Uvod u programiranje

- predavanja -

listopad 2020.

Kontrola toka programa

- 2. dio -

Selekcija

skretnica

Skretnica - switch

- U slučaju kada se kaskadna selekcija temelji na relacijskim izrazima u kojima se testira jednakost cjelobrojnih vrijednosti, prikladno je koristiti naredbu `switch`
  - to znači da je naredba `switch` primjenjiva samo u jednom od sljedećih slučajeva kaskadne selekcije:

```
if (ocj == 5) {
   printf("izvrstan");
} else if (ocj == 4) {
   printf("vrlo dobar");
} else if (ocj == 3) {
   printf("dobar");
} else if (ocj == 2) {
   printf("dovoljan");
} else if (ocj == 1) {
   printf("nedovoljan");
} else {
   printf("neispravna ocjena");
}
```

```
if (temp >= 37.0f && temp < 38.0f)
   printf("Temperatura je blago povisena");
else if (temp >= 38.0f && temp < 39.0f)
   printf("Temperatura je znacajno povisena");
else if (temp >= 39.0f)
   printf("Temperatura je opasno povisena");
```

Skretnica - switch

C program - sintaksa

```
switch (cjelobrojni_izraz) {
case konstantni_cjelobrojni_izraz_1:
   naredbe_1
case konstantni_cjelobrojni_izraz_2:
   naredbe_2
case konstantni_cjelobrojni_izraz_n:
   naredbe_n
default:
   naredbe_d
}
```

- `cjelobrojni_izraz`: izraz koji rezultira cjelobrojnom vrijednošću (cjelobrojne konstante, varijable, operatori)
- `konstantni_cjelobrojni_izraz`: cjelobrojni izraz koji ne sadrži varijable

Labela i označena naredba (*labeled statement*)

- Labela (oznaka naredbe, *label*) koristi se za označavanje naredbi. Time neke od naredbi za kontrolu toka programa dobivaju mogućnost izravno usmjeriti daljnje izvršavanje programa na tako označene naredbe (označena naredba, *labeled statement*). Npr. naredba `switch` može daljnje izvršavanje programa usmjeriti na naredbe označene jednim od dvaju* oblika labela:
  - `case konstantni_cjelobrojni_izraz:`
  - `default:`
- redoslijed labela u naredbi switch nije propisan (npr. labela `default:` se može navesti prva), ali poredak labela može utjecati na ukupno ponašanje naredbe
- navođenje labele `default:` je opcionalno

* Kasnije će biti opisan još jedan oblik labele koji se koristi u kombinaciji s naredbom goto

Skretnica - switch - princip djelovanja

- izračuna se vrijednost cjelobrojnog izraza S (izraz u zagradama iza ključne riječi `switch`)
  - **ako postoji** labela L s vrijednošću jednakoj S, izvršavanje programa se nastavlja naredbom označenom labelom L (i ne prekida se nailaskom na sljedeću labelu!).
    - kolokvijalno: nastavlja se "propadanje niz labele"
  - **inače** (ako ne postoji takva labela L) izvršavanje programa se nastavlja naredbom označenom labelom `default:` (ako takva labela u naredbi postoji, inače naredba `switch` odmah završava)

- pokušajmo ispis naziva ocjena umjesto kaskadnom selekcijom riješiti pomoću naredbe `switch`

Primjer: neispravno rješenje

```
switch (ocj) {
case 5:
   printf("izvrstan");
case 4:
   printf("vrlo dobar");
case 3:
   printf("dobar");
case 2:
   printf("dovoljan");
case 1:
   printf("nedovoljan");
default:
   printf("neispravna ocjena");
}
```

- zašto je rješenje neispravno?
  - ako se u varijabli `ocj` nalazi vrijednost `3`, ispisat će se `dobardovoljannedovoljanneispravna ocjena`
- Naredbu `switch` treba dopuniti tako da se njeno izvršavanje prekine nakon što se obave naredbe iza odgovarajuće labele
  - koristiti naredbu `break`

