Uvod u programiranje

- predavanja -

listopad 2019.

3. Kontrola toka programa

Kontrola toka programa, kontrolne strukture

- jedan od najvažnijih aspekata programiranja jest mogućnost upravljanja redoslijedom izvršavanja naredbi (*flow control*): propisivanje koja naredba će se izvršiti u sljedećem koraku izvršavanja programa. U tu svrhu koriste se kontrolne programske strukture (*programming control structures*)
  - selekcija
    - jednostrana (*if ... then ...*)
    - dvostrana (*if ... then ... else ...*)
    - skretnica (*case ... then ..., case ... then ..., case ... then ..., ...*)
  - petlja
    - ponavljanje naredbi dok je zadovoljen uvjet (*condition-controlled*)
    - ponavljanje naredbi određeni broj puta (*count-controlled*)

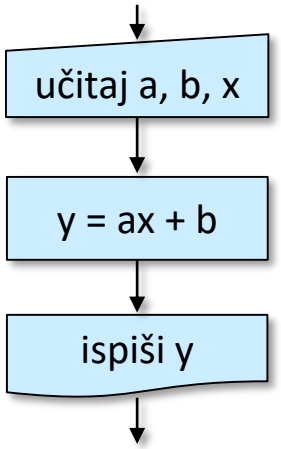
Niz ili sekvenca

- niz naredbi koje se izvršavaju točno jedna iza druge, redom kako su napisane

Pseudo-kod

```
...
učitaj(a, b, x)
y := ax + b
ispiši(y)
...
```

Dijagram toka



C program

```
...
scanf("%f %f %f", &a, &b, &x);
y = a * x + b;
printf("%f", y);
...
```

Selekcija

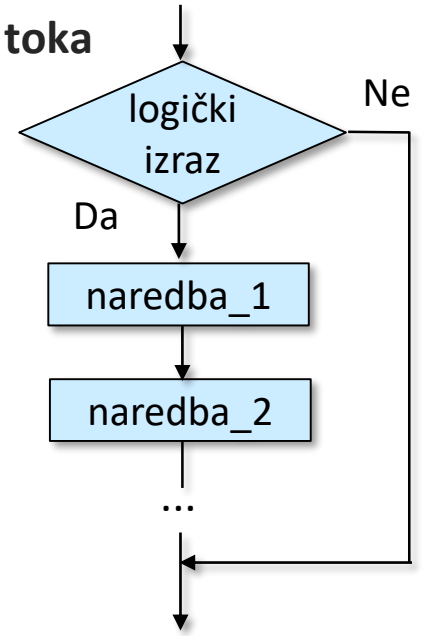
Jednostrana selekcija

Jednostrana selekcija - if

Pseudo-kod

```
...
ako je logički_izraz tada
   | naredba_1
   | naredba_2
   | ...
...
```

Dijagram toka



C program - sintaksa

```
if (logički_izraz)
   naredba;
```

jedna naredba!

C program - primjer

```
...
if (ocjena > 1)
   printf("Ispit je polozen!");
...
```

Što ako u slučaju, kada je ocjena pozitivna, treba izvršiti više od jedne naredbe?

Rješenje: koristiti složenu naredbu

Složena naredba (*compound statement*)

- **problem:** u naredbi if dopušteno je navođenje samo jedne naredbe koja će se izvršiti kada je rezultat logičkog izraza istina. Što učiniti ako je potrebno izvršiti više od jedne naredbe?
- **rješenje:** koristiti složenu naredbu (blok, blok naredbi)
  - jedna ili više naredbi omeđenih vitičastim zagradama
  - može se koristiti na mjestima gdje je prema sintaksi predviđena samo jedna naredba (u naredbama if, while, ...)
  - složena naredba se nikad ne terminira znakom ;

Sintaksa složene naredbe

```
{
   naredba_1;
   naredba_2;
   ...
}
```

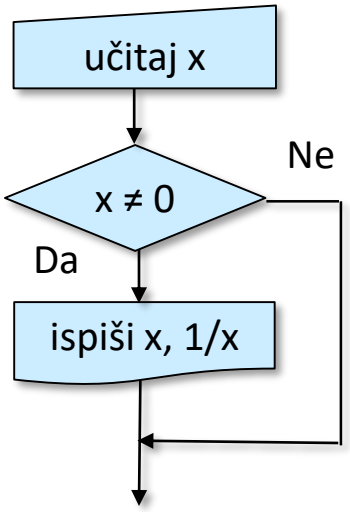
Primjer korištenja složene naredbe

```
...
if (ocjena > 1) {
   printf("Ispit je polozen.");
   brojPol = brojPol + 1;
}
...
```

Primjer

- Programski zadatak
  - u varijablu x s tipkovnice učitati realni broj. Ako je recipročna vrijednost za x definirana, na zaslon ispisati vrijednost varijable x i njezinu recipročnu vrijednost
- Pseudo-kod i dijagram toka

```
učitaj(x)
ako je x ≠ 0 tada
   ispiši(x, 1/x)
```



Primjer: C program

```c
#include <stdio.h>

int main(void) {
   float x;

   scanf("%f", &x);
   if (x != 0)
      printf("%f %f", x, 1/x);
   return 0;
}
```

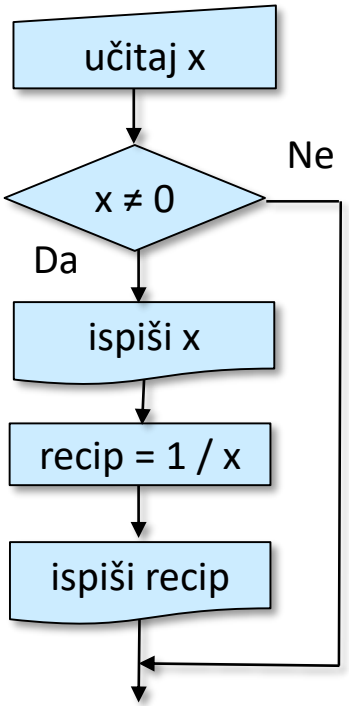
```c
#include <stdio.h>

int main(void) {
   float x;

   scanf("%f", &x);
   if (x != 0) {
      printf("%f %f", x, 1/x);
   }
   return 0;
}
```

- obje varijante su ispravne, međutim
  - iako se radi o samo jednoj naredbi koju treba obaviti kada je uvjet zadovoljen, **preporuča se** koristiti oblik složene naredbe, tj. naredbu omeđiti vitičastim zagradama
    - smanjuje se mogućnost logičke pogreške koja može nastati nepažljivim prepravljanjem programa, kao što je prikazano u sljedećem primjeru

Primjer

- Programski zadatak vrlo sličan prethodnom
  - u varijablu x s tipkovnice učitati realni broj. Ako je recipročna vrijednost za x definirana, na zaslon ispisati sadržaj varijable x, nakon toga izračunati i zatim ispisati recipročnu vrijednost
- Pseudo-kod i dijagram toka

```
učitaj(x)
ako je x ≠ 0 tada
    ispiši(x)
    recip := 1 / x
    ispiši(recip)
```



