Uvod u računarstvo
12. predavanje

Saša Singer

singer@math.hr

web.math.hr/~singer

PMF – Matematički odjel, Zagreb

Funkcije

Sadržaj predavanja

- Funkcije:
  - Definicija funkcije.
  - Naredba `return`.
  - Funkcija tipa `void`.
  - Funkcija bez argumenata.
  - Deklaracija funkcije.
  - Prijenos argumenata.
  - Rekurzivne funkcije.
  - Funkcije s varijabilnim brojem argumenata.

Definicija funkcije

Funkcija je programska cjelina koja

- uzima neke ulazne podatke,
- izvršava određeni niz naredbi,
- i vraća rezultat svog izvršavanja pozivnom programu.

Definicija funkcije ima oblik:

```
tip_podatka ime_funkcije(tip_1 arg_1,
                         ..., tip_n arg_n)
{
    tijelo funkcije
}
```

Definicija funkcije (nastavak)

Opis pojedinih dijelova definicije funkcije:

- `tip_podatka` je tip podatka koji će funkcija vratiti kao rezultat svog izvršavanja. Ako nije naveden, pretpostavlja se da funkcija vraća `int`.
- `ime_funkcije` je identifikator.
- Unutar zagrada iza imena funkcije nalazi se deklaracija formalnih argumenata funkcije (ako ih ima).
  - Prvi argument `arg_1` je (lokalna) varijabla tipa `tip_1`, i tako redom.
- Deklaracije pojedinih argumenata međusobno se odvajaju zarezom (to nije zarez operator).
- Unutar vitičastih zagrada nalazi se tijelo funkcije koje se sastoji od deklaracija varijabli i izvršnih naredbi.

Naredba `return`

Funkcija vraća rezultat svog izvršavanja naredbom **`return`**. Opći oblik te naredbe je:

```
return izraz;
```

Izraz se može staviti u oble zagrade, ali to nije nužno.

```
return (izraz);
```

Pravilo: funkcija može vratiti:

- aritmetički tip, strukturu, uniju ili pokazivač,

ali ne može vratiti drugu funkciju ili polje.

Ako je tip izraza u naredbi **`return`** različit od tipa podatka koji funkcija vraća, izraz će biti konvertiran u `tip_podatka`.

Primjer funkcije

Primjer. Sljedeća funkcija pretvara mala slova (engleske abecede) u velika (kao standardna funkcija `toupper`).

- Formalni argument je samo jedan (`z`) i tipa je `char`.
- Vraćena vrijednost je tipa `char`.
- Ime funkcije je `malo_u_veliko`.

```
char malo_u_veliko(char z)
{
    char c;
    c = (z >= 'a' && z <= 'z') ?
            ('A' + z - 'a') : z;
    return c;
}
```

Poziv funkcije

Funkcija se poziva navođenjem imena i liste (popisa) stvarnih argumenata u zagradama.

Primjer. Poziv funkcije iz prethodnog primjera:

```
int main(void)
{
    char malo, veliko;

    printf("Unesite malo slovo: ");
    scanf("%c", &malo);
    veliko = malo_u_veliko(malo);
    printf("\nVeliko slovo = %c\n", veliko);
    return 0;
}
```

Poziv funkcije — izraz kao stvarni argument

Stvarni argument funkcije može biti izraz. Sasvim općenito,

- stvarni argument je uvijek izraz.

Primjer: funkcija `sqrt` iz `<math.h>` s prototipom

```
double sqrt(double)
```

može biti pozvana ovako:

```
double x, y;
...
y = sqrt(2 * x - 3);
```

Varijabla `y` primit će vrijednost $\sqrt{2x-3}$.

Višestruke `return` naredbe

Ako se programski tok unutar funkcije grana, onda

- možemo imati više `return` naredbi unutar iste funkcije.

Primjer. Funkcija koja pretvara mala u velika slova, napisana `if-else` naredbom.

```
char malo_u_veliko(char z)
{
    if (z >= 'a' && z <= 'z')
        return ('A' + z - 'a');
    else
        return z;
}
```

Funkcija bez rezultata — tipa `void`

Ako funkcija ne vraća nikakvu vrijednost, onda se za tip "vraćene vrijednosti" koristi ključna riječ **`void`** ("prazan").

