanf("%d%d", &a, &b);` je:

- prva učitana vrijednost (po prvoj oznaci `%d`) sprema se (kao sadržaj) u prostor zadan adresom varijable `a`,
- a druga učitana vrijednost (po drugoj oznaci `%d`) sprema se u prostor zadan adresom varijable `b`.

Varijable `a` i `b` su tipa `int`, i po tipu odgovaraju oznakama konverzije `%d` za čitanje vrijednosti tipa `int`.

Zapamtite: funkcija `scanf` iza kontrolnog stringa mora dobiti

- adrese — kamo treba spremiti učitane vrijednosti!

Zato drugi argument u pozivu nije `a`, već `&a`. Isto tako, treći argument nije `b`, već `&b`.

Ovdje koristimo adresni operator `&` koji daje adresu varijable.

Drugi program — opis (dodatak)

Razlog za ovu "čaroliju" s adresama je

- način prijenosa argumenata (ili parametara) u funkciju,

pri pozivu funkcije.

U C-u postoji samo tzv. prijenos argumenata po vrijednosti.

- Stvarni argumenti koje pišemo u pozivu funkcije su, općenito, izrazi.

Kod poziva bilo koje funkcije,

- prvo se izračunaju vrijednosti tih izraza,
- a zatim se te vrijednosti kopiraju u odgovarajuće "lokalne" varijable u toj funkciji.

Zato funkcija ne može promijeniti vrijednost argumenta (ako to proba napraviti — mijenja lokalnu kopiju).

Drugi program — opis (dodatak)

Međutim, ako pošaljemo adresu neke varijable,

- onda funkcija ne može promijeniti tu adresu (radi s lokalnom kopijom adrese),
- ali smije spremiti neki sadržaj na to mjesto i tako promijeniti vrijednost te varijable!

Zato ne smije pisati `a` u pozivu funkcija `scanf`, jer

- varijabla `a` još niti nema vrijednost,
- a i da ima — ne valja, jer ju `scanf` ne može promijeniti.

Naime, `scanf` mora dodijeliti vrijednost toj varijabli `a`, tj. pročitati neku vrijednost i spremiti ju na adresu te varijable.

Onda mora stići "vrijednost" adrese `&a`, da se može nešto tamo spremiti.

Drugi program — opis (dodatak)

Kako radi pretvaranje znakova s ulaza u cjelobrojnu vrijednost tipa `int` po oznaci konverzije `%d`?

- Vodeće bjeline se preskaču. To uključuje tabulatore i znakove za prijelaz u novi red.
- Zatim se čita najdulji mogući niz znakova koji odgovara decimalnom zapisu cjelobrojne konstante.
  - Dozvoljen je predznak i dekadske znamenke `0`, ... ,`9`.
- Taj niz znakova se "znak–po–znak" pretvara u cjelobrojnu vrijednost tipa `int` (po Hornerovom algoritmu, v. kasnije).
- Čitanje se prekida ispred prvog sljedećeg znaka. Taj znak se "pogleda", ali ne učita!

Drugi program — opis (dodatak)

Sljedeća naredba

```
c = 3 * a * a - b;
```

je naredba pridruživanja ili dodjeljivanja.

Opći oblik te naredbe je `varijabla = izraz;` . Znak `=` je

- operator pridruživanja (ili dodjeljivanja) vrijednosti.

Oprez: ovaj operator nije simetričan. Operator jednakosti u C-u se piše kao `==` .

Naredba pridruživanja izvršava se na sljedeći način:

- prvo se računa vrijednost izraza na desnoj strani,
- a zatim se ta vrijednost dodjeljuje navedenoj varijabli.

Drugi program — opis (dodatak)

Izraz smije sadržavati operande i operatore, a

- izračunava se po pravilima prioriteta operatora, slično kao u matematici.

U C-u postoje još i pravila asocijativnosti za operatore, ali o tome više — malo kasnije.

Naš izraz `3 * a * a - b` sadrži samo

- standardne aritmetičke operatore `*` i `-`,

koji imaju standardna pravila prioriteta (`*` je iznad `-`).

Napomena.

- U C-u ne postoji poseban operator za potenciranje.

U matematičkoj biblioteci postoji funkcija `pow` (v. kasnije).

Drugi program — opis (dodatak)

Uobičajeni i najjednostavniji operandi su

- konstante i varijable.

Možemo imati i pozive funkcija, podizraze (opet, v. kasnije).

