Uvod u programiranje

- predavanja -

listopad 2025.

6. Kontrola toka programa

Vodič „05 Prije šestog predavanja”

- Očekuje se da ste savladali (na razini na kojoj je obrađeno u vodiču):
  - break
  - continue
  - goto

- Pitanja?

Strukturirano programiranje

- programska paradigma u kojoj je dopušteno korištenje sljedećih elemenata za kontrolu toka:
  - selekcija
  - iteracija (petlje)
  - poziv potprograma (funkcije)
- izbjegavaju se:
  - naredbe za prijevremeni prekid ili nastavak petlji (break, continue)
  - korištenje više od jedne naredbe return u jednoj funkciji
- i nije dopušteno:
  - korištenje naredbe goto

Strukturirano programiranje

- disciplinirano programiranje
- programi koji nisu napisani u skladu s pravilima strukturiranog programiranja rezultat su neznanja ili lijenosti programera (osim u prije spomenutim iznimnim slučajevima)

> *„Any fool can write code that a computer can understand. Good programmers write code that humans can understand."*
>
> M. Fowler, K. Beck, J. Brant, W. Opdyke, D. Roberts: Refactoring: Improving the Design of Existing Code. Addison-Wesley, 1999.

> *"Indeed, the ratio of time spent reading versus writing is well over 10 to 1. We are constantly reading old code as part of the effort to write new code. ...[Therefore,] making it easy to read makes it easier to write."*
>
> Robert C. Martin, Clean Code: A Handbook of Agile Software Craftsmanship

Primjer

strukturirano

```
...
for (i = 1; i <= 10; i = i + 1)
   if (i % 2 == 0) {
      printf("%d je paran\n", i);
   else
      printf("%d je neparan\n", i);
...
```

nestrukturirano

```
...
i = 1;
ponovi:
   if (i % 2 == 0)
      goto ispisParan;
   printf("%d je neparan\n", i);
   i = i + 1;
   goto ponovi;
   ispisParan:
      printf("%d je paran\n", i);
      i = i + 1;
      goto ponovi;
...
```

Primjer

- Programski zadatak
  - Učitati prirodni broj (nije potrebno provjeravati je li učitan ispravan broj). Na zaslon ispisati je li učitani broj prim broj.
    - Primjeri izvršavanja programa

```
Upisite prirodni broj > 37↵
37 jest prim broj↵
```

```
Upisite prirodni broj > 35↵
35 nije prim broj↵
```

```
Upisite prirodni broj > 1↵
1 nije prim broj↵
```

Rješenje s korištenjem break

```
#include <stdio.h>
int main(void) {
   int i, n, djeljiv = 0;          // hipoteza: nije djeljiv
   printf("Upisite prirodni broj > ");
   scanf("%d", &n);
   for (i = 2; i <= n - 1; i = i + 1) {
      if (n % i == 0) {
         djeljiv = 1;             // hipoteza je bila pogresna
         break;                   // daljnja ispitivanja nisu potrebna
      }
   }
   if (djeljiv == 1 || n == 1)     // jer broj 1 je specijalan slucaj
      printf("%d nije prim broj\n", n);
   else
      printf("%d jest prim broj\n", n);
   return 0;
}
```

Rješenje bez korištenja naredbe break

```
#include <stdio.h>
int main(void) {
   int i, n, djeljiv = 0;          // hipoteza: nije djeljiv
   printf("Upisite prirodni broj > ");
   scanf("%d", &n);
   i = 2;
   while (i <= n - 1 && djeljiv == 0) {
      if (n % i == 0) {
         djeljiv = 1;              // hipoteza je bila pogresna
      }
      i = i + 1;
   }
   if (djeljiv == 1 || n == 1)     // jer broj 1 je specijalan slucaj
      printf("%d nije prim broj\n", n);
   else
      printf("%d jest prim broj\n", n);
   return 0;
}
```

Beskonačna petlja

- niz naredbi koje će se ponavljati beskonačno mnogo puta, sve do izvana nametnutog prekida programa (npr. signal iz operacijskog sustava ili gašenje računala)
  - najčešće je rezultat logičke pogreške
  - osim u posebnim slučajevima, u kojima se beskonačna petlja namjerno koristi zbog prirode problema kojeg treba riješiti
    - Primjer: vrlo pojednostavljeni pseudo-kod programa koji upravlja brzinom broda

```
ponavljaj zauvijek
  | pročitaj položaj komande za brzinu
  | pročitaj trenutačnu brzinu broda
  | izračunaj optimalni potreban broj okretaja motora
  | u odgovarajući registar kontrolera motora upiši rezultat
```