Primjer: ispravno rješenje

```
switch (ocj) {
case 5:
   printf("izvrstan");
   break;
case 4:
   printf("vrlo dobar");
   break;
case 3:
   printf("dobar");
   break;
case 2:
   printf("dovoljan");
   break;
case 1:
   printf("nedovoljan");
   break;
default:
   printf("neispravna ocjena");
   break;
}
```

- naredba `break` prekida daljnje izvršavanje naredbe u kojoj je navedena
  - (kasnije će se ta ista naredba koristiti i za prekidanje daljnjeg obavljanja petlje)
- posljednju naredbu `break` nije nužno napisati, ali se preporuča radi izbjegavanja kasnijih logičkih pogrešaka
- u ovom primjeru redoslijed navođenja labela nije važan

Namjerno izostavljanje naredbe break

- izostanak naredbe `break` je česta logička pogreška. Ipak, postoje slučajevi u kojima se željena funkcionalnost postiže upravo izostavljanjem naredbe `break`
  - primjer: ako `ocj` pripada skupu { 2, 3, 4, 5 } ispisati `ispit polozen`, ako je `ocj` jednaka 1 ispisati `ispit nije polozen`, inače ispisati `neispravna ocjena`
  - u ovom primjeru redoslijed labela jest važan. Što bi se dogodilo kad bi se labela `case 4:` preselila na posljednje mjesto?

```
switch (ocj) {
case 2: /*FALLTHROUGH*/
case 3: /*FALLTHROUGH*/
case 4: /*FALLTHROUGH*/
case 5:
   printf("ispit polozen");
   break;
case 1:
   printf("ispit nije polozen");
   break;
default:
   printf("neispravna ocjena");
   break;
}
```

Kaskadna selekcija ili skretnica?

- svaka naredba `switch` može se napisati u obliku kaskadne selekcije (obrnuto ne vrijedi)
- prethodni primjer može se riješiti i pomoću kaskadne selekcije:

```
if (ocj == 5 || ocj == 4 ||
    ocj == 3 || ocj == 2) {
   printf("ispit polozen");
} else if (ocj == 1) {
   printf("ispit nije polozen");
} else {
   printf("neispravna ocjena");
}
```

Kaskadna selekcija ili skretnica?

- zašto onda uopće postoji naredba switch?
  - program je razumljiviji jer je odmah vidljivo da se radi o selekciji koja se temelju na usporedbi pojedinačnih cjelobrojnih vrijednosti
  - izvršavanje naredbe `switch` je (najčešće) brže od izvršavanja ekvivalentne naredbe napisane u obliku kaskadne selekcije
    - u kaskadnoj selekciji logički izrazi se "kaskadno" evaluiraju, sve dok se ne dospije do logičkog izraza koji se evaluira kao istina
    - u naredbi `switch` obavlja se izravni skok (strojna instrukcija `JUMP`) na prvu naredbu iza odgovarajuće labele. Upravo je to razlog zašto izraz u labeli mora biti konstantni cjelobrojni izraz: na taj način prevodilac može unaprijed odrediti adresu strojne naredbe na koju treba izravno "skočiti" zavisno od vrijednosti izraza navedenog iza ključne riječi `switch`

Programske petlje

Programske petlje

- namijenjene su obavljanju određenog programskog odsječka (jedne ili više naredbi koje čine tijelo petlje) više puta
- U programskom jeziku C koriste se tri vrste petlji
  - programske petlje s ispitivanjem uvjeta na početku
    - ovisno o početnim uvjetima može se dogoditi da se tijelo petlje uopće neće izvršiti
  - programske petlje s ispitivanjem uvjeta na kraju
    - tijelo petlje će se izvršiti barem jednom
  - programske petlje s poznatim brojem ponavljanja
    - broj ponavljanja može se unaprijed izračunati i (u principu) ne ovisi o izvršavanju tijela petlje

Programska petlja

S ispitivanjem uvjeta na početku

Programske petlje - motivacija

- Programski zadatak
  - na zaslon ispisati prvih 20 prirodnih brojeva

```
1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20
```

```
#include <stdio.h>

int main(void) {
   printf("%d ",  1);
   printf("%d ",  2);
   printf("%d ",  3);
   printf("%d ",  4);
   ...
   printf("%d ", 19);
   printf("%d ", 20);
   return 0;
}
```

