Funkcije

Autori:
Milan Segedinac
Goran Savić

1. Funkcije

U programiranju *specifikacijom problema* definišemo kakva želimo da bude veza između ulaza i izlaza, a *algoritmom* za dati problem definišemo *kako* će zadati ulaz biti transformisan u željeni izlaz. Međutim, ukoliko bismo morali da implementiramo svaki algoritam uvek kada nam zatreba kada programiramo, bilo bi nemoguće razviti kompleksne softverske aplikacije. *Funkcije* u programiranju su mehanizam pomoću kog se implementirani algoritmi mogu ponovo koristiti u novim situacijama. Shodno tome, funkciju možemo da *definišemo* – da napišemo programski kod kojim implementiramo algoritam za tu funkciju – i da je proizvoljan broj puta *pozovemo* – izvršimo je za zadate ulazne podatke

Definiciju funkcije čine sledeći elementi:

- *Naziv funkcije* – ime dodeljeno funkciji, po kome ćemo pristupati funkciji
- *Parametri* – ulazni podaci
- *Telo funkcije* – implementacija algoritma transformacije ulaznih podataka u izlazne podatke
- *Povratna vrednost* – izlazni podaci

Prilikom *poziva* funkcije potrebno je samo da navedemo *naziv funkcije* i *konkretne vrednosti ulaznih podataka* da bismo pozvali izvršavanje tela funkcije koje će proizvesti i vratiti povratnu vrednost. Povratnu vrednost funkcije možemo dodeliti varijabli.

Kada pozivamo funkciju ne moramo da se obaziremo na njenu implementaciju, to jest poziv funkcije možemo da *apstrahujemo* od njene implementacije. Pojam *apstrakcije* – izdvajanje bitnih karakteristika pojma od konkretnih predmeta na koje se taj pojam odnosi – je centralni pojam računarstva. Do sada smo se već susreli sa apstrakcijama: kada smo vrednosti složenih izraza dodeljivali varijablama, varijabla je apstrahovala konkretan složeni izraz koji je evaluiran u vrednost koja joj je dodeljena. Apstrakcija kakvu uvode funkcije – u kojoj *šta* hoćemo da uradimo apstrahujemo od konkretnog načina (*kako*) to implementiramo – zove se *proceduralna apstrakcija*.

Programski jezik Java, na žalost, ne podržava pisanje funkcija van objekata. Proceduralna apstrakcija u tom programskom jeziku realizuje se pomoću *metoda* koje imaju sve navedene elemente funkcija, ali se, pored tog, mora i specificirati *objekat* ili *klasa* nad kojim se pozivaju. O ovim pojmovima biće više reči kada se budemo bavili objektno-orjentisanim programiranjem. Za sada ćemo kreirati i pozivati samo *statičke metode* – metode definisane na nivou klase kojima možemo da pristupimo direktno iz main metode. U ovoj lekciji (i jedino u njoj!) termini *statička metoda* i *funkcija* koristiće se kao sinonimi. Kasnije ćemo videti da između njih postoje veoma velike razlike.

Definisanje, poziv i izvršavanje funkcija

Korišćenje funkcija ilustrovaćemo na primeru ranije objašnjenog Euklidovog algoritma. Podsetimo se da je programski kod za pronalaženje NZS pomoću Euklidovog algoritma za brojeve 25 i 40 implementiran kao što je prikazano na listingu ispod.

```
public static void main(String[] args) {
	int a = 25;
	int b = 40;
	while(b!=0){
		if(b>=a){
			b -= a;
		}
		else{
			a -= b;
		}
	}
	System.out.println(a);
}
```

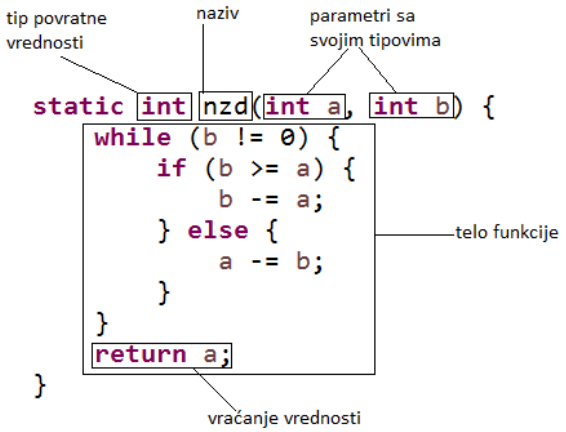
Listing – Euklidov algoritam

Ovaj programski kod je praktično neupotrebljiv jer se može koristiti samo za konkretne brojeve 25 i 40. Stoga ćemo implementaciju samog algoritma izdvojiti u zasebnu funkciju, koja je prikazana listingom ispod.

```
static int nzd(int a, int b) {
	while (b != 0) {
		if (b >= a) {
			b -= a;
		} else {
			a -= b;
		}
	}
	return a;
}
```

Listing – Funkcija kojom je implementiran Euklidov algoritam

Na ovom primeru možemo da uočimo osnovne elemente funkcije, koji su označeni na slici ispod.