Primjer. Ispis maksimalnog od dva cijela broja.

```
void maximum(int x, int y)
{
    int max;
    max = (x >= y) ? x : y;
    printf("\nMaksimalna vrijednost =%d\n", max);
    return;
}
```

Naredba **`return`** nema izraz. Ako je na kraju funkcije, može biti izostavljena, no bolje ju je zadržati, radi preglednosti.

Funkcija bez argumenata

Funkcija koja nema nikakve argumente definira se ovako:

```
tip_podatka ime_funkcije(void)
{
    tijelo funkcije
}
```

Ključna riječ `void` (unutar zagrada) označava da funkcija ne uzima argumente.

Funkcija bez argumenata — poziv

Poziv takve funkcije ima praznu listu stvarnih argumenata u zagradama:

```
varijabla = ime_funkcije();
```

Zagrade su obavezne, jer informiraju prevoditelj da je identifikator `ime_funkcije` ime funkcije, a ne nešto drugo.

Tijelo funkcije

Tijelo funkcije sastoji se od deklaracija varijabli (objekata) i izvršnih naredbi.

- Po standardu C90, deklaracije varijabli moraju prethoditi prvoj izvršnoj naredbi.
- Standard C99 dozvoljava deklaracije varijabli bilo gdje u tijelu funkcije (bloku), samo da su prije korištenja objekata.

Uočite da se tijelo funkcije piše unutar vitičastih zagrada, tj.

- ima strukturu bloka, odnosno, složene naredbe.

Gornja pravila o redoslijedu deklaracija i izvršnih naredbi

- vrijede za bilo koji blok u programu.

(Detaljnije u sljedećem poglavlju.)

Deklaracija funkcije (1)

Svaka bi funkcija prije svoga poziva u programu trebala biti deklarirana — navođenjem prototipa.

- Mogućnost da se to ne napravi ostavljena je samo zbog kompatibilnosti s prastarim C programima, i ne treba ju koristiti!

Osim toga, u jeziku C++ više nije dozvoljena.

Svrha deklaracije (prototipa) je kontrola ispravnosti poziva funkcije prilikom prevođenja programa.

- Deklaracija informira prevoditelj o:
  - imenu funkcije,
  - broju i tipu argumenata,
  - te tipu vrijednosti kojeg funkcija vraća.

Deklaracija funkcije (1) (nastavak)

Ako je funkcija definirana u istoj datoteci u kojoj se poziva, i to prije svog prvog poziva, onda definicija služi kao deklaracija (pa posebna deklaracija nije potrebna).

Primjer. Funkcija definirana prije svog poziva (u `main`).

```
#include <stdio.h>

void maximum(int x, int y)
{
    int max;
    max = (x >= y) ? x : y;
    printf("\nMaksimalna vrijednost =%d\n", max);
    return;
}
```

Deklaracija funkcije (1) (nastavak)

```
int main(void)
{
    int x, y;

    printf("Unesite dva cijela broja: ");
    scanf("%d %d", &x, &y);
    maximum(x, y);
    return 0;
}
```

U trenutku prvog poziva prevoditelj zna da je `maximum` funkcija koja

- ima dva argumenta tipa `int`,
- i ne vraća ništa.

Deklaracija funkcije (2)

Ako definiciju funkcije smjestimo nakon poziva funkcije, moramo tu funkciju

- deklarirati prije prvog poziva,

(obično na početku datoteke, ili u funkciji u kojoj je poziv).

Deklaracija ili prototip funkcije ima oblik:

```
tip_podatka ime_funkcije(tip_1 arg_1,
                         ..., tip_n arg_n);
```

Deklaracija sadrži samo zaglavlje funkcije, bez bloka u kojem je tijelo funkcije.

Imena argumenata `arg_1`, . . . , `arg_n` mogu biti izostavljena (jer se tip argumenata vidi i bez toga).

