Uvod u programiranje

- predavanja -

listopad 2025.

4. Kontrola toka programa

if-else-if, switch, do-while

Vodič „03 Prije četvrtog predavanja”

- Očekuje se da ste savladali (na razini na kojoj je obrađeno u vodiču):
  - Stil pisanja programa
  - Stilske preporuke

- Pitanja?

Primjer

- Programski zadatak
  - s tipkovnice učitati tri različita cijela broja (nije potrebno kontrolirati jesu li brojevi ispravno upisani)
  - na zaslon ispisati najveću učitanu vrijednost
  - primjer izvršavanja programa

```
Upisite tri razlicita cijela broja > 1 17 -2↵
Najveci broj je 17↵
```

Rješenje (1. varijanta)

```
#include <stdio.h>

int main(void) {
   int x, y, z, rez;
   printf("Upisite tri razlicita cijela broja > ");
   scanf("%d %d %d", &x, &y, &z);
   if (x > y) {
      if (x > z) {
         rez = x;
      } else {
         rez = z;
      }
   } else {
      if (y > z) {
         rez = y;
      } else {
         rez = z;
      }
   }
   printf("Najveci broj je %d\n", rez);
   return 0;
}
```

Rješenje (2. varijanta)

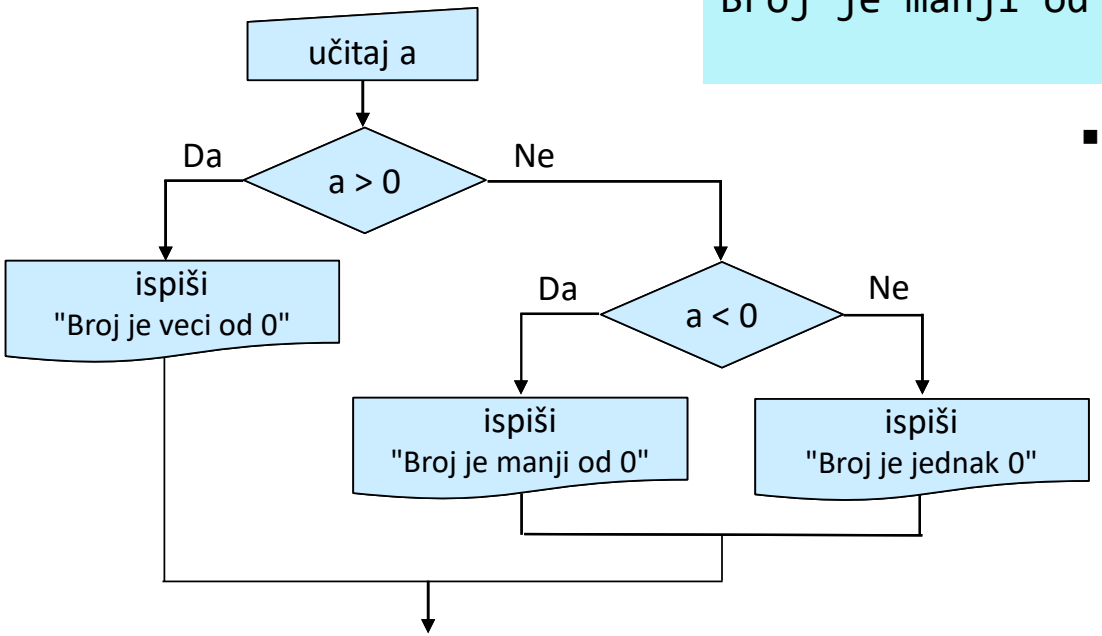
```
#include <stdio.h>

int main(void) {
   int x, y, z, rez;
   printf("Upisite tri razlicita cijela broja > ");
   scanf("%d %d %d", &x, &y, &z);
   rez = x;
   if (y > rez)
      rez = y;
   if (z > rez)
      rez = z;
   printf("Najveci broj je %d\n", rez);
   return 0;
}
```

Kaskadna selekcija

Primjer

- Programski zadatak
  - učitati cijeli broj. Ovisno o učitanoj vrijednosti, na zaslon ispisati jednu od poruka "Broj je veci od 0", "Broj je jednak 0" ili "Broj je manji od 0"

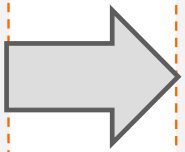
```
Upisite cijeli broj > -47↵
Broj je manji od 0↵
```



- kaskadna selekcija: obavljanje niza testova koje se zaustavlja u prvom slučaju u kojem je neki od uvjeta zadovoljen

Rješenje

```c
#include <stdio.h>

int main(void) {
   int a;

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

   if (a > 0) {
      printf("Broj je veci od 0\n");
   } else {
      if (a < 0) {
         printf("Broj je manji od 0\n");
      } else {
         printf("Broj je jednak 0\n");
      }
   }
   return 0;
}
```



```c
#include <stdio.h>

int main(void) {
   int a;

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

   if (a > 0) {
      printf("Broj je veci od 0\n");
   } else if (a < 0) {
      printf("Broj je manji od 0\n");
   } else {
      printf("Broj je jednak 0\n");
   }

   return 0;
```

Primijetiti da je u žutim zagradama samo jedna naredba – if-else.
Što ako izostavimo zagrade?

Primjer

- Programski zadatak
  - učitati cijeli broj koji predstavlja brojčanu ocjenu. Ovisno o učitanoj vrijednosti, na zaslon ispisati jednu od poruka:
    - izvrstan
    - vrlo dobar
    - dobar
    - dovoljan
    - nedovoljan
    - neispravna ocjena

Rješenje (odsječak programa)

```
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");
            }
         }
      }
   }
}
```

- program je napisan stilski ispravno
- ipak, zbog velikog broja selekcija u kaskadi postaje težak za pisanje i čitanje. Lako je pogriješiti u broju zagrada i indentaciji za koju se troši previše prostora
- također primijetiti da su vitičaste zagrade iza else (osim zadnjeg) nepotrebne jer se iza else uvijek nalazi samo jedna naredba - sljedeća naredba if

- napišimo program malo drugačije: uklonimo nepotrebne vitičaste zagrade i suvišnu indentaciju

Rješenje (odsječak programa)

```
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");
}
```

dodatno: u ovom konkretnom primjeru smiju se ispustiti sve vitičaste zagrade

- Uočiti:
  - pojednostavljuje se pisanje i povećava preglednost koda
  - programeri će lako uočiti da se radi o ničem drugom nego o nizu ispitivanja uvjeta koji jedan drugog isključuju

Objašnjenje

- U nekim programskim jezicima postoji poseban oblik naredbe za kaskadnu selekciju. Npr. u jeziku Python:

```
if ocj == 5:
   print "izvrstan"
elif ocj == 4:
   print "vrlo dobar"
elif ocj == 3:
   print "dobar"
elif ocj == 2:
   print "dovoljan"
elif ocj == 1:
   print "nedovoljan"
else:
   print "neispravna ocjena"
```

Objašnjenje

- U nekim programskim jezicima postoji poseban oblik naredbe za kaskadnu selekciju (npr. u jeziku Python).
- U programskom jeziku C **ne postoji** posebna naredba za **kaskadnu selekciju**
  - koristi se niz ugniježđenih dvostranih selekcija, koje se samo radi preglednosti pišu na karakterističan način. Praktički, **radi se o stilu pisanja, a ne o posebnoj naredbi za kaskadnu selekciju:**

```
if (logički_izraz_1)
   naredba_1;
else if (logički_izraz_2)
   naredba_2;
else if (logički_izraz_3)
   naredba_3;
...
else
   naredba_n;
```

- prema potrebi, na mjestima pojedinačnih naredbi (naredba_1, naredba_2, ...) mogu se koristiti složene naredbe
- prema potrebi, posljednji else i pripadna naredba se mogu izostaviti

Moguće dvojbe u vezi dijela naredbe else

- **Važno pravilo:** else dio naredbe za selekciju "pripada" najbližoj if naredbi (koja već nema "svoj" else dio)
- Nepažljivo ugniježđena naredba za selekciju koja nema svoj else dio, može dovesti do kasnije teško uočljive logičke pogreške
  - Primjer: ako je cjelobrojna vrijednost b različita od nule, tada provjeriti je li cjelobrojna vrijednost a djeljiva s b, i ako jest, ispisati poruku "a je djeljiv s b". Inače (ako je b jednaka nuli), ispisati poruku "b je nula"

Pogrešno! Indentacija ne pomaže!

```
if (b != 0)
   if (a % b == 0)
      printf("a je djeljiv s b");
else
   printf("b je nula");
```

Ispravno

```
if (b != 0) {
   if (a % b == 0)
      printf("a je djeljiv s b");
} else
   printf("b je nula");
```

Skretnica - switch

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 (x < 0)
   printf("x je negativan");
else if (x > 0)
   printf("x je pozitivan");
else
   printf("x je nula");
```

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:` nije obavezno

* 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 s 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 s 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
  1. programske petlje s ispitivanjem uvjeta **na početku**
     - ovisno o početnim uvjetima može se dogoditi da se tijelo petlje uopće neće izvršiti
  2. programske petlje s ispitivanjem uvjeta **na kraju**
     - tijelo petlje će se izvršiti barem jednom
  3. programske petlje s **poznatim brojem ponavljanja**
     - broj ponavljanja može se unaprijed izračunati i (u principu) ne ovisi o izvršavanju tijela petlje

1. Programska petlja while

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

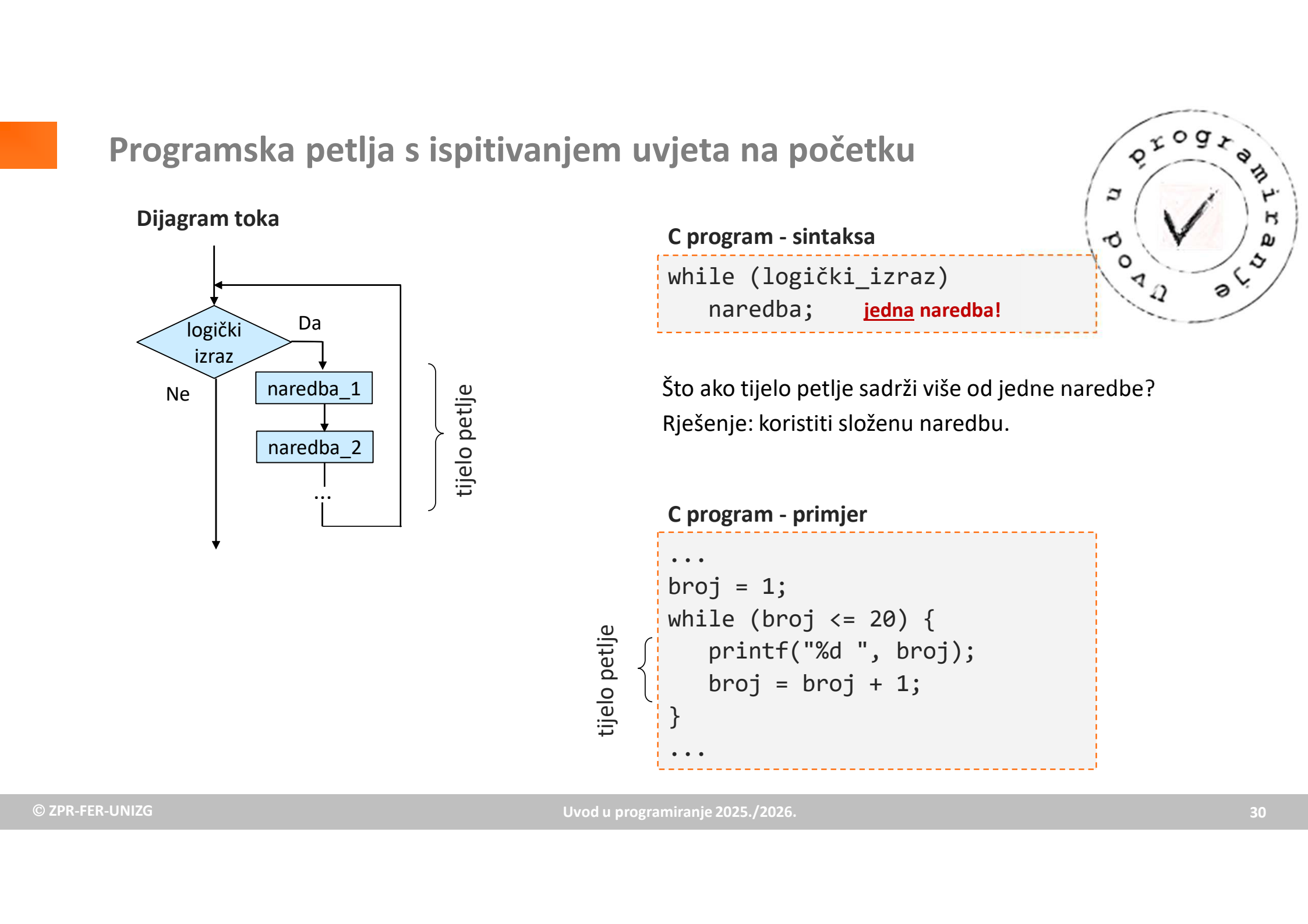
- modificirati prethodni program. Umjesto konstante koristiti varijablu. Sada će isti niz naredbi obavljati mnogo puta

```
#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

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

tijelo petlje

```
...
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 zbrajati 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;
}
```

Zadatak

- Unesite cijeli broj i rastavite ga na proste faktore

Primjer obavljanja:

```
Unesite broj: 750

750 = 2 * 3 * 5 * 5 * 5
```

Zadatak

- **Newtonovom** metodom izračunajmo **drugi korijen** pozitivnog realnog broja:
  - kao početna vrijednost za y se odabire vrijednost 1.
  - porastom broja iteracija y postaje sve bliži točnoj vrijednosti drugog korijena od x
  - Prestati računanje kad apsolutna vrijednost razlike y u dva uzastopna koraka bude manja od 0.1%.

x	y	x/y	Prosjek (y, x/y)
3	1.0	3.	2.0
3	2	1.5	1.75
3	1.75	1.71429	1.73214
3	1.73214	1.73196	1.73205

Prije sljedećeg predavanja

- Edgar:
  - Tutorial: **04. Prije petog predavanja**
  - Public exams: **4. vježbe uz predavanja – kontrola toka programa**