Beskonačna petlja

- primjeri beskonačnih petlji koje su nastale zbog logičke pogreške

```
for (i = 1; i <= 10; i == i + 1) {
   printf("%d\n", i);
}
```

```
i = 1;
while (i <= 10) {
   printf("%d\n", i);
}
```

```
i = 1;
while (i <= 10); {
   printf("%d\n", i);
   i = i + 1;
}
```

Pseudo-beskonačna petlja i naredba break

- da bi se petlja ipak nekako prekinula, koristi se naredba break

```
do {
   scanf("%d", &broj);
   if (broj > 0)
      brojac = brojac + 1;
} while (broj > 0);
```

```
while (1 == 1) {   // while (1)
   scanf("%d", &broj);
   if (broj > 0)
      brojac = brojac + 1;
   else
      break;
}
```

```
for (;;) {
   scanf("%d", &broj);
   if (broj > 0)
      brojac = brojac + 1;
   else
      break;
}
```

Agregatni tipovi podataka

Polja - motivacija

- Programski zadatak
  - s tipkovnice učitati 10 cijelih brojeva, ispisati njihovu aritmetičku sredinu i brojeve koji su veći od aritmetičke sredine
  - primjer izvršavanja programa

```
5 15 1 2 3 -4 25 6 8 7↵
sredina = 6.800000↵
15↵
25↵
8↵
7↵
```

  - očito, svih 10 vrijednosti će trebati pohraniti u varijable, izračunati prosjek, a zatim ispisati vrijednosti varijabli koje zadovoljavaju uvjet

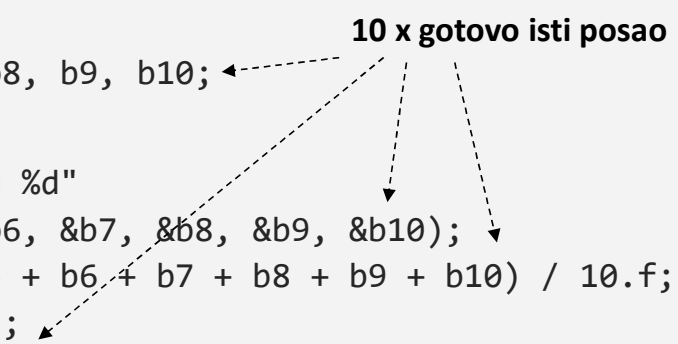
Rješenje (loše)

```
#include <stdio.h>

int main(void) {
   int b1, b2, b3, b4, b5, b6, b7, b8, b9, b10;
   float sredina;

   scanf("%d %d %d %d %d %d %d %d %d %d"
       , &b1, &b2, &b3, &b4, &b5, &b6, &b7, &b8, &b9, &b10);
   sredina = (b1 + b2 + b3 + b4 + b5 + b6 + b7 + b8 + b9 + b10) / 10.f;
   printf("sredina = %f\n", sredina);
   if (b1 > sredina) printf("%d\n", b1);
   if (b2 > sredina) printf("%d\n", b2);
   ...
   if (b9 > sredina) printf("%d\n", b9);
   if (b10 > sredina) printf("%d\n", b10);

   return 0;
}
```



Matematički niz

- Niz brojeva koji imaju zajedničko ime i čiji se članovi identificiraju indeksom: $broj_0$, $broj_1$, $broj_2$, ...
- Kada bi se niz mogao koristiti u programu, prethodni zadatak bi se mogao riješiti na sljedeći način:

```
suma := 0
za i = 1 do 10
  |učitaj(b_i)
  |suma := suma + b_i
sredina := suma / 10
za i = 1 do 10
  |ako je b_i > sredina
  |  | ispiši(b_i)
```