Deklaracija funkcije (2) (nastavak)

Primjer. Funkcija definirana iza svog poziva (u `main`).

```
#include <stdio.h>

int main(void)
{
    int x, y;
    void maximum(int x, int y);  /* Deklaracija */

    printf("Unesite dva cijela broja: ");
    scanf("%d %d", &x, &y);
    maximum(x, y);
    return 0;
}
```

Deklaracija funkcije (2) (nastavak)

```
void maximum(int x, int y)
{
    int max;
    max = (x >= y) ? x : y;
    printf("\nMaksimalna vrijednost =%d\n", max);
    return;
}
```

U ovom primjeru je deklaracija (prototip) funkcije:

```
    void maximum(int x, int y);
```

navedena unutar funkcije `main` (gdje je i poziv).

Deklaracija funkcije (2) (nastavak)

Deklaracija funkcije može biti i izvan glavnog programa.

```
#include <stdio.h>

void maximum(int, int);     /* Deklaracija */

int main(void)
{
    int x, y;

    printf("Unesite dva cijela broja: ");
    scanf("%d %d", &x, &y);
    maximum(x, y);
    return 0;
}
```

Deklaracija funkcije (2) (nastavak)

```
void maximum(int x, int y)
{
    int max;
    max = (x >= y) ? x : y;
    printf("\nMaksimalna vrijednost =%d\n", max);
    return;
}
```

Prednost ove deklaracije: funkcija `maximum` se može pozvati u svim funkcijama iza deklaracije.

Prijenos argumenata

Formalni i stvarni argumenti:

- Argumenti deklarirani u definiciji funkcije nazivaju se formalni argumenti.
- Izrazi koji se pri pozivu funkcije nalaze na mjestima formalnih argumenata nazivaju se stvarni argumenti.

Veza formalnih i stvarnih argumenata pri pozivu funkcije:

- C ima samo tzv. prijenos argumenata po vrijednosti.
- Svaki formalni argument ujedno je i lokalna varijabla u toj funkciji (v. sljedeće poglavlje).
- Prilikom poziva funkcije, prvo se računaju vrijednosti stvarnih argumenata (izrazi) i onda kopiraju u formalne argumente.

Prijenos argumenata po vrijednosti

- Funkcija prima kopije stvarnih argumenata što znači da ne može izmijeniti stvarne argumente.

Primjer. Prijenos argumenata po vrijednosti.

```
#include <stdio.h>

void f(int x) {
    x += 1;
    printf("Unutar funkcije x=%d\n", x);
    return;
}
```

Povećanje `x` za `1` događa se u lokalnoj varijabli `x`, pa nema traga izvan funkcije `f`.

Prijenos argumenata po vrijednosti (nastavak)

```
int main(void)
{
    int x = 5;
    printf("Prije poziva funkcije x=%d\n", x);
    f(x);
    printf("Nakon poziva funkcije x=%d\n", x);
    return 0;
}
```

Rezultat izvršavanja programa bit će:

```
Prije poziva funkcije x=5
Unutar funkcije x=6
Nakon poziva funkcije x=5
```

Prijenos argumenata po adresi

- Funkcija prima adrese stvarnih argumenata, što znači da može izmijeniti stvarne argumente (sadržaj na tim adresama).

Napomena: U C-u toga nema, i svi argumenti se prenose po vrijednosti.

Ako funkcijom želimo promijeniti vrijednost nekog podatka, pripadni argument treba biti pokazivač na taj podatak (njegova adresa).

Tada se adresa prenosi po vrijednosti (tj. kopira u funkciju), ali smijemo promijeniti sadržaj na toj adresi (operatorom dereferenciranja).

Prijenos argumenata po adresi (nastavak)

Primjer. Prijenos argumenata "po adresi" preko pokazivača.

```
#include <stdio.h>

void f(int *x) {
    *x += 1;
    printf("Unutar funkcije x=%d\n", *x);
    return;
}
```

Povećavamo sadržaj na adresi `x` za `1`.