Slika – Definicija funkcije

Funkcija ima *naziv* `nzd` i ima dva *parametra*, `a` i `b` koji su oba tipa `int`. U programskom jeziku Java parametri se navode u zagradi koja sledi nakon naziva. Za svaki parametar se navodi tip i naziv parametra, a, ako ih ima više, razdvajaju se zarezom. Funkcija može da ne primi ni jedan parametar –

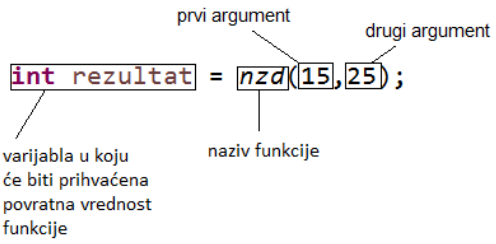
u tom slučaju imali bismo smo otvorenu i zatvorenu zagradu bez liste parametara. Ova funkcija vraća vrednost koja je tipa `int`, što je navedeno neposredno pre naziva funkcije. Telo ove funkcije implementira Euklidov algoritam. Poslednjim redom tela funkcije (`return a;`) naznačili smo da je `a` *povratna vrednost* funkcije. U programskom Jeziku Java metoda ne mora da vrati vrednost, odnosno return iskaz može da izostane – za takve metode postavlja se da je povratna vrednost tipa `void`.

Poziv funkcije dat je listingom ispod.

```
int rezultat = nzd(15,25);
```

Listing – Poziv funkcije

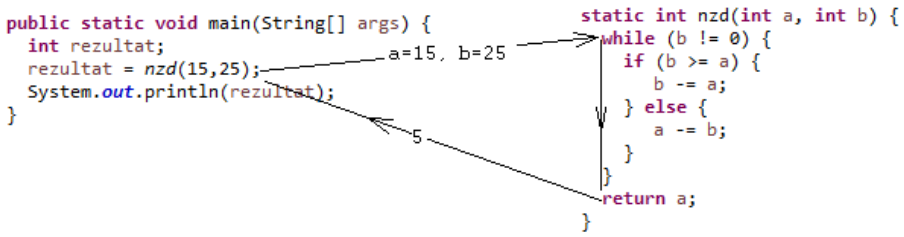
Elementi poziva funkcije izdvojeni su na slici ispod.



Slika – Poziv funkcije

Prilikom poziva funkcije navodimo njen naziv i listu *argumenata* – konkretnih vrednosti koje se dodeljuju parametrima iz definicije funkcije. Argumenti se sa parametrima uparuju po redosledu kojim su navedeni (prvi sa prvim, drugi sa drugim i tako do kraja). Shodno tome broj argumenata mora da bude identičan sa brojem parametara iz definicije funkcije (na primer, ovu funkciju nismo mogli da pozovemo pozivom `nzd(15,25,10)`) i tipovi podataka argumenata moraju da odgovaraju tipovima parametara sa kojim se uparuju. Povratnu vrednost funkcije dodelili smo varijabli `rezultat`. I one su, naravno morale da bude odgovarajućih tipova (u našem slučaju povratna vrednost funkcije `nzd` i varijabla `rezultat` su tipa `int`).

Nakon što je funkcija pozvana, privremeno se prekida izvršavanje programskog koda iz kog je funkcija pozvana i počinje izvršavanje pozvane funkcije. Pozvana funkcija dobija vrednosti parametara koje su joj prosleđene prilikom poziva i izvršava se njeno telo. Kada se završi izvršavanje tela pozvane funkcije, pozvana funkcija šalje povratnu vrednost i program nastavlja da se izvršava na mestu poziva funkcije (gde je bio privremeno prekinut) i prima se vrednost koju je pozvana funkcija vratila. Tok izvršavanja programa u kome se poziva funkcija `nzd` prikazan je na slici ispod.