Matematički niz

- Bez teškoća bi se mogli riješiti i mnogo veći problemi, koje bi bilo iznimno teško riješiti bez korištenja niza: npr. učitavanje *n* brojeva i ispis onih brojeva koji su veći od njihove aritmetičke sredine

```
učitaj(n)
suma := 0
za i = 1 do n
  | učitaj(b_i)
  | suma := suma + b_i
sredina := suma / n
za i = 1 do n
  | ako je b_i > sredina
  |   | ispiši(b_i)
```

Agregatni tipovi podataka

Polja

Strukture

Agregatni tipovi podataka

- do sada su korišteni isključivo jednostavni tipovi podataka: int i float
  - u varijablu jednostavnog tipa moguće je pohraniti samo jedan elementarni podatak. Jedna varijabla ↔ jedna vrijednost
    - **skalarni** tip podatka, **skalarna** varijabla, **skalarni** podatak
- agregatni podatak (*data aggregate*) ili složeni podatak obuhvaća više skalarnih podataka i/ili više agregatnih podataka, objedinjenih pod istim imenom
  - polje (*array*)
  - struktura ili zapis (*structure, record*)
- prednosti korištenja agregatnih podataka
  - jednostavniji postupci nad skupinama podataka
  - naglašava se logička povezanost podataka

Polja

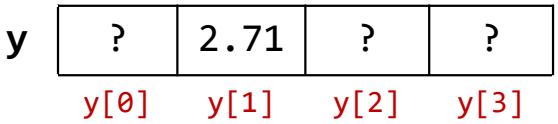
Jednodimenzijska polja

Polje

- **Jednostavni tipovi podataka**
  - jedna varijabla ↔ jedna vrijednost
  - do vrijednosti varijable pristupa se navođenjem imena varijable
  - varijabla jednostavnog tipa jest *modifiable lvalue*
- **Polje** je složeni tip podatka koji obuhvaća više članova istog tipa
  - jedna varijabla ↔ više vrijednosti
  - pojedinim vrijednostima (članovima) pristupa se pomoću indeksa
  - ime varijable je *non-modifiable lvalue*; ime varijable uz navedeni indeks elementa jest *modifiable lvalue*

```
float x;
x = 3.14f;
printf("%f", x);
```


```
float y[4];
y[1] = 2.71f;
printf("%f", y[1]);
```



Definicija varijable tipa polje

- Pri definiciji varijable potrebno je odrediti
  - ime varijable: određuje se na isti način kao ime varijable za jednostavne tipove podataka
  - tip podatka za članove polja (int, float, ...)
  - veličinu polja izraženu brojem članova polja
    - polje poznatih konstantnih dimenzija
      - dimenzija polja poznata je u trenutku prevođenja
      - u uglatim zagradama navodi se **konstantni cjelobrojni izraz**
    - polje varijabilne veličine (*variable-length array, VLA*)
      - veličina polja se utvrđuje u trenutku izvršavanja naredbe za definiciju polja (ali nakon toga se ne mijenja)
      - u uglatim zagradama navodi se **cjelobrojni izraz** (može sadržavati varijable) čiji rezultat **mora** biti veći od nule

Primjer

```
#define VELICINA_VEKTORA 50
#define BROJ_CLANOVA (5 * 10)
   ...
   // primjeri definicije polja poznatih konstantnih dimenzija
   //float vektor[50];
   float vektor[VELICINA_VEKTORA];  // bolje nego 50

   //int velicine[5 * 10];
   int velicine[BROJ_CLANOVA];      // bolje nego 5 * 10

   // primjeri definicije polja varijabilnih dimenzija (VLA)
   int n;
   scanf("%d", &n);                 // osigurati da n bude > 0 !
   float temperature[n];
   int tlakovi[2 * n];
```

Pristupanje članovima polja

- Članovima (elementima) polja pristupa se korištenjem indeksa
  - indeks može biti cjelobrojni izraz (konstante, varijable, operatori, funkcije) čiji rezultat mora biti nenegativni cijeli broj iz intervala

    [0, brojElemenataPolja - 1] (znači: indeks prvog člana je 0)
  - C prevodilac može utvrditi jedino pogrešnu upotrebu tipa podatka indeksa (npr. float umjesto int)
  - poštovanje pravila o dopuštenim granicama indeksa odgovornost je isključivo programera
    - rezultat korištenja neispravnih indeksa za vrijeme izvršavanja programa je nedefiniran: dobije se pogrešna vrijednost (logička pogreška), a pridruživanje vrijednosti uz korištenje neispravnog indeksa može, osim logičke pogreške, dovesti i do pogreške izvršavanja i prekida programa

Primjer

```
int i = 2, m, brojevi[10];
float x = 5.0f;
brojevi[0] =  0;
brojevi[1] = 10;
...
brojevi[i * 4] = 80;
brojevi[i * 4 + 1] = 90;

brojevi[x] = 1;
...
m = brojevi[10];
brojevi[-1] = 15;
```

prevodilac dojavljuje pogrešku

rezultat je nedefiniran (*garbage value*). Logička pogreška.