- Dvije nove varijante programa pokušat će se napisati prepravljanjem dvaju programa iz prethodnog primjera

Primjer: C program

```c
#include <stdio.h>

int main(void) {
   float x, recip;

   scanf("%f", &x);

   if (x != 0)
      printf("%f", x);
      recip = 1 / x;
      printf(" %f", recip);
   return 0;
}
```

```c
#include <stdio.h>

int main(void) {
   float x, recip;

   scanf("%f", &x);

   if (x != 0) {
      printf("%f", x);
      recip = 1 / x;
      printf(" %f", recip);
   }
   return 0;
}
```

- program na lijevoj strani je neispravan! Prepravljajući lijevu varijantu programa iz prethodnog primjera programer je previdio da je trebalo dodati vitičaste zagrade i time napravio teško uočljivu logičku pogrešku

Primjeri

- U nekim slučajevima, radi kompaktnosti i preglednosti programa, prikladno je za jednu naredbu ne koristiti vitičaste zagrade . Npr.

```
// varijablu a postavi na njenu apsolutnu vrijednost
if (a < 0) a = -1 * a;
```

- Oprez!

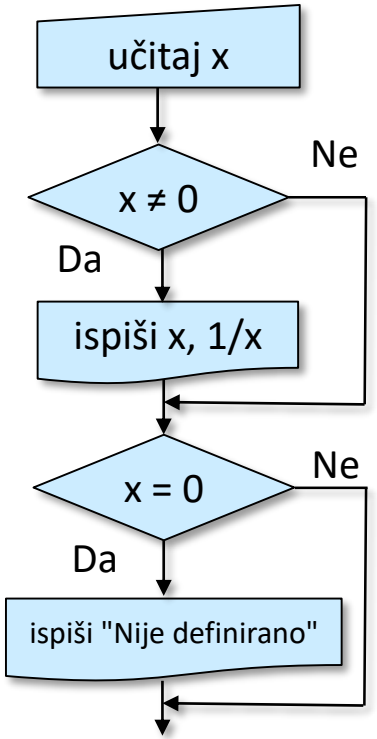
```
if (a < 0); a = -1 * a;
```

  - C prevodilac ovo smatra (sintaktički) ispravnim. "Ništa" terminirano znakom `;` naziva se nul-naredba (*null-statement*). Sasvim očekivano, nul-naredba ne obavlja nikakvu akciju
    - analizirati posljedice ove logičke greške

Primjer

- Programski zadatak
  - u varijablu x s tipkovnice učitati realni broj. Ako je recipročna vrijednost za x definirana, na zaslon ispisati vrijednost varijable x i njezinu recipročnu vrijednost, inače ispisati poruku "Nije definirano"
- **Loš** pseudo-kod i **loš** dijagram toka

```
učitaj(x)
ako je x ≠ 0 tada
   ispiši(x, 1/x)
ako je x = 0 tada
   ispiši("Nije definirano)
```



Selekcija

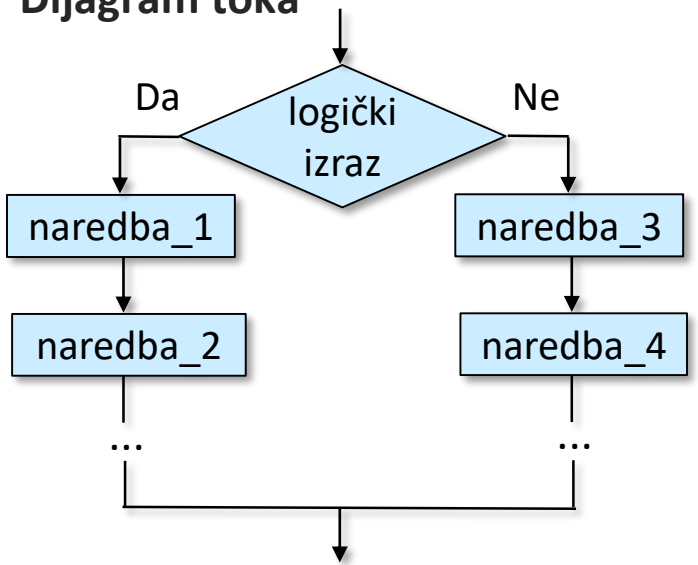
Dvostrana

Dvostrana selekcija - if - else

Pseudo-kod

```
...
ako je logički_izraz tada
   | naredba_1
   | naredba_2
   | ...
inače
   | naredba_3
   | naredba_4
   | ...
...
```

Dijagram toka



C program - sintaksa

```
if (logički_izraz)
   naredba_1;    jedna naredba!
else
   naredba_2;    jedna naredba!
```

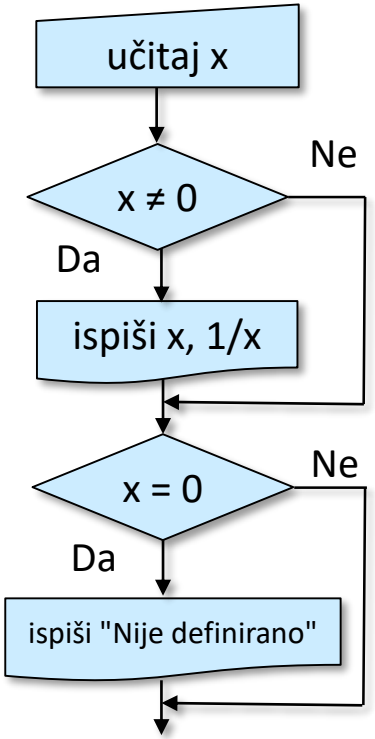
C program - primjer

```
if (ocjena > 1)
   printf("Ispit je polozen!");
else
   printf("Ispit nije polozen!");
```

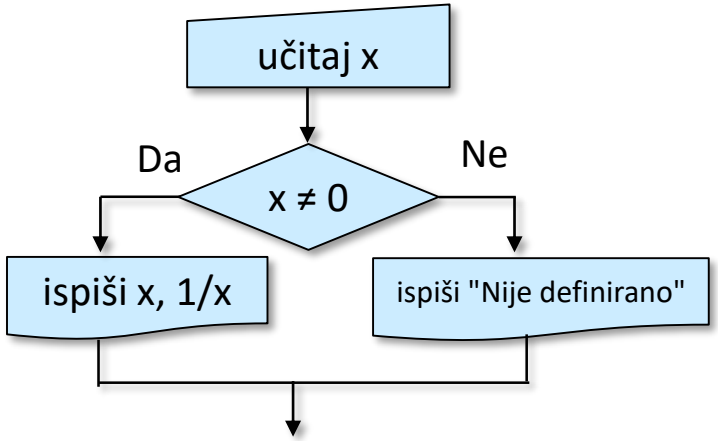
Primjer

- Ispravak lošeg rješenja za izračunavanje recipročne vrijednosti

Loše



Dobro




```
if (x != 0)
   printf("%f %f", x, 1/x);
else
   printf("Nije definirano");
```