Na primjer, operandi u izrazu `3 * a * a - b` su

- konstanta `3`, varijabla `a` (dva puta) i varijabla `b`.

U postupku računanja vrijednosti izraza, kad je operand varijabla, na tom mjestu

- koristi se ("uvrštava") trenutna vrijednost te varijable.

To odgovara ranije rečenom: ime varijable je sinonim za trenutnu vrijednost varijable (sadržaj na pripadnoj adresi).

Drugi program — opis (dodatak)

Naredba

```
printf(" Rezultat = %d\n", c);
```

je poziv standardne funkcije `printf` za formatirano pisanje.

Ova funkcija piše neki niz znakova na izlaz.

Prvi argument funkcije `printf` je kontrolni string koji opisuje

- "format" pisanja znakova i način pretvaranja vrijednosti ostalih argumenata u odgovarajuće nizove znakova.

Svi argumenti iza toga, ako ih ima, moraju biti izrazi.

- Pišu se vrijednosti tih izraza (nakon što se izračunaju), prema odgovarajućim oznakama konverzije u kontrolnom stringu.

Drugi program — opis (dodatak)

Sasvim općenito, kontrolni string može sadržavati dvije vrste znakova:

- obične znakove — koji se doslovno prepisuju na izlaz,
- i oznake konverzije koje zadaju način (format) ispisa vrijednosti preostalih argumenata.

Kao i kod funkcije `scanf`,

- prva oznaka konverzije odgovara prvom sljedećem argumentu iza kontrolnog stringa, druga oznaka drugom sljedećem argumentu, i tako redom.

Drugi program — opis (dodatak)

Naš kontrolni string sadrži jednu oznaku konverzije `%d`. Ona kaže da

- vrijednost prvog sljedećeg argumenta treba ispisati kao decimalni cijeli broj (s predznakom, ako ga ima).

Prvi sljedeći argument je izraz `c`,

- a vrijednost tog izraza je trenutna vrijednost varijable `c`.

Dakle, umjesto oznake `%d`, ispisuje se niz znakova

- koji odgovara dekadskom zapisu vrijednosti cjelobrojne varijable `c`.

Svi ostali znakovi iz kontrolnog stringa doslovno se prepisuju.

Drugi program — opis (dodatak)

Na primjer, ako je vrijednost varijable `c` jednaka `25`, naš poziv funkcije `printf` ispisuje niz znakova:

- `Rezultat = 25`

Ovdje se baš i ne vidi dobro, ali

- prvi znak (ispred `R`) je praznina (blank),
- a zadnji znak je prijelaz u novi red (znak `\n`).

Ponekad ćemo prazninu (blank, razmak) pisati kao

- posebni znak ␣ — da se bolje vidi.

Da ne bude zabune, to je i dalje "obična" praznina!

Drugi program — izvršavanje i rezultat

Kad pokrenemo program u Code::Blocks, otvori se komandni prozor u kojem se ništa ne događa!

- U stvari, program uredno radi, ali čeka nas da upišemo vrijednosti za `a` i `b`.

Zato je vrlo korisno, prije svakog čitanja, ispisati neki tekst koji kaže što se od nas očekuje (v. treći program).

Kad (na ulazu) napišemo niz znakova:

- `3␣2` i stisnemo `ENTER`,

dobivamo izlaz (opet niz znakova):

- `Rezultat = 25`

Nevjerojatno, ali radi! Provjerite!

Ulaz je u datoteci `prog_2.in`, a izlaz u `prog_2.out`.

Drugi program — još malo

Zadatak. Program možemo napisati i tako da odmah ispišemo vrijednost izraza, bez spremanja u varijablu c.

```
#include <stdlib.h>

int main(void) {
    int a, b;

    scanf("%d%d", &a, &b);
    printf(" Rezultat = %d\n", 3 * a * a - b);

    return 0;
}
```

Primjer 3 — učitaj, izračunaj, ispiši (double)

Primjer 3. Napišite program koji

- učitava dva realna broja x, y (tipa `double`),
- računa vrijednost izraza $2x^2 - y^3$ i sprema tu vrijednost u varijablu z,
- a zatim ispisuje vrijednost te varijable z.

Osnovna razlika između ovog i prethodnog programa je u tipu podataka s kojim radimo. Tamo su bili cijeli brojevi, a ovdje su realni.

Sve ostalo je vrlo slično!

Tekst programa spremljen je u datoteci `prog_3.c`.

