Devete vaje iz računalništva

Mojca Murovec

08.12.1998

1. Dopolnilo k zgledu `zaporedj.c`

Na prejšnjih vajah smo s pomočjo rekurzije napisali program, ki nam je računal člene in vsoto poljubnega aritmetičnega zaporedja. Pri programu je bila težava, da se členi niso nehali izpisovati, ker program ni imel ustavljalnega pogoja in se je izvajal v neskončnost. Naj ponovim, zaporedje smo seštevali takole:

1. najprej seštejemo ostale člene zaporedja z ukazom:

```
vsota = sestej_zaporedje1(zacetni_clen + korak, korak,
                          stevilo_clenov - 1);
```

2. vsoti prištejemo še začetni člen aritmetičnega zaporedja z:

```
vsota += zacetni_clen;
```

Pri tem je nastala prej omenjena težava. Program se ni znal ustaviti, saj je poskušal sešteti neskončno aritmetično zaporedje.

Pri seštevanju zaporedja smo zato omejili število seštetih členov. Koliko členov smo sešteli določa parameter `int stevilo_clenov`, ki se nahaja v glavi definicije funkcije:

```
int sestej_zaporedje1(int zacetni_clen, int korak,
                      int stevilo_clenov)}
```

Dokončno napako popravimo takole:
Dodati moramo pogoj, ki, ko je število členov aritmetičnega zaporedja 0, vrne vsoto 0 (kot nam to prikaže program). V primeru, ko je število členov zaporedja večje od 0, je vsota enaka vsoti preostalih členov aritmetičnega zaporedja, ki ji prištejemo začetni člen. To izvedemo z:

```
int sestej_zaporedje1(int zacetni_clen, int korak,
                      int stevilo_clenov)
{
  int vsota;
  if(stevilo_clenov ==0)
    vsota =0;
  else
  {
              /* sestej ostale clene zaporedja */
```

```
    vsota = sestej_zaporedje1(zacetni_clen + korak, korak,
                              stevilo_clenov - 1);
              /* pristej zacetni clen */
    vsota += zacetni_clen;
  };
              /* vrni vsoto */
  return vsota;
} /* sestej_zaporedje1 */
```

2. Geometrijsko zaporedje

Program spremenimo tako, da nam bo namesto aritmetičnega zaporedja izpisoval geometrijsko zaporedje in vsoto teh členov zaporedja.

- Izpis izvedemo s pomočjo zanke `for`. Torej:

  ```
  void izpisi_geo_zaporedje(int zacetni_clen, int kolicnik,
                            int stevilo_clenov)
  {
    int a;              /* tekoci clen zaporedja */
    int i;              /* indeks zaporedja */

    a = zacetni_clen;
                        /* za vse clene do stevilo_clenov: */
    for(i = 1; i <= stevilo_clenov; i++)
    {
                        /* izpisi clen */
      printf("%d", a);
                        /* izracunaj naslednji clen */
      a = a * kolicnik;
    }
                        /* prehod v novo vrstico ob koncu
                           zaporedja */
    printf("\n");
  } /* izpisi_geo_zaporedje */
  ```

  V glavni del programa pa dodamo:

  ```
  izpisi_geo_zaporedje(zacetni_clen, kolicnik, stevilo_clenov);
  ```

- Vsoto členov geometrijskega zaporedja seštejemo rekurzivno.
  - Ko je število členov zaporedja 0 je vsota 0.
  - V primeru, ko je število členov zaporedja večje od 0, je vsota enaka vsoti preostalih členov geometrijskega zaporedja, kateri prištejemo začetni člen. Preostalo geometrijsko zaporedje pa izračunamo tako, da poiščemo novi člen, ki je enak produktu količnika in starega začetnega člena.