Primjer

- Programski zadatak
  - Na zaslon ispisati poruku `Upisite cijeli broj >`
  - Učitati cijeli broj s tipkovnice
  - Izračunati apsolutnu vrijednost učitanog broja te na zaslon ispisati učitani broj i njegovu apsolutnu vrijednost
  - Primjer izvršavanja programa

```
Upisite cijeli broj > -47↵
Apsolutna vrijednost od -47 je 47↵
```

Rješenje (loše!)

```
#include <stdio.h>

int main(void) {
   int broj, aps;

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

   if (broj < 0) {
      aps = -1 * broj;                  // aps = -broj;
      printf("Apsolutna vrijednost od %d je %d\n", broj, aps);
   } else {
      aps = broj;
      printf("Apsolutna vrijednost od %d je %d\n", broj, aps);
   }
   return 0;
}
```

Rješenje (dobro!)

```
#include <stdio.h>

int main(void) {
   int broj, aps;

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

   if (broj < 0) {
      aps = -1 * broj;              // aps = -broj;
   } else {
      aps = broj;
   }

   printf("Apsolutna vrijednost od %d je %d\n", broj, aps);
   return 0;
}
```

Rješenje (također dobro!)

```
#include <stdio.h>
#include <stdlib.h>

int main(void) {
   int broj;

   printf("Upisite cijeli broj > ");
   scanf("%d", &broj);
   printf("Apsolutna vrijednost od %d je %d\n", broj, abs(broj));
   return 0;
}
```

Primjer

- Programski zadatak
  - Na zaslon ispisati poruku `Upisite cijeli broj >`
  - Učitati cijeli broj s tipkovnice i ovisno o učitanoj vrijednosti ispisati jednu, obje ili niti jednu od sljedećih poruka:
    - Broj je pozitivan
    - Broj je paran
- Primjeri izvršavanja programa

```
Upisite cijeli broj > 12↵
Broj je pozitivan↵
Broj je paran↵
```

```
Upisite cijeli broj > 13↵
Broj je pozitivan↵
```

```
Upisite cijeli broj > -12↵
Broj je paran↵
```

```
Upisite cijeli broj > 0↵
Broj je paran↵
```

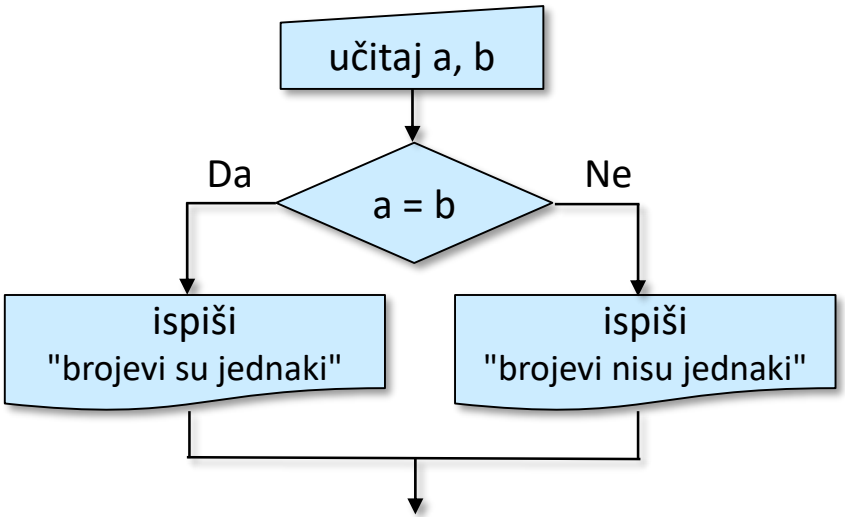
```
Upisite cijeli broj > -47↵
```

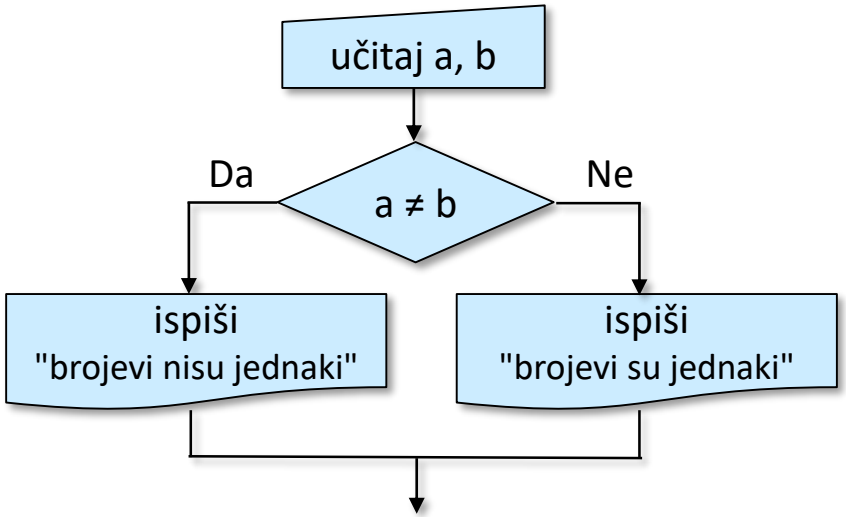
Rješenje

```
#include <stdio.h>

int main(void) {
   int broj;

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

   if (broj > 0) {
      printf("Broj je pozitivan\n");
   }

   if (broj % 2 == 0) {
      printf("Broj je paran\n");
   }
   return 0;
}
```

Primjer

- Programski zadatak
  - S tipkovnice učitati dva cijela broja. Ovisno o učitanim vrijednostima, na zaslon ispisati ili poruku "brojevi su jednaki" ili poruku "brojevi nisu jednaki"





  - uočiti: rješenja su jednako vrijedna
  - za vježbu: napisati C programe za oba dijagrama