15 se upisuje na **pogrešno** mjesto u memoriji, što može izazvati logičku pogrešku ili pogrešku tijekom izvršavanja

Primjer

- s tipkovnice učitati 10 cijelih brojeva, ispisati njihovu aritmetičku sredinu i brojeve koji su veći od aritmetičke sredine

```
#include <stdio.h>
#define DIMENZIJA 10

int main(void) {
   int brojevi[DIMENZIJA], suma = 0, i;
   float sredina;

   for (i = 0; i < DIMENZIJA; i = i + 1) {
      scanf("%d", &brojevi[i]);
      suma = suma + brojevi[i];
   }
   sredina = 1.f * suma / DIMENZIJA;
   printf("sredina = %f\n", sredina);

   for (i = 0; i < DIMENZIJA; i = i + 1) {
      if (brojevi[i] > sredina) {
         printf("%d\n", brojevi[i]);
      }
   }
   return 0;
}
```

Primjer (kada koristiti VLA)

- s tipkovnice učitati n, zatim n cijelih brojeva, ispisati njihovu aritmetičku sredinu i brojeve koji su veći od aritmetičke sredine

```
...
int n, suma = 0, i;
float sredina;

scanf("%d", &n);
int brojevi[n];
for (i = 0; i < n; i = i + 1) {
   scanf("%d", &brojevi[i]);
   suma = suma + brojevi[i];
}
sredina = 1.f * suma / n;
printf("sredina = %f\n", sredina);

for (i = 0; i < n; i = i + 1) {
   if (brojevi[i] > sredina) {
      printf("%d\n", brojevi[i]);
   }
}
...
```

Definicija polja uz inicijalizaciju

- Članovi polja mogu se inicijalizirati u trenutku definicije polja
  - nije primjenjivo za VLA polja!
  - bez inicijalizacije, članovi polja sadrže nedefinirane vrijednosti

```
int polje[5];
```

polje

?	?	?	?	?

  - početne vrijednosti se mogu redom navesti u tzv. inicijalizatoru

```
int polje[5] = {1, 3, 5, 7, 9};
```

polje

1	3	5	7	9

  - ako se navede premalo vrijednosti, ostali članovi se postavljaju na 0

```
int polje[5] = {1, 3, 5};
```

polje

1	3	5	0	0

  - što se može iskoristiti za inicijalizaciju svih članova na nulu

```
int polje[5] = {0};
```

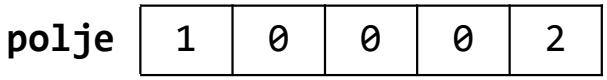
polje

0	0	0	0	0

Definicija polja uz inicijalizaciju

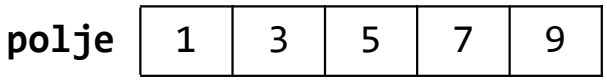
- ciljanim (designiranim) inicijalizatorom se članovi polja mogu ciljano postaviti

```
int polje[] = {1, [4] = 2};
```



- automatsko određivanje veličine polja na temelju inicijalizatora

```
int polje[] = {1, 3, 5, 7, 9};
```



- u inicijalizatoru se mora nalaziti barem jedna vrijednost

```
int polje[5] = {};
```

Prevodilac dojavljuje pogrešku

- u inicijalizatoru se ne smije napisati previše vrijednosti

```
int polje[5] = {1, 3, 5, 7, 9, 11};
```

Prevodilac dojavljuje pogrešku

Inicijalizacija nije pridruživanje!