- mnogo puta se ponavlja (gotovo) isti posao, s time da je sav teret ponavljanja istog posla pao na programera
- s ovakvim rješenjem ne možemo biti zadovoljni - želimo da računalo veliki broj puta ponovi naredbe koje programer napiše samo jednom
- problem je što se u ovom rješenju ne ponavlja potpuno isti posao jer svaki put se ispisuje nova **konstanta**

Programske petlje - motivacija

- modificirati prethodni program. Umjesto konstante koristiti varijablu. Sada će se isti niz naredbi obavljati mnogo puta

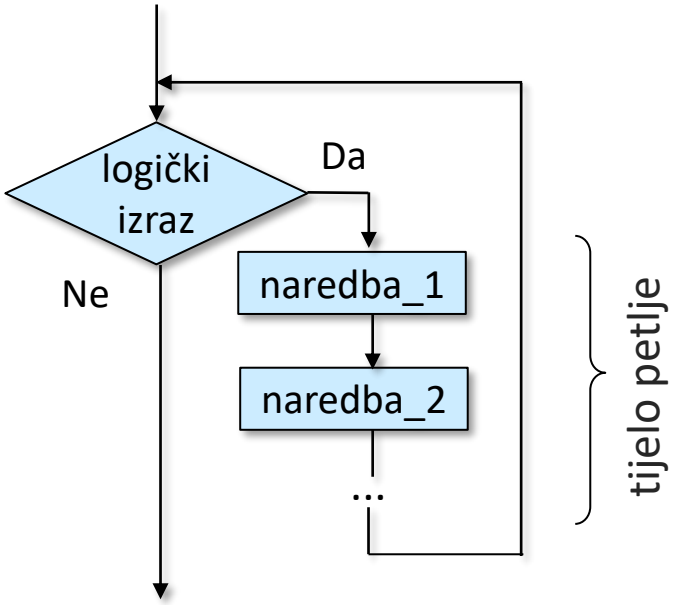
```
#include <stdio.h>

int main(void) {
   int broj = 1;
   printf("%d ", broj);
   broj = broj + 1;
   printf("%d ", broj);
   broj = broj + 1;
   ...
   printf("%d ", broj);
   broj = broj + 1;
   return 0;
}
```

- još nismo zadovoljni. I ovdje je programer ponavljao potpuno iste naredbe (dok god je vrijednost varijable broj bila manja ili jednaka 20)
- potrebna je kontrolna programska struktura pomoću koje će se računalu moći zadati ponavljanje istih naredbi. Nešto poput:

```
int broj = 1;
ponavljaj dok je broj ≤ 20
   printf("%d ", broj);
   broj = broj + 1;
```

Programska petlja s ispitivanjem uvjeta na početku

Dijagram toka



Pseudo-kod

tijelo petlje

```
...
dok je logički_izraz
    naredba_1
    naredba_2
    ...
...
```

Programska petlja s ispitivanjem uvjeta na početku

C program - sintaksa

```
while (logički_izraz)
   naredba;
```

jedna naredba!

Što ako tijelo petlje sadrži više od jedne naredbe?

Rješenje: koristiti složenu naredbu.

C program - primjer

```
...
broj = 1;
while (broj <= 20) {
   printf("%d ", broj);
   broj = broj + 1;
}
...
```

Primjer

- Programski zadatak
  - Učitati nenegativni cijeli broj. Nije potrebno provjeravati ispravnost unesenog broja. Ispisivati ostatke uzastopnog dijeljenja učitanog broja s 2, a postupak prekinuti kad se dijeljenjem dođe do 0
    - učitani broj može biti 0. Može se dogoditi da se neće ispisati niti jedan ostatak dijeljenja, odnosno da se tijelo petlje neće izvršiti niti jednom
  - Primjeri izvršavanja programa

```
Upisite nenegativan cijeli broj > 11↵
Upisali ste 11↵
Ostatak = 1↵
Ostatak = 1↵
Ostatak = 0↵
Ostatak = 1↵
```

```
Upisite nenegativan cijeli broj > 0↵
Upisali ste 0↵
```