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;
}
```

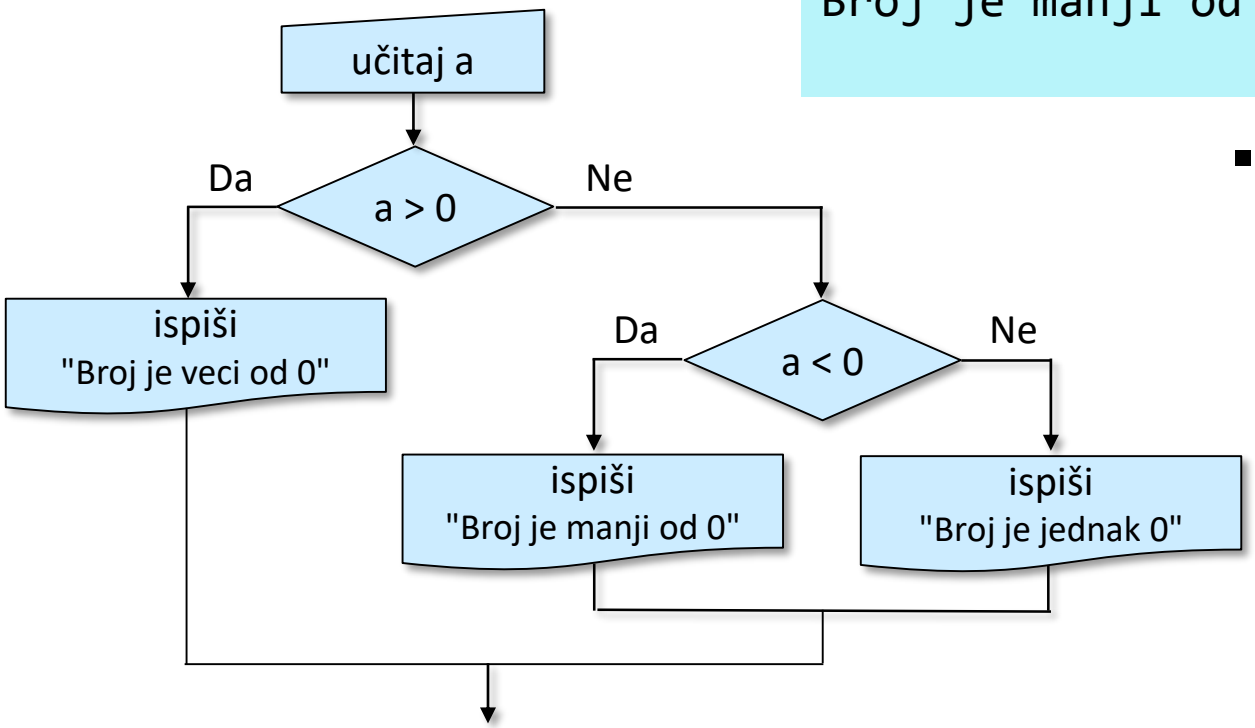
Selekcija

Kaskadna

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

```
#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;
}
```

Stil pisanja programa

Važna digresija

Stilovi pisanja programa

- praznine, tabulatori ili skokovi u novi red u programskom jeziku C nemaju specijalno značenje
  - npr. sljedeći program će se korektno prevesti:

```
#include <stdio.h>
int main(void){int a;scanf("%d",&a);if(a%2==0)printf(
"Paran");else printf("Neparan");return 0;}
```

  - tako napisane programe čovjek će vrlo teško razumjeti. Zato je dobro odabrati i pri pisanju programa čvrsto se držati neke od konvencija za pisanje koda

Thus, programs must be written for people to read, and only incidentally for machines to execute.

H. Abelson, J. Sussman: Structure and Interpretation of Computer Programs; MIT Press, 1984.

Stilovi pisanja programa

- stilski korektan program
  - je lakše razumjeti - stoga je vjerojatnije da je ispravan
  - je lakše održavati - stoga je vjerojatnije da će i ostati ispravan
- postoje mnogi različiti stilovi pisanja C programa
  - ne postoji "najbolji" stil. Koji će se stil koristiti može ovisiti o različitim faktorima
    - kompanijskim pravilima, osobnim preferencijama, ...
  - primjeri različitih stilova pisanja jednostrane selekcije

```
if (x > y) {
   rez = x;
}
```

```
if (x > y)
{
   rez = x;
}
```

```
if (x > y)
   {
   rez = x;
   }
```

```
if (x > y)
   {
      rez = x;
   }
```

  - najvažnije je: odabrati jedan stil i konzistentno ga primjenjivati najmanje u okviru istog projekta

Stilovi pisanja programa

- na predmetu Uvod u programiranje **obavezno je** pridržavati se preporuka koje su prikazane u nastavku ovih predavanja
  - ovdje prikazane preporuke temelje se na standardima pisanja programskog koda u projektu LLVM
    - LLVM Coding Standards
    - https://llvm.org/docs/CodingStandards.html

- neki drugi poznati stilovi za pisanje C/C++ programa:
  - Google
  - WebKit
  - Microsoft Windows

Stilovi pisanja programa - preporuke

- svaku naredbu treba započeti u novom retku. Redak ne bi trebao biti dulji od 80 znakova
- blok naredbi koji je logički podređen treba uvući za određeni broj mjesta. Kolokvijalno se taj postupak naziva *indentacija*
  - na predmetu UPRO: 3 znaka praznine, ne koristiti tabulator
- oznaku početka bloka **{** napisati na kraju retka koji je neposredno nadređen tom bloku, a oznaku završetka bloka **}** točno ispod početka tog istog retka

```
if (x > y) {
    rez = x;
    if (rez == 0) {
        z = 10;
    }
    printf("%d", z);
}
```

Stilovi pisanja programa - preporuke

- ključna riječ else treba se nalaziti u istom retku u kojem se nalazi znak **}** kojim je zatvoren blok naredbi iza if. Slično vrijedi za ključnu riječ while u petlji do-while (objašnjeno kasnije)

```
if (x > y) {
   rez = x;
} else {
   rez = y;
}
```

```
do {
   ...
} while (brojac < 100);
```

Stilovi pisanja programa - preporuke

- praznim redcima razdvojiti logičke cjeline programa
  - direktive pretprocesoru
  - definicije varijabli
  - ostale ključne funkcionalne cjeline (čitanje, izračunavanje, ...)

```
#include <stdio.h>

int main(void) {
   int a;
   float x, y;

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

   if (a > 0) {
      printf("Broj je veci od 0\n");
   ...
```

Stilovi pisanja programa - preporuke

- umetnuti prazninu prije znaka **{** kojim započinje blok

```
int main(void) {
   printf("%d", r);
```

- umetnuti prazninu iza zareza, ali ne prije zareza

```
scanf("%d %d", &m, &n);
```

- umetnuti prazninu iza ključne riječi koja upravlja programskim slijedom (if, switch, while, for)

```
if (x > y) {
```

Stilovi pisanja programa - preporuke

- umetnuti prazninu između operatora i operanada, operatora i zagrada, ali ne i između zagrada i operanada

```
if (m > n) {
   r = (m + n) * 10;
```

- izuzetak od pravila su unarni operatori i sljedeći binarni operatori
  - . ->
  - značenje tih operatora bit će objašnjeno kasnije

```
godina = p_student->god_rod;
s_student.prosj_ocj = 4.5f;
godina++;
a = -a;
```

Stilovi pisanja programa - preporuke

- ne stavljati praznine između imena funkcije i otvorene zagrade, nakon otvorene zagrade, prije zatvorene zagrade niti između zatvorene zagrade i znaka ;

```
scanf("%d %d", &m, &n);
```

- složene izraze logično razlomiti u više redaka i vertikalno uskladiti