Slika – Tok izvršavanja funkcije

Tok izvršavanja programa iz primera sa slike iznad je sledeći:

1. U `main` metodi se deklariše varijabla `rezultat` tipa `int`
2. Prekine se izvršavanje `main` metode i pozove se funkcija `nzd` za vrednosti `a=15` i `b=25`
3. Izvrši se telo funkcije `nzd` i izračuna `a`
4. Završi se izvršavanje `nzd` metode i vrati vrednost `5`
5. Nastavi se sa izvršavanjem `main` metode od dodele povratne vrednosti funkcije `nzd` (povratna vrednost je `5`) varijabli `rezultat`
6. Ispiše se vrednost varijable `rezultat` (i ova linija koda je poziv metode, ali o tome će biti reči kasnije)

Prilikom izvršavanja funkcija privremeno se prekida tok programa i prelazi se na telo pozvane funkcije uz prenos parametara. Nakon izvršavanja funkcije, program treba se nastavi od tačke u kojoj je bio prekinut, sa vrednostima koje su lokalne varijable imale u momentu poziva funkcije. To se ostvaruje korišćenjem memorijskog objekta koji se zove *stack poziva* (*call stack*). Pri svakom pozivu funkcije, funkciji se dodeli jedan deo te memorije koji se zove *stack frame* i on sadrži povratnu adresu i lokalne varijable i parametre sa njihovim vrednostima. Funkcija može da pristupi samo lokalnim varijablama svog stack frame-a. Shodno tome, iz funkcije `nzd` ne može se pristupiti varijabli `rezultat` iz main metode, kao što ni iz `main` metode nije moguće pristupiti varijablama `a` i `b` u `nzd`.

2. Rekurzivne funkcije

Ne postoji ni jedan razlog koji bi sprečavao da funkcija pozove samu sebe, odnosno da funkcija bude *rekurzivna*. Definisanje takvih funkcija česta je praksa u matematici. Tako faktorijel definišemo na sledeći način:

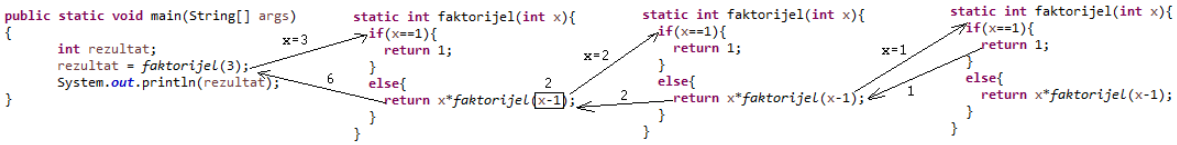
$$x! = \begin{cases} 1, x = 1 \\ x(x-1)!, x > 1 \end{cases}$$

Ovakvu definiciju mogli bismo prevesti u program prikazan listingom ispod.

```
static int faktorijel(int x){
  if(x==1){
    return 1;
  }
  else{
    return x*faktorijel(x-1);
  }
}
```

Listing – Rekurzivno računanje faktorijela

Izvršavanje funkcija za računanje faktorijela prikazano je na slici ispod.



Slika – Rekurzivno računanje faktorijela

Uočimo da činjenica da se u svakom izvršavanju funkcije radi sa parametrom koji se zove isto (`x`) nije predstavljala problem. To je zbog toga što je svakom izvršavanju dodeljen poseban stack frame, kome ne mogu da pristupe drugi pozivi funkcija. Poziv funkcije možemo stoga (za sada) posmatrati kao svet za sebe koji sa drugim programskim kodom komunicira samo preko ulaznih parametara i povratnih vrednosti.

Rekurzivne funkcije moraju da imaju makar jedan *osnovni slučaj* za koji je vrednost poznata po definiciji, a za ostale vrednosti programski kod koji će ih približavati osnovnom slučaju. Na primeru faktorijela, osnovni slučaj je bio kada je `x` jednako `1`, a činjenica da je u svakom pozivu funkcije `faktorijel` vrednost `x` umanjeno za `1` približavala je vrednosti osnovnom slučaju. Ako bismo izostavili bilo koji od ova dva uslova dobili bismo *beskonačnu rekurziju*. Obzirom da je stek poziva funkcija konačno velik (svaki stack frame je deo memorije u koju se skladište podaci, a memorija je ograničena), izvršavanje programa rezultovaće time da, ukoliko program sadrži beskonačnu (ili legitimnu ali preveliku!) rekurziju, nakon zauzeća čitavog steka poziva, program prekine izvršavanje greškom.

Faktorijel smo mogli da izračunamo i pomoću iteracije. Funkcija koja na ovaj način računa faktorijel je prikazana u listingu ispod.

```
static int faktorijelIterativni(int x){
  int retVal = 1;
    while(x>1){
      retVal *= x;
      x--;
    }
  return retVal;
}
```

Listing – iterativno računanje faktorijela