Treći program — tekst

```
#include <stdio.h>

int main(void) {
    double x, y, z;

    printf(" Upisi x i y:\n");
    scanf("%lg %lg", &x, &y);

    z = 2 * x * x - y * y * y;

    printf(" Rezultat = %g\n", z);
```

Kad nema žute crte na kraju, nastavak je na sljedećoj stranici!

Treći program — tekst (nastavak)

```
    return 0;
}
```

Jedina stvarna razlika obzirom na prethodni program je

- u oznakama konverzije za formatirano čitanje i pisanje.

Zato obratite pažnju na ta mjesta u programu.

Treći program — opis (dodatak)

Prema savjetu iz prethodnog programa, dodali smo naredbu

```
printf(" Upisi x i y:\n");
```

za ispis poruke prije čitanja,

- tako da znamo što treba napraviti.

Treći program — opis (dodatak)

Naredba

```
scanf("%lg %lg", &x, &y);
```

učitava dvije realne vrijednosti

- po oznakama konverzije `%lg` za tip `double`,
- i učitane vrijednosti dodjeljuje, redom, varijablama `x` i `y`.

Napomena. Oznake konverzije za formatirano čitanje realnih brojeva su (zasad):

- `%g` — za `float`,
- `%lg` — za `double`.

Slovo `l` ispred znaka konverzije `g` je modifikator duljine tipa (`l` dolazi od long).

Treći program — opis (dodatak)

Sljedeća naredba

```
z = 2 * x * x - y * y * y;
```

računa vrijednost zadanog izraza i sprema tu vrijednost u varijablu `z`.

Već smo rekli da C nema poseban operator za potenciranje.

- Zato potencije računamo ponovljenim množenjem.

Tek toliko da znate: za veće potencije, počev od četvrte, ima i bolji algoritam — tzv. binarno potenciranje.

- Možemo napisati i `z = 2 * pow(x, 2) - pow(y, 3);`. Za funkciju `pow`, u program treba uključiti matematičku biblioteku `<math.h>` i linkeru narediti da ju poveže (`-lm`).

Treći program — opis (dodatak)

Naredba

```
printf(" Rezultat = %g\n", z);
```

osim navedenog teksta, ispisuje i vrijednost varijable `z`.

Napomena. Oznaka konverzija za formatirano pisanje realnih brojeva je (zasad):

- `%g` — za `double` i za `float`.

Nema posebne oznake za `float`, jer se, prilikom ispisa,

- vrijednost tipa `float` uvijek pretvara u `double`.

Treći program — izvršavanje i rezultat

Kad pokrenemo program, prvo se ispiše poruka

- `Upisi x i y:` s prijelazom u novi red.

Zatim program čeka da upišemo vrijednosti za `x` i `y`.
Ako napišemo niz znakova:

- `3.0␣2.0` i stisnemo `ENTER`,

dobivamo izlaz:

- `Rezultat = 10`

Oznaka konverzije `%g` ne piše nepotrebne nule i decimalnu točku, pa rezultat izgleda kao cijeli broj.

Probajte neke druge vrijednosti na ulazu!

Ulaz je u datoteci `prog_3.in`, a izlaz u `prog_3.out`.

Treći program — još malo (čitanje)

Oprez s oznakom konverzije za čitanje realnih brojeva:

- `%g` — služi za čitanje vrijednosti tipa float,
- `%lg` — služi za čitanje vrijednosti tipa double.

Nemojte zaboraviti slovo `l` kod čitanja za `double`!

Što se dogodi ako zaboravimo slovo `l`?

- Pristojan prevoditelj se pobuni s porukom — ako ste ga "zamolili" da javlja sve što može. Inače, može i "šutiti"!

Na pr., `gcc` u Code::Blocks, uz `-Wall -Wextra -pedantic`,

- javi: 0 errors, 2 warnings,
- s vrlo urednim opisom što ga "smeta".

Nemojte ignorirati te poruke, čak ni upozorenja!

Treći program — još malo (pisanje)

Ako ipak izvršimo takav program,

- čita se `float` i sprema u prva 4 bajta na zadanoj adresi, a ne na svih 8 bajtova.

Dobijemo "svašta" na zadnja 4 bajta!

- Rezultati su slučajni.

Pogledati: `prog_4.c`, ulaz `prog_4.in`, izlaz `prog_4.out`.

Kod pisanja nema te opasnosti.

Oznaka konverzije za formatirano pisanje realnih brojeva:

- `%g` — služi i za `double` i za `float`.

(Tip `float` se pretvara u `double`.)