Rješenje

```
#include <stdio.h>

int main(void) {
   int broj, ostatak;

   printf("Upisite nenegativan cijeli broj > ");
   scanf("%d", &broj);
   printf("Upisali ste %d\n", broj);

   while (broj != 0) {
      ostatak = broj % 2;
      printf("Ostatak = %d\n", ostatak);
      broj = broj / 2;
   }
   return 0;
}
```

Primjer

- Programski zadatak
  - Učitavati i sumirati cijele brojeve dok se ne upiše cijeli broj 0. Ispisati sumu učitanih brojeva.

```
#include <stdio.h>

int main(void) {
   int broj, suma = 0;
   printf("Upisite broj > ");
   scanf("%d", &broj);
   while (broj != 0) {
      suma = suma + broj;
      printf("Upisite broj > ");
      scanf("%d", &broj);
   }
   printf("Suma = %d\n", suma);
   return 0;
}
```

- broj treba učitati barem jednom, a zatim ponovo u svakom koraku petlje
- petlja s ispitivanjem uvjeta na početku nije naročito prikladna za rješavanje ovog zadatka.

Primjer

- Ponavljanje dijela programskog koda može se izbjeći *trikom* kojim će se osigurati barem jedan ulazak u tijelo petlje

```
#include <stdio.h>

int main(void) {
   int broj = 1, suma = 0;

   while (broj != 0) {
      printf("Upisite broj > ");
      scanf("%d", &broj);
      suma = suma + broj;
   }
   printf("Suma = %d\n", suma);
   return 0;
}
```

Primjer

- ... ili korištenjem pomoćne varijable

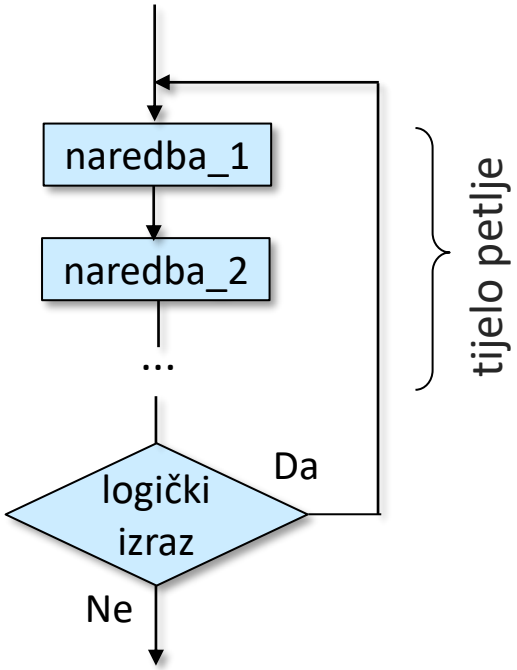
```
#include <stdio.h>

int main(void) {
   int broj, suma = 0, prviProlaz = 1;

   while (broj != 0 || prviProlaz == 1) {
      if (prviProlaz == 1) prviProlaz = 0;
      printf("Upisite broj > ");
      scanf("%d", &broj);
      suma = suma + broj;
   }
   printf("Suma = %d\n", suma);
   return 0;
}
```

Programska petlja

S ispitivanjem uvjeta na kraju

Programska petlja s ispitivanjem uvjeta na kraju

Dijagram toka



Pseudo-kod

tijelo petlje

```
...
ponavljaj
   | naredba_1
   | naredba_2
   | ...
dok je logički_izraz
...
```

Programska petlja s ispitivanjem uvjeta na kraju

C program - sintaksa

```
do
   naredba;
while (logički_izraz);
```

jedna naredba!

Što ako tijelo petlje sadrži više od jedne naredbe?

Rješenje: koristiti složenu naredbu.