- varijabla tipa polje ne smije se nalaziti na lijevoj strani izraza pridruživanja jer polje nije *modifiable lvalue*
  - neispravan način kopiranja sadržaja polja izvor u polje cilj:

```
int izvor[4] = {1, 2, 3, 4};          inicijalizacija polja izvor. O.K.
int cilj[4];

cilj = izvor;                         prevodilac dojavljuje pogrešku!
```

  - ispravan način kopiranja sadržaja polja izvor u polje cilj:

```
int izvor[4] = {1, 2, 3, 4};          inicijalizacija polja izvor. O.K.
int cilj[4];
int i;

for (i = 0; i < 4; i = i + 1) {
   cilj[i] = izvor[i];
}
```

Primjer

- Programski zadatak
  - s tipkovnice učitati cijeli broj n koji predstavlja broj članova polja koji će biti učitani. Ponavljati učitavanje vrijednosti za n sve dok broj članova polja ne bude ispravan. Zatim učitati n realnih članova polja i ispisati ih u obrnutom poretku od onog u kojem su učitani
  - primjer izvršavanja programa

```
Upisite broj clanova polja > 0↵
Upisite broj clanova polja > -2↵
Upisite broj clanova polja > 4↵
Upisite 1. clan > 9.1↵
Upisite 2. clan > 101.55↵
Upisite 3. clan > -476.3333↵
Upisite 4. clan > 5↵
5.0, -476.3, 101.6, 9.1↵
```

Rješenje (1. dio)

```
#include <stdio.h>

int main(void) {
   int n;   // velicina polja
   int i;   // kontrolna varijabla petlje za ucitavanje i ispis

   /* ponavljati upisivanje velicine polja dok ne bude ispravna */
   do {
      printf("Upisite broj clanova polja > ");
      scanf("%d", &n);
   } while (n < 1);

   /* definicija VLA polja */
   float polje[n];
```

Rješenje (2. dio)

```
   /* ucitavanje clanova polja */
   for (i = 0; i < n; i = i + 1) {
      printf("Upisite %d. clan > ", i + 1);
      scanf("%f", &polje[i]);
   }
   /* ispisivanje clanova polja u obrnutom poretku */
   for (i = n - 1; i >= 0; i = i - 1) {
      if (i < n - 1) {    // ispisati zarez prije svakog osim prvog
         printf(", ");
      }
      printf("%.1f", polje[i]);
   }
   printf("\n");
   return 0;
}
```

Primjer

- Programski zadatak
  - Učitavati cijele brojeve iz intervala [0, 9]. Učitavanje prekinuti kad se upiše broj izvan zadanog intervala. Zatim ispisati koliko je puta učitan svaki broj iz zadanog intervala, pri čemu treba ispisati samo one brojeve koji su učitani barem jednom.

```
Upisite broj iz intervala [0, 9] > 1↵
Upisite broj iz intervala [0, 9] > 5↵
Upisite broj iz intervala [0, 9] > 7↵
Upisite broj iz intervala [0, 9] > 5↵
Upisite broj iz intervala [0, 9] > 0↵
Upisite broj iz intervala [0, 9] > 5↵
Upisite broj iz intervala [0, 9] > 7↵
Upisite broj iz intervala [0, 9] > 10↵
↵
Broj 0 se pojavio 1 puta↵
Broj 1 se pojavio 1 puta↵
Broj 5 se pojavio 3 puta↵
Broj 7 se pojavio 2 puta↵
```

Rješenje

- Utvrđivanje frekvencije pojavljivanja brojeva
  - za svaki broj potreban je jedan brojač
  - na početku se svi brojači postave na nulu
  - kad god se učita neki broj, odgovarajući brojač se poveća za 1

brojac	0	0	0	0	0	0	0	0	0	0
	brojac[0]	brojac[1]	brojac[2]	brojac[3]	brojac[4]	brojac[5]	brojac[6]	brojac[7]	brojac[8]	brojac[9]

```
Upisite broj iz intervala [0, 9] > 1↵    → brojac[1] = brojac[1] + 1
Upisite broj iz intervala [0, 9] > 5↵    → brojac[5] = brojac[5] + 1
Upisite broj iz intervala [0, 9] > 7↵    → brojac[7] = brojac[7] + 1
Upisite broj iz intervala [0, 9] > 5↵    → brojac[5] = brojac[5] + 1
Upisite broj iz intervala [0, 9] > 0↵    → brojac[0] = brojac[0] + 1
Upisite broj iz intervala [0, 9] > 5↵    → brojac[5] = brojac[5] + 1
Upisite broj iz intervala [0, 9] > 7↵    → brojac[7] = brojac[7] + 1
```

brojac	1	1	0	0	0	3	0	2	0	0
	brojac[0]	brojac[1]	brojac[2]	brojac[3]	brojac[4]	brojac[5]	brojac[6]	brojac[7]	brojac[8]	brojac[9]