Pokazivač (adresa) je lokalna varijabla `x`. Promjena te varijable (tj. adrese) nema traga izvan funkcije `f`.

Prijenos argumenata po adresi (nastavak)

```
int main(void)
{
    int x = 5;
    printf("Prije poziva funkcije x=%d\n", x);
    f(&x);  /* Stvarni argument je pokazivac */
    printf("Nakon poziva funkcije x=%d\n", x);
    return 0;
}
```

Rezultat izvršavanja programa bit će:

```
Prije poziva funkcije x=5
Unutar funkcije x=6
Nakon poziva funkcije x=6
```

Primjer — po vrijednosti

```
#include <stdio.h>
void f(int x, int y){
    x += y;
    y += x;
    printf("Unutar funkcije x=%d, y=%d", x, y);
}
int main(){
    int x = 2, y = 3;
    printf("Prije poziva funkcije x=%d, y=%d", x, y);
    f(y, x + y);
    printf("Nakon poziva funkcije x=%d, y=%d", x, y);
    return 0; }
```

Ispis: 2 3 8 13 2 3

Primjer — "po adresi"

```
#include <stdio.h>
void f(int *x, int *y){
    *x += *y;
    *y += *x;
    printf("Unutar funkcije x=%d, y=%d", *x, *y);
}
int main(){
    int z, x = 2, y = 3;
    z = x + y;
    printf("Prije poziva funkcije x=%d, y=%d", x, y);
    f(&y, &z);
    printf("Nakon poziva funkcije x=%d, y=%d", x, y);
    return 0; }
```

Ispis: 2 3 8 13 2 8

Pravila o argumentima

Pravila pri prijenosu argumenata:

- Broj stvarnih argumenata pri svakom pozivu funkcije mora biti jednak broju formalnih argumenata.
- Ako je funkcija ispravno deklarirana (ima prototip), stvarni argumenti kojima se tip razlikuje od tipa odgovarajućih formalnih argumenta, konvertiraju se u tip formalnih argumenata (kao pri pridruživanju).
- Redosljed izračunavanja stvarnih argumenata nije definiran i može ovisiti o implementaciji (zarez u popisu argumenata nije operator).

Rekurzivne funkcije

C dozvoljava da se funkcije koriste rekurzivno, odnosno da pozivaju same sebe. Na primjer, za računanje faktorijela

$$n! = 1 \cdot 2 \cdot 3 \cdots n = n \cdot (n-1)!$$

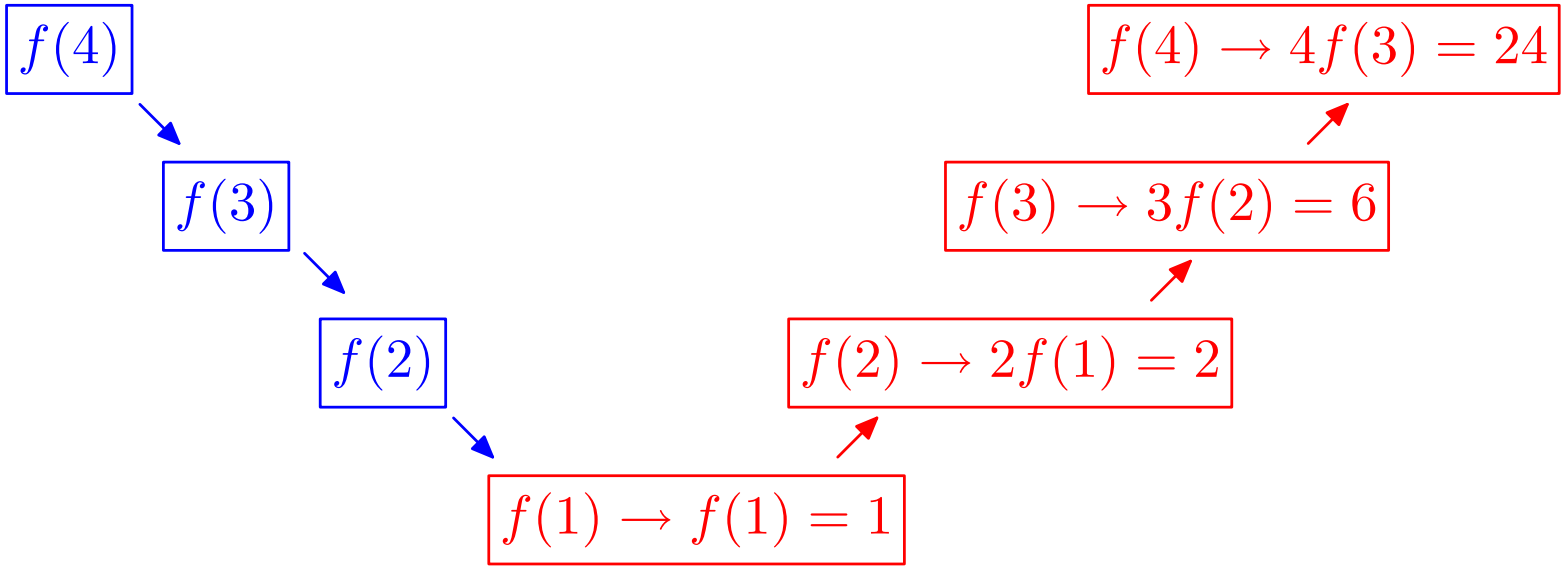
možemo napisati rekurzivnu funkciju:

```
long faktorijeli(long n) {
    if (n <= 1)
        return 1;
    else
        return n * faktorijeli(n - 1);
}
```