  Celotni podprogram izgleda:

```
int sestej_geo_zaporedje1(int zacetni_clen, int kolicnik,
                          int stevilo_clenov)
{
  int vsota;

  if (stevilo_clenov == 0)
    vsota = 0;
  else
  {
                   /* sestej ostale clene zaporedja */
    vsota = sestej_geo_zaporedje1(zacetni_clen * kolicnik,
                                  kolicnik,
                                  stevilo_clenov - 1);
                   /* pristej zacetni clen */
    vsota += zacetni_clen;
  }
                   /* vrni vsoto */
  return vsota;
} /* sestej_geo_zaporedje1 */
```

3. Dogovor med izvajalcem in uporabnikom programa

Nek program lahko uporabljata izvajalec in uporabnik, če poznata glavo funkcije. Kot vemo, moramo vsako funkcijo najprej *definirati* in šele nato *poklicati.* Definicija funkcije se sestoji iz dveh delov:

- glava,
- telo (omejeno je z zavitimi oklepaji).

Kot primer si oglejmo definicijo za rekurzivno seštevanje funkcije (ogledali si bomo glavo funkcije, ki je sestavljena iz treh delov):

```
int sestej_zaporedje1(int zacetni_clen, int korak,
                      int stevilo_clenov)
```

V našem primeru je:

- tip funkcije (seštevanje je celoštevilsko) – `int`;
- ime funkcije – `sestej_zaporedje1`;
- parametri – `int zacetni_clen, int korak, int stevilo_clenov`. Parametri pomenijo začetni člen, korak in koliko členov zaporedja naj sešteje.

Takrat ko nam uporabnik pove, kakšno funkcijo naj izvaja program in kakšna naj bo glava funkcije, dobi izvajalec *definicijski modul* (uporabnik in izvajalec ga poznata). Definicijski modul sestavljata glava funkcije in pogodba (glej spodaj). Izvajalec sestavi program in *objektno kodo* izroči uporabniku. Objektna koda je koda, ki jo dobimo po prevajanju *izvorne kode* in jo lahko uporabnik neposredno uporabi – uporabnika ne zanima izvorna koda (npr. program v jeziku c).

3.1 Pogodba pri definicijskem modulu

Običajno pogodba ni enolična – z drugimi besedami, sestavimo lahko več različnih pogodb z istim pomenom.

Pogodba se sestoji iz petih členov:

1. `DESC` – kratek opis delovanja programa (*description*);
2. `PARMS` – pove nam parametre funkcije (*parameters*);
3. `RESULT` – rezultat programa (*result*);
4. `PRE` – predpogoj, ki pove, kako naj uporabnik dela (kaj mora zagotoviti pred klicem podprograma), da bo podprogram pravilno deloval (*precondition*);
5. `POST` – popogoj, ki pove, kakšen pogoj je izpolnjen, ko se funkcija vrne nazaj (*postcondition*).

Prva člena sta opisna in sta namenjena predvsem razumevanju funkcije. Zadnji trije členi bolj natančno določajo funkcijo. Mi jih bomo zapisali v tako imenovani *pol-formalni* obliki – zapis ni povsem formalen, ki bi sicer omogočal samodejno preverjanje pravilnosti delovanja programa.

3.2 Primer

V primeru `zaporedj.c` imamo za funkcijo `sestej_zaporedje1` naslednjo pogodbo:

```
/* Desc:    Funkcija sesteje stevilo_clenov aritmeticnega
            zaporedja, ki je definiran z zacetnim clenom
            zacetni_clen in korakom korak.
   Parms:   zacetni_clen - zacetni clen zaporedja (po vrednosti)
            korak - razlika med zaporednima clenoma zaporedja (po
            vrednosti)
            stevilo_clenov - stevilo clenov zaporedja, ki jih
            bomo sesteli (po vrednosti)
   Result:  Sum(ai: ai = zacetni_clen + (i * korak),
                0 <= i < stevilo_clenov)
   Pre:     (RESULT IN int) AND (stevilo_clenov >= 0)
   Post:    -
*/
```

Na našem področju si lahko ogledate drugo inačico pogodbe, pri kateri sta spremenjna člena „Result” in „Post“.