Rješenje (1. dio)

```
#include <stdio.h>
#define D_GR 0          // donja granica intervala
#define G_GR 9          // gornja granica intervala

int main(void) {
   int broj;
   int brojac[G_GR - D_GR + 1] = { 0 };   // inicijalizacija na nulu

   /* ucitavanje brojeva i inkrementiranje odgovarajucih brojaca */
   do {
      printf("Upisite broj u intervalu [%d, %d] > ", D_GR, G_GR);
      scanf("%d", &broj);
      if (broj >= D_GR && broj <= G_GR) {
         brojac[broj] = brojac[broj] + 1;
      }
   } while (broj >= D_GR && broj <= G_GR);
```

Rješenje (2. dio)

```
   printf("\n");
   /* ispis sadrzaja onih brojaca koji su veci od nule */
   int i;
   for (i = D_GR; i <= G_GR; i = i + 1) {
      if (brojac[i] > 0) {
         printf("Broj %d se pojavio %d puta\n", i, brojac[i]);
      }
   }
   return 0;
}
```

Primjer

- Programski zadatak (varijanta prethodnog zadatka - promijenjene su samo granice intervala)
  - Učitavati cijele brojeve iz intervala [1000005, 1000014]. Učitavanje prekinuti kad se upiše broj izvan zadanog intervala. Zatim ispisati koliko je puta učitan svaki broj iz zadanog intervala, pri čemu treba ispisati samo one brojeve koji su učitani barem jednom.

```
Upisite broj iz intervala [1000005, 1000014] > 1000008↵
Upisite broj iz intervala [1000005, 1000014] > 1000012↵
Upisite broj iz intervala [1000005, 1000014] > 1000012↵
Upisite broj iz intervala [1000005, 1000014] > 1000008↵
Upisite broj iz intervala [1000005, 1000014] > 1000012↵
Upisite broj iz intervala [1000005, 1000014] > 1000014↵
Upisite broj iz intervala [1000005, 1000014] > 1000000↵
↵
Broj 1000008 se pojavio 2 puta↵
Broj 1000012 se pojavio 3 puta↵
Broj 1000014 se pojavio 1 puta↵
```

Rješenje (1. dio)

```
#include <stdio.h>
#define D_GR 1000005            // donja granica intervala
#define G_GR 1000014            // gornja granica intervala

int main(void) {
   int broj;
   int brojac[G_GR - D_GR + 1] = { 0 };   // inicijalizacija na nulu

   /* ucitavanje brojeva i inkrementiranje odgovarajucih brojaca */
   do {
      printf("Upisite broj u intervalu [%d, %d] > ", D_GR, G_GR);
      scanf("%d", &broj);
      if (broj >= D_GR && broj <= G_GR) {
         brojac[broj - D_GR] = brojac[broj - D_GR] + 1;
      }
   } while (broj >= D_GR && broj <= G_GR);
```

Rješenje (2. dio)

```
   printf("\n");
   /* ispis sadrzaja onih brojaca koji su veci od nule */
   int i;
   for (i = D_GR; i <= G_GR; i = i + 1) {
      if (brojac[i - D_GR] > 0) {
         printf("Broj %d se pojavio %d puta\n", i, brojac[i - D_GR]);
      }
   }
   return 0;
}
```

Zadatak

- **Palindrom** je riječ, fraza, broj ili druga sekvenca znakova koja se čita isto naprijed i nazad.
- Primjeri palindroma:
  - Riječi: radar, level, rotor, madam
  - Fraze: "Never odd or even"
  - Brojevi: 121, 13131, 10001
  - Sekvence znakova: ABBA, racecar

Zadatak: koliko ima (cjelobrojnih) palindroma u intervalu [1, 100000]?

Prije sljedećeg predavanja

- Edgar:
  - Tutorial: **06. Prije sedmog predavanja**
  - **6. vježbe uz predavanja**