Ali, nemojte to raditi. Zabranjujem!

Rekurzivne funkcije (nastavak)

Funkcija (rekurzivno) poziva samu sebe `n-1` puta kako bi izračunala `n!`.



Osnovni nedostatak ovog primjera:

- pozivi su "linearni" i potrebno je n poziva funkcije (uključivo i polazni vanjski poziv), da se izračuna $n!$.

Rekurzivne funkcije (nastavak)

To ide puno brže nerekurzivno, bez svih tih silnih poziva.

Faktorijele, naravno, možemo izračunati u jednoj petlji:

```
long faktorijeli(long n) {
    long f = 1;
    for (; n > 1; n--) f *= n;
    return f;
}
```

To je bitno efikasnije jer trebamo samo jedan poziv funkcije. Sve ostalo (množenja) traje podjednako!

Rekurzivne funkcije (nastavak)

Primjer. Funkcija čita znakove sa standardnog ulaza, sve dok ne naiđe na prijelaz u novu liniju, i ispisuje učitane znakove obrnutim redosljedom.

```
#include <stdio.h>
void unos(void) {
    char znak;
    if ((znak = getchar()) != '\n') unos();
    putchar(znak);
}
```

Rekurzija služi pamćenju učitanih znakova u lokalnoj varijabli `znak`. Ispis nakon rekurzivnog poziva daje ispis unatrag (kako se vraćamo iz rekurzije).

Rekurzivne funkcije (nastavak)

Glavni program (funkcija `main`):

```
int main(void) {
   printf(" Unesite niz znakova: ");
   unos();
   return 0;
}
```

Izvršavanjem s ulazom `Zdravo`, dobit ćemo ovaj rezultat:

```
 Unesite niz znakova: Zdravo

ovardZ
```

Prvo je ispisan je zadnji učitani znak `\n`.

Rekurzivne funkcije (nastavak)

Pravi primjeri rekurzivnih funkcija su:

- `quicksort` i `mergesort` algoritmi za sortiranje,
- Hanojski tornjevi,
- obrada binarnih stabala i drugih sličnih struktura,
- sintaktička analiza programa, po gramatičkim pravilima jezika.

Funkcije s varijabilnim brojem argumenata

Funkcije `scanf` i `printf` primjeri su funkcija koje primaju varijabilan broj argumenata.

Datoteka zaglavlja `<stdarg.h>` sadrži niz definicija i makro naredbi koje omogućavaju u pisanje funkcija s varijabilnim brojem argumenata.

Opširnije u skripti i knjizi KR2.