C program - primjer

```
...
broj = 1;
do  {
   printf("%d ", broj);
   broj = broj + 1;
} while (broj <= 20);
...
```

Primjer

- Programski zadatak
  - Učitavati i sumirati cijele brojeve dok se ne upiše cijeli broj 0. Ispisati sumu učitanih brojeva.

```
#include <stdio.h>

int main(void) {
   int broj, suma = 0;

   do {
      printf("Upisite broj > ");
      scanf("%d", &broj);
      suma = suma + broj;
   } while (broj != 0);
   printf("Suma = %d\n", suma);
   return 0;
}
```

Primjer

- Programski zadatak
  - Učitati pozitivni cijeli broj koji određuje gornju granicu sume brojeva (ne treba provjeravati ispravnost učitanog broja). Zatim učitavati i sumirati cijele brojeve sve dok njihova suma ne prekorači zadanu gornju granicu sume. Nakon toga ispisati dosegnutu sumu, broj učitanih brojeva i aritmetičku sredinu učitanih brojeva.
  - Primjer izvršavanja programa

```
Upisite gornju granicu > 11↵
2↵
4↵
1↵
8↵
15 4 3.750000↵
```

Rješenje

```
#include <stdio.h>

int main(void) {
   int brojac = 0, suma = 0, broj, gg;
   float as;

   printf("Upisite gornju granicu > ");
   scanf("%d", &gg);

   do {
      scanf("%d", &broj);
      suma = suma + broj;
      brojac = brojac + 1;
   } while (suma <= gg);

   as = 1.f * suma / brojac;
   printf("%d %d %f\n", suma, brojac, as);
   return 0;
}
```

tijelo petlje obavit će se barem jednom jer se koristi petlja s ispitivanjem uvjeta ponavljanja na kraju

Množenje s 1.f potrebno radi realnog dijeljenja

Rješenje

```
#include <stdio.h>

int main(void) {
   int brojac = 0, suma = 0, broj, gg;
   float as;

   printf("Upisite gornju granicu > ");
   scanf("%d", &gg);

   while (suma <= gg) {
      scanf("%d", &broj);
      suma = suma + broj;
      brojac = brojac + 1;
   }

   as = 1.f * suma / brojac;
   printf("%d %d %f\n", suma, brojac, as);
   return 0;
}
```

tijelo petlje obavit će se barem jednom jer je u ovom slučaju uvjet za obavljanje tijela petlje na početku sigurno zadovoljen

Množenje s 1.f potrebno radi realnog dijeljenja

Primjer

- Programski zadatak
  - Učitavati cijele brojeve iz intervala [-100, 100]. Učitavanje brojeva prekinuti kada se učita broj izvan intervala [-100, 100]. Ispisati broj učitanih pozitivnih brojeva, broj učitanih negativnih brojeva i broj učitanih nula. U obzir uzeti samo brojeve iz intervala [-50, 50].

Rješenje

```
#include <stdio.h>
int main(void) {
   int broj;
   int brojPozitivnih = 0, brojNegativnih = 0, brojNula = 0;
   printf("Upisite brojeve > ");
   do {
      scanf("%d", &broj);
      if (broj >= -50 && broj <= 50) {
         if (broj == 0) brojNula = brojNula + 1;
         else if (broj > 0) brojPozitivnih = brojPozitivnih + 1;
         else brojNegativnih = brojNegativnih + 1;
      }
   } while (broj >= -100 && broj <= 100);

   printf("Pozitivnih je %d, negativnih je %d, nula je %d\n"
         , brojPozitivnih, brojNegativnih, brojNula);
   return 0;
}
```

Primjer

- Programski zadatak
  - S tipkovnice učitati 10 cijelih brojeva, odrediti i ispisati najveći broj.
  - Primjer izvršavanja programa

```
Upisite 10 cijelih brojeva >↵
3 -15 8 45 0 72 -99 72 0 11↵
Najveci broj je 72↵
```

Oblikovanje algoritma

- Kod traženja najveće (slično i kod traženja najmanje) vrijednosti u nekom nizu vrijednosti koristi se sljedeći algoritam:
  - prvu vrijednost proglasiti najvećom i pohraniti u pomoćnu varijablu koja predstavlja trenutačni maksimum
  - redom ispitivati jednu po jednu preostalu vrijednost i svaki puta kada se nađe vrijednost koja je veća od trenutačnog maksimuma, trenutačni maksimum ažurirati na tu vrijednost
  - nakon što se ispitaju sve vrijednosti, u pomoćnoj varijabli će se nalaziti najveća vrijednost
- Primjer: `3 -15 8 45 0 72 -99 72 0 11`

broj	3	-15	8	45	0	72	-99	72	0	11
		-15>3?	**8>3?**	**45>8?**	**0>45?**	**72>45?**	**-99>72?**	**72>72?**	**0>72?**	**11>72?**
maks	3	3	8	45	45	72	72	72	72	72

Rješenje (1. varijanta)

```
#include <stdio.h>
#define UKUP_BROJEVA 10                                   mora biti najmanje 1

int main(void) {
   int korak = 1, broj, maks;

   printf("Upisite %d cijelih brojeva >\n", UKUP_BROJEVA);
   scanf("%d", &broj);                                    prvi broj
   maks = broj;

   while (korak < UKUP_BROJEVA) {                         preostali brojevi
      korak = korak + 1;
      scanf("%d", &broj);
      if (broj > maks)
         maks = broj;
   }

   printf("Najveci broj je %d", maks);
   return 0;
}
```

Rješenje (2. varijanta)

```
#include <stdio.h>
#define UKUP_BROJEVA 10                                   mora biti najmanje 1

int main(void) {
   int korak = 0, broj, maks;

   printf("Upisite %d cijelih brojeva >\n", UKUP_BROJEVA);

   do {
      korak = korak + 1;
      scanf("%d", &broj);
      if (korak == 1)                                     prvi broj
         maks = broj;
      else                                                preostali brojevi
         if (broj > maks)
            maks = broj;
   } while (korak < UKUP_BROJEVA);

   printf("Najveci broj je %d", maks);
   return 0;
}
```

Programska petlja

S poznatim brojem ponavljanja

Programska petlja s poznatim brojem ponavljanja

C program - sintaksa

```
for (izraz_1; izraz_2; izraz_3)
   naredba;
```

jedna naredba!

Što ako tijelo petlje sadrži više od jedne naredbe?

Rješenje: koristiti složenu naredbu.

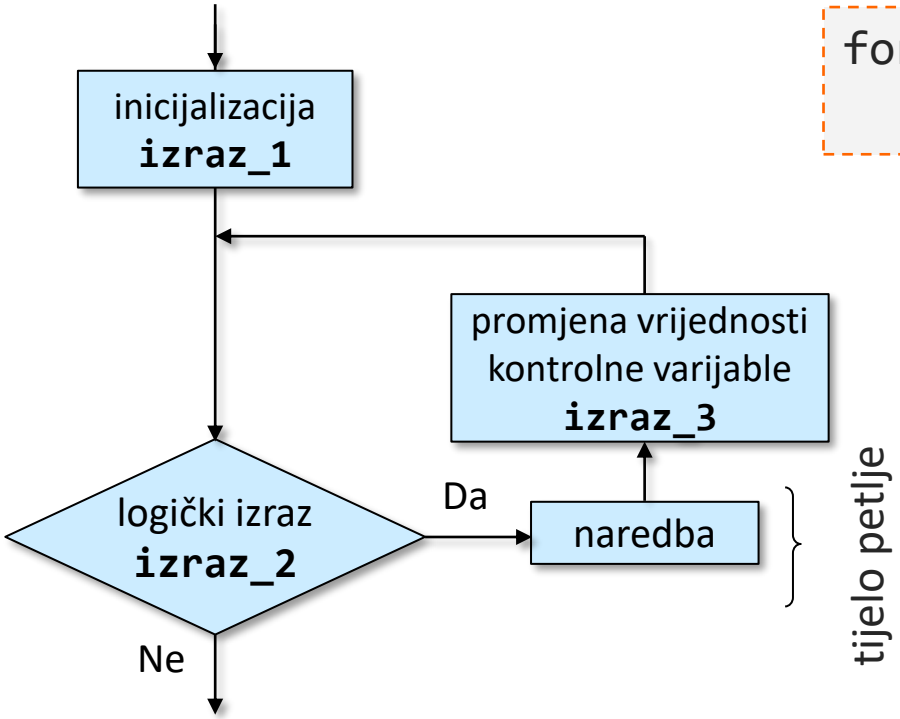
C program - primjer Ispis neparnih brojeva iz intervala [1, 10]

```
...
  int brojac;
  for (brojac = 1; brojac <= 10; brojac = brojac + 2)
      printf("%d\n", brojac);
...
```

može se staviti 9

Programska petlja s poznatim brojem ponavljanja

Dijagram toka



C program - sintaksa

```
for (izraz_1; izraz_2; izraz_3)
   naredba;
```

Značenje izraza u zagradama

```
for (izraz_1; izraz_2; izraz_3)
   naredba;
```

- izraz_1 je izraz koji će se izvršiti samo jednom, prije ulaska u prvu iteraciju petlje. Najčešće se koristi za inicijalizaciju brojača.
- izraz_2 je logički izraz. Tijelo petlje se izvršava ako je izraz_2 zadovoljen (istinit). Ako nije, petlja se prekida (nastavlja se s prvom sljedećom naredbom iza petlje).
- izraz_3 se obavlja nakon svakog prolaska kroz tijelo petlje. Najčešće se koristi za povećavanje/smanjenje vrijednosti kontrolne varijable/brojača. Nakon obavljanja izraza izraz_3, ponovo se testira uvjet u izraz_2
- Svaki od izraza (izraz_1, izraz_2, izraz_3) se može izostaviti. Ako se izostavi izraz_2, smatra se da je rezultat izraza izraz_2 uvijek istina.

Programska petlja s poznatim brojem ponavljanja

- svaka petlja s poznatim brojem ponavljanja (for-petlja) može se realizirati petljom s ispitivanjem uvjeta na početku. Glavni razlozi za korištenje for-petlje su:
  - uputa ostalim programerima da se radi o petlji za koju se odmah na početku može izračunati koliko puta će se tijelo petlje izvršiti
  - mogućnost pisanja kompaktnijeg koda

```
for (izraz_1; izraz_2; izraz_3)
   naredba;
```

```
izraz_1;
while (izraz_2) {
   naredba;
   izraz_3;
}
```

Primjer

- Programski zadatak
  - Učitati pozitivan cijeli broj *n* (nije potrebno provjeravati je li učitan ispravan broj). Zatim učitati *n* cijelih brojeva, izračunati i na zaslon ispisati njihovu aritmetičku sredinu
  - Koja vrsta petlje je najprikladnija za rješavanje ovog zadatka?
  - Primjer izvršavanja programa:

```
Koliko brojeva zelite ucitati? > 5↵
Upisite 1. broj > 3↵
Upisite 2. broj > -15↵
Upisite 3. broj > 8↵
Upisite 4. broj > 45↵
Upisite 5. broj > 7↵
Aritmeticka sredina je 9.600↵
```

Rješenje

```
#include <stdio.h>

int main(void) {
   int n, brojac, ucitani_broj, suma = 0;
   float arit_sred;

   printf("Koliko brojeva zelite ucitati? > ");
   scanf("%d", &n);

   for (brojac = 1; brojac <= n; brojac = brojac + 1) {
      printf("Upisite %d. broj > ", brojac);
      scanf("%d", &ucitani_broj);
      suma = suma + ucitani_broj;
   }

   arit_sred = 1.f * suma / n;
   printf("Aritmeticka sredina je %.3f\n", arit_sred);
   return 0;
}
```

radi realnog dijeljenja

Primjer

- Programski zadatak
  - Učitati pozitivan cijeli broj *n* (nije potrebno provjeravati je li učitan ispravan broj). Zatim od većih prema manjim, ispisati sve prirodne brojeve djeljive sa 7, 13 ili 19, koji su manji od broja *n*.
  - Koja je vrsta petlje najprikladnija za rješavanje ovog zadatka?
  - Primjer izvršavanja programa:

```
Upisite broj n > 30↵
28↵
26↵
21↵
19↵
14↵
13↵
7↵
```

Rješenje

```
#include <stdio.h>

int main(void) {
   int i, n;

   printf("Upisite broj n > ");
   scanf("%d", &n);

   for (i = n - 1; i > 0; i = i - 1) {
      if ((i % 7 == 0) ||
          (i % 13 == 0) ||
          (i % 19 == 0)) {
         printf("%d\n", i);
      }
   }
   return 0;
}
```

u konkretnom slučaju oba para vitičastih zagrada mogu se ispustiti