```
if (mjesec == 1 && dan == 1 || mjesec == 5 && dan ==  1 || mjesec == 6 &&
    dan == 22 || mjesec == 6 && dan == 25 || mjesec == 8 && dan == 10) {
   printf("Drzavni praznik");
```

```
if (mjesec == 1 && dan ==  1 ||
    mjesec == 5 && dan ==  1 ||
    mjesec == 6 && dan == 22 ||
    mjesec == 6 && dan == 25 ||
    mjesec == 8 && dan ==  5) {
   printf("Drzavni praznik");
}
```

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

Bolje rješenje jednog od prethodnih primjera

```
#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;
}
```

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 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

Primjer

- Programski zadatak
  - učitati realni broj T koji predstavlja izmjerenu tjelesnu temperaturu. Ovisno o učitanoj vrijednosti na zaslon ispisati jedno od upozorenja:
    - Temperatura je blago povisena za $37.0 \leq T < 38.0$
    - Temperatura je znacajno povisena za $38.0 \leq T < 39.0$
    - Temperatura je opasno povisena za $T \geq 39$
  - Primjer izvršavanja programa

```
Upisite temperaturu > 39↵
Temperatura je opasno povisena↵
```

Rješenje

```
#include <stdio.h>

int main(void) {
   float temp;

   printf("Upisite temperaturu > ");
   scanf("%f", &temp);

   if (temp >= 37.0f && temp < 38.0f) {
      printf("Temperatura je blago povisena\n");
   } else if (temp >= 38.0f && temp < 39.0f) {
      printf("Temperatura je znacajno povisena\n");
   } else if (temp >= 39.0f) {
      printf("Temperatura je opasno povisena\n");
   }
   return 0;
}
```

i u ovom konkretnom primjeru se sve vitičaste zagrade smiju ispustiti

nema "zadnjeg else" jer u konkretnom primjeru nije potreban

Moguće dileme 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");
```

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

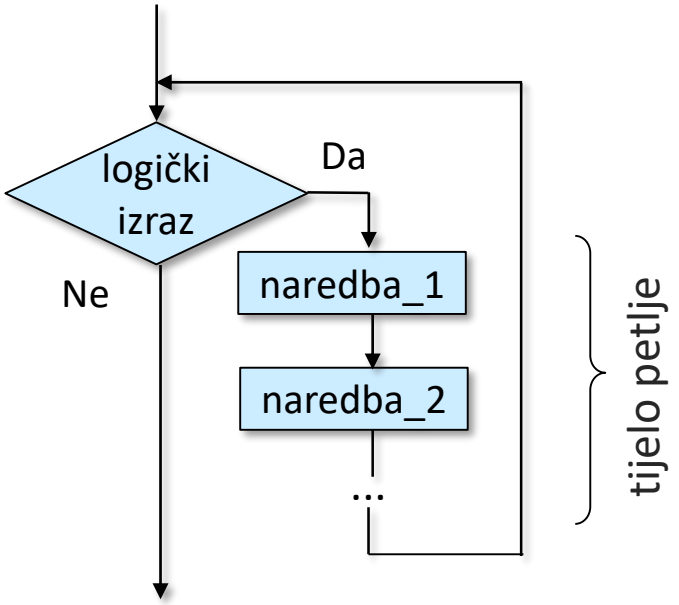
```
#include <stdio.h>

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

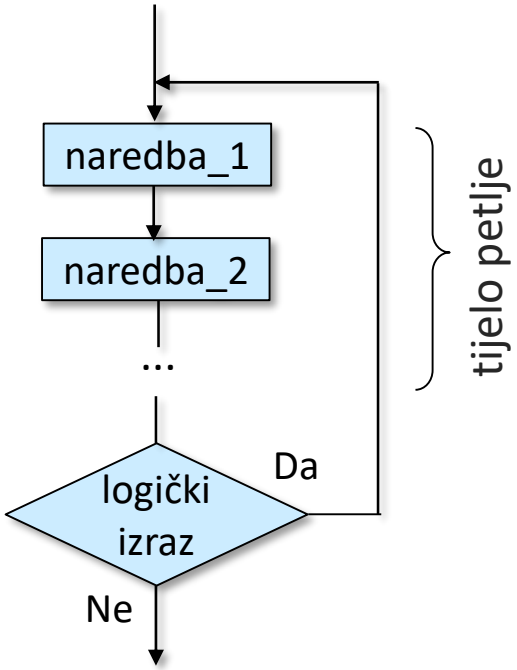
```
#include <stdio.h>

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.

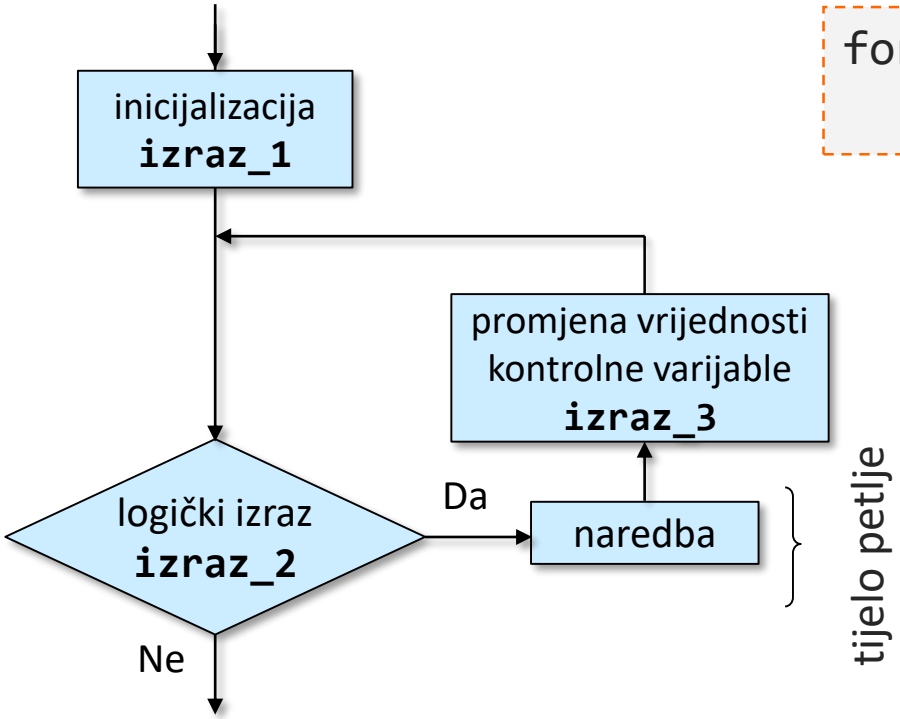
C program - primjer Ispis neparnih brojeva iz intervala [1, 10]

```
...
  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

Primjer

- Programski zadatak
  - Učitati nenegativan cijeli broj n (nije potrebno provjeravati je li učitan ispravan broj). Ispisati prvih n članova Fibonaccijevog niza.
  - definicija niza:

    $a_1 = a_2 = 1$

    $a_i = a_{i-1} + a_{i-2}$ za $i > 2$

  - Primjer izvršavanja programa:

```
Upisite broj clanova Fibonaccijevog niza > 15↵
1, 1, 2, 3, 5, 8, 13, 21, 34, 55, 89, 144, 233, 377, 610
```

Rješenje

```
#include <stdio.h>

int main(void) {
   int n, i, a_i_minus2, a_i_minus1 = 1, a_i = 1;

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

   for (i = 1; i <= n; i = i + 1) {
      if (i > 2) {
         a_i_minus2 = a_i_minus1;
         a_i_minus1 = a_i;
         a_i = a_i_minus1 + a_i_minus2;
      }
      if (i > 1) printf(", ");
      printf("%d", a_i);
   }
   return 0;
}
```

Primjer

- Programski zadatak
  - Ispisati tablicu množenja do 100, u 10 redaka i 10 stupaca.
  - Primjer izvršavanja programa:

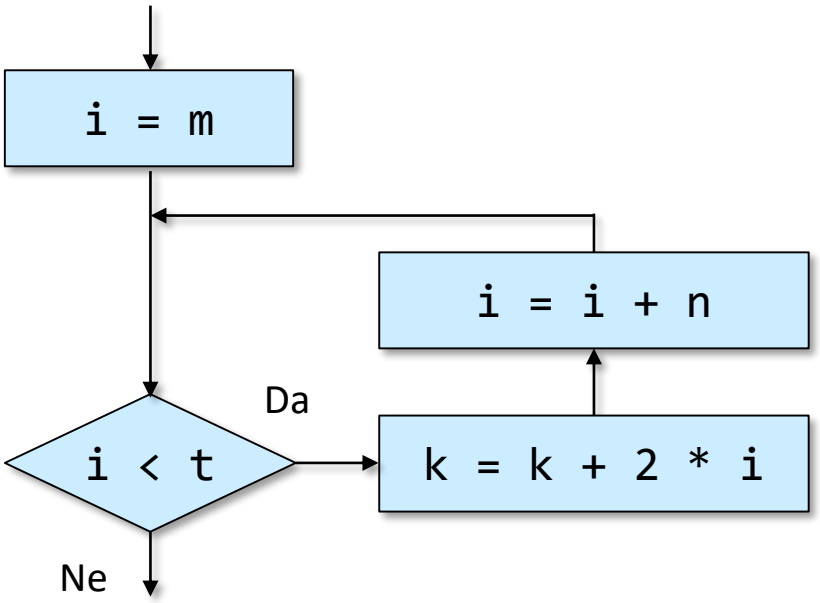
```
  1   2   3   4   5   6   7   8   9  10
  2   4   6   8  10  12  14  16  18  20
  3   6   9  12  15  18  21  24  27  30
  4   8  12  16  20  24  28  32  36  40
  5  10  15  20  25  30  35  40  45  50
  6  12  18  24  30  36  42  48  54  60
  7  14  21  28  35  42  49  56  63  70
  8  16  24  32  40  48  56  64  72  80
  9  18  27  36  45  54  63  72  81  90
 10  20  30  40  50  60  70  80  90 100
```

Rješenje

```
#include <stdio.h>

int main(void) {
   int redak, stupac;

   for (redak = 1; redak <= 10; redak = redak + 1) {
      for (stupac = 1; stupac <= 10; stupac = stupac + 1) {
         printf("%4d", redak * stupac);
      }
      printf("\n");
   }
   return 0;
}
```

Primjer:
Realizacija istog algoritma raznim vrstama programskih petlji (1)

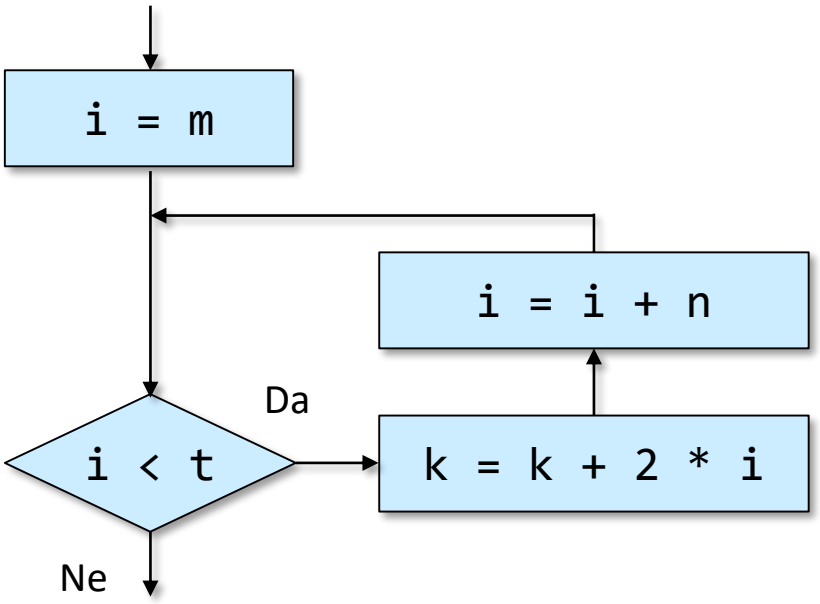
- Programski odsječak prikazan dijagramom toka treba realizirati petljom s ispitivanjem uvjeta ponavljanja na početku




```
i = m;
while (i < t) {
    k = k + 2 * i;
    i = i + n;
}
```

Primjer:
Realizacija istog algoritma raznim vrstama programskih petlji (2)

- Programski odsječak prikazan dijagramom toka treba realizirati petljom s poznatim brojem ponavljanja

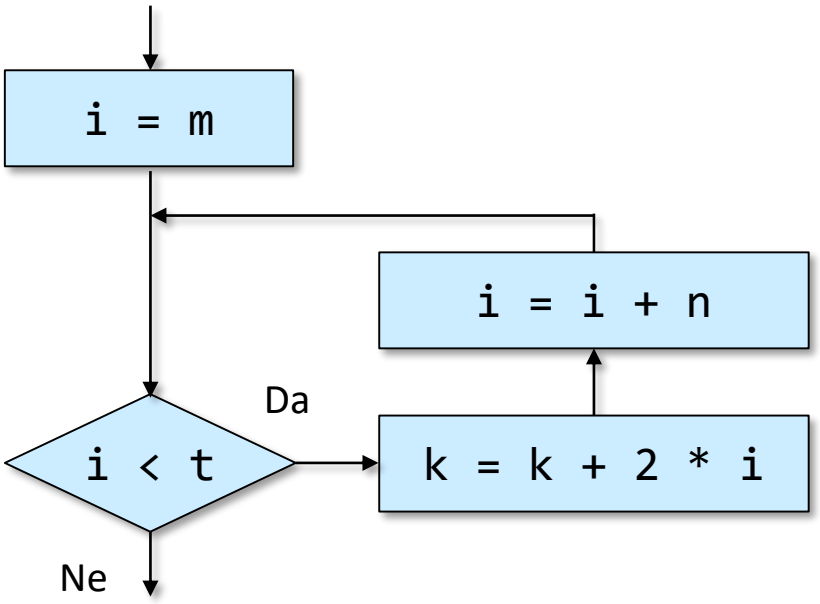



```
for (i = m; i < t; i = i + n) {
    k = k + 2 * i;
}
```

Prednost for petlje u ovom slučaju je u tome što se početna inicijalizacija brojača, ispitivanje uvjeta i korak brojača nalaze na jednom mjestu u kodu.

Primjer:
Realizacija istog algoritma raznim vrstama programskih petlji (3)

- Programski odsječak prikazan dijagramom toka treba realizirati petljom s ispitivanjem uvjeta ponavljanja na kraju




```
i = m;
if (i < t) {
    do {
        k = k + 2 * i;
        i = i + n;
    } while (i < t)
}
```

Nedostatak do-while petlje u ovom slučaju je u tome što je nužna dodatna provjera treba li obaviti prvi prolaz kroz tijelo petlje.

Primjer: odabir vrste petlje

- Programski zadatak
  - Učitati nenegativan cijeli broj n (nije potrebno provjeravati je li učitan ispravan broj). Izračunati i ispisati n faktorijela
  - Zadatak riješiti
    - petljom s ispitivanjem uvjeta na početku
    - petljom s ispitivanjem uvjeta na kraju
    - petljom s poznatim brojem ponavljanja
  - Procijeniti koja je vrsta petlje najprikladnija
  - Primjeri izvršavanja programa:

```
Upisite n > 12↵
12! = 479001600
```

```
Upisite n > 0↵
0! = 1
```

Rješenje (1)

- programska petlja s ispitivanjem uvjeta na početku

```
#include <stdio.h>

int main(void) {
   int n, i, fakt;

   scanf("%d", &n);
   fakt = 1;
   i = 2;
   while (i <= n) {
      fakt = fakt * i;
      i = i + 1;
   }

   printf("%d! = %d", n, fakt);
   return 0;
}
```

Rješenje (2)

- programska petlja s ispitivanjem uvjeta na kraju

```
#include <stdio.h>

int main(void) {
   int n, i, fakt;

   scanf("%d", &n);
   fakt = 1;
   i = 1;
   do {
      fakt = fakt * i;
      i = i + 1;
   } while (i <= n);

   printf("%d! = %d", n, fakt);
   return 0;
}
```

ne smije se početi od 2 zbog do-while

Rješenje (3)

- programska petlja s poznatim brojem ponavljanja

```
#include <stdio.h>

int main(void) {
   int n, i, fakt;

   scanf("%d", &n);
   fakt = 1;
   for (i = 2; i <= n; i = i + 1)  {
      fakt = fakt * i;
   }

   printf("%d! = %d", n, fakt);
   return 0;
}
```

Naredbe za bezuvjetne programske skokove

- Naredbe koje kontrolu toka (daljnje izvršavanje programa) bezuvjetno prenose na neko određeno mjesto u programu
  - **break;**
    - trenutačni prekid naredbe switch (već viđeno) ili petlje i nastavljanje izvršavanja prve sljedeće naredbe iza naredbe switch ili petlje
  - **continue;**
    - trenutačni prekid tekuće iteracije petlje i nastavljanje izvršavanja programa sljedećim korakom petlje
  - **goto** *labela*;
    - nastavljanje izvršavanja programa naredbom koja je označena labelom (*labeled statement*) koja je zadana naredbom goto
  - **return;**
    - naredba za povratak kontrole toka i rezultata u pozivajuću funkciju
    - detaljno će se razmatrati tek u poglavljima o funkcijama

Naredba break u petlji

- prekida izvršavanje petlje najniže razine u čijem je tijelu navedena i usmjerava daljnje izvršavanje programa na prvu naredbu koja se nalazi iza petlje koja je prekinuta

```
...
for (...)  {
    ...;
    while (...) {
        ...;
        if (...) {
            break;
        }
        do {
            ...
        } while (...);
    }
    ...
 }
 ...
```

petlja najniže razine u kojoj je navedena naredba break

Primjer

- Programski zadatak
  - Učitati dva prirodna broja *a* i *b* te ispisati njihov zbroj. Učitavanje para brojeva i ispis njihova zbroja ponoviti točno tri puta. Međutim, ako se za *a* ili *b* učita broj koji nije prirodan, prekinuti sva daljnja učitavanja. Program završiti porukom Kraj.

```
1. zbrajanje
 Upisite prvi prirodni broj > 1↵
 Upisite drugi prirodni broj > 2↵
1 + 2 = 3
2. zbrajanje
 Upisite prvi prirodni broj > 3↵
 Upisite drugi prirodni broj > 8↵
3 + 8 = 11
3. zbrajanje
 Upisite prvi prirodni broj > 9↵
 Upisite drugi prirodni broj > 6↵
9 + 6 = 15
Kraj.
```

```
1. zbrajanje
 Upisite prvi prirodni broj > 1↵
 Upisite drugi prirodni broj > 2↵
1 + 2 = 3
2. zbrajanje
 Upisite prvi prirodni broj > 0↵
Kraj.
```

```
1. zbrajanje
 Upisite prvi prirodni broj > 1↵
 Upisite drugi prirodni broj > 0↵
Kraj.
```

Primjer (rješenje s break)

```
#include <stdio.h>

int main(void) {
   int i, a, b;

   for (i = 1; i <= 3; i = i + 1) {
      printf("%d. zbrajanje\n", i);
      printf(" Upisite prvi prirodni broj > ");
      scanf("%d", &a);

      if (a <= 0) break;
      printf(" Upisite drugi prirodni broj > ");
      scanf("%d", &b);

      if (b <= 0) break;
      printf("%d + %d = %d\n", a, b, a + b);
   }
   printf("Kraj.\n");

   return 0;
}
```

Primjer (rješenje bez break)

```
#include <stdio.h>

int main(void) {
   int a, b, i = 0;
   do {
      i = i + 1;
      printf("%d. zbrajanje\n", i);
      printf(" Upisite prvi prirodni broj > ");
      scanf("%d", &a);
      if (a > 0) {
         printf(" Upisite drugi prirodni broj > ");
         scanf("%d", &b);
         if (b > 0) {
            printf("%d + %d = %d\n", a, b, a + b);
         }
      }
   } while (i < 3 && a > 0 && b > 0);
   printf("Kraj.\n");

   return 0;
}
```

Naredba continue

- naredba se (za razliku od naredbe break) koristi samo u petljama
- kao i naredba break, odnosi se na petlju najniže razine u čijem je tijelu navedena
  - unutar petlje while ili do-while
    - usmjerava izvršavanje programa na ispitivanje uvjeta. Ako je uvjet zadovoljen, obavlja se sljedeća iteracija, inače se petlja prekida
  - unutar petlje for
    - usmjerava izvršavanje programa na promjenu vrijednosti kontrolne varijable ("izraz_3") i potom na ispitivanje uvjeta ("izraz_2"). Ako je uvjet zadovoljen, obavlja se sljedeća iteracija, inače se petlja prekida

Primjer

- Programski zadatak
  - Učitati dva prirodna broja *a* i *b* te ispisati njihov zbroj. Učitavanje para brojeva i ispis njihova zbroja ponoviti točno tri puta. Međutim, ako se tijekom učitavanja para učita broj koji nije prirodan, prekinuti učitavanje dotičnog para te nastaviti s učitavanjem sljedećeg para. Program završiti porukom Kraj.

```
1. zbrajanje
 Upisite prvi prirodni broj > 1↵
 Upisite drugi prirodni broj > 2↵
1 + 2 = 3
2. zbrajanje
 Upisite prvi prirodni broj > 0↵
3. zbrajanje
 Upisite prvi prirodni broj > 9↵
 Upisite drugi prirodni broj > 6↵
9 + 6 = 15
Kraj.
```

```
1. zbrajanje
 Upisite prvi prirodni broj > 1↵
 Upisite drugi prirodni broj > 0↵
2. zbrajanje
 Upisite prvi prirodni broj > 0↵
3. zbrajanje
 Upisite prvi prirodni broj > 1↵
 Upisite drugi prirodni broj > 2↵
1 + 2 = 3
Kraj.
```

Primjer (rješenje s continue)

```
#include <stdio.h>

int main(void) {
   int i, a, b;
   for (i = 1; i <= 3; i = i + 1) {
      printf("%d. zbrajanje\n", i);
      printf(" Upisite prvi prirodni broj > ");
      scanf("%d", &a);
      if (a <= 0) continue;
      printf(" Upisite drugi prirodni broj > ");
      scanf("%d", &b);
      if (b <= 0) continue;
      printf("%d + %d = %d\n", a, b, a + b);
   }
   printf("Kraj.\n");

   return 0;
}
```

Primjer (rješenje bez continue)

```
#include <stdio.h>

int main(void) {
   int i, a, b;
   for (i = 1; i <= 3; i = i + 1) {
      printf("%d. zbrajanje\n", i);
      printf(" Upisite prvi prirodni broj > ");
      scanf("%d", &a);
      if (a > 0) {
         printf(" Upisite drugi prirodni broj > ");
         scanf("%d", &b);
         if (b > 0) {
            printf("%d + %d = %d\n", a, b, a + b);
         }
      }
   }
   printf("Kraj.\n");

   return 0;
}
```

Primjer: nepotrebno korištenje break i continue

- Programski zadatak
  - napisati program koji će učitavati cijele brojeve s tipkovnice i postupati prema sljedećem pravilu
    - ako je učitani broj manji od nule, ispisati poruku "Nedopustena vrijednost" i prestati s učitavanjem brojeva
    - ako je učitani broj jednak nuli, ispisati učitani broj i prestati s učitavanjem brojeva
    - ako je učitani broj veći od 100, treba ga zanemariti, ispisati poruku "Zanemarujem vrijednost" i nastaviti s učitavanjem
    - inače (ako niti jedan od prethodnih uvjeta nije zadovoljen), ispisati učitani broj i nastaviti s učitavanjem

Rješenje (loše) s break i continue

```
#include <stdio.h>
int main(void) {
   int broj;
   do {
      printf("Upisite broj > ");
      scanf("%d", &broj);
      if (broj < 0) {
         printf("Nedopustena vrijednost\n");
         break;
      }
      if (broj > 100) {
         printf("Zanemarujem vrijednost\n");
         continue;
      }
      printf("Upisani broj je : %d\n", broj);
   } while (broj != 0);
   return 0;
}
```

Rješenje (dobro) bez break i continue

```
#include <stdio.h>
int main(void) {
   int broj;
   do {
      printf("Upisite broj > ");
      scanf("%d", &broj);
      if (broj > 100)
         printf("Zanemarujem vrijednost\n");
      else if (broj >= 0)
         printf("Upisani broj je: %d\n", broj);
      else
         printf("Nedopustena vrijednost\n");
   } while (broj > 0);
   return 0;
}
```

Naredba goto

C program - sintaksa

```
goto oznaka_naredbe;
...
oznaka_naredbe naredba;
...
```

- `oznaka_naredbe naredba` je označena naredba (*labeled statement*) koja se može nalaziti bilo gdje unutar funkcije
  - slično naredbi označenoj labelom case: ili default: unutar switch
- naredbom `goto oznaka_naredbe;` kontrola toka programa se usmjerava na naredbu označenom labelom `oznaka_naredbe`
  - naredba goto i pripadajuća označena naredba moraju se nalaziti u istoj funkciji
    - istu označenu naredbu može koristiti više naredbi goto
    - označena naredba se može nalaziti prije ili poslije naredbe goto

Naredba goto

- programski kôd u kojem se nepotrebno (a gotovo uvijek je nepotrebno) koristi naredba goto smatra se zapetljanim (pogrdno: špageti-kôd) i stoga teškim za održavanje


```
for (i = 1; i <= 10; i = i + 1) {
    scanf("%d", &a);
    if (a < 0)
       printf("manji je od 0\n");
    else if (a == 0)
       printf("jednak je 0\n");
    else
       printf("veci je od 0\n");
}
```

U rješenjima zadataka na ovom predmetu **nije dopušteno** koristiti naredbu goto. Svako rješenje koje će sadržavati naredbu goto smatrat će se u cijelosti neispravnim.

```
i = 1;
ponovi:
   if (i > 10) goto kraj;
   i = i + 1;
   scanf("%d", &a);
   if (a < 0) goto ispisManji;
   if (a == 0) goto ispisJednak;
   printf("veci je od 0\n");
   goto ponovi;
   ispisManji:
      printf("manji je od 0\n");
      goto ponovi;
   ispisJednak:
      printf("jednak je 0\n");
      goto ponovi;
kraj:;
```

Naredba goto

- rijetki slučaj opravdanog korištenja naredbe goto
  - npr. iz nekog razloga (*uvjet4*) treba prekinuti duboko ugniježđenu petlju i nastaviti s izvršavanjem naredbi nakon vanjske petlje

```
for (...; uvjet1; ...) {
   while (uvjet2) {
      do {
         ...
         if (uvjet4) prekinuti sve tri petlje;
         ...
      } while (uvjet3);
      nastavak3;
   }
   nastavak2;
}
nastavak1;
```

Rješenje s naredbom goto

```
for (...; uvjet1; ...) {
   while (uvjet2) {
      do {
         ...
         if (uvjet4) goto van;
         ...
      } while (uvjet3);
      nastavak3;
   }
   nastavak2;
}
van:
nastavak1;
```

Rješenje bez naredbe goto

- može se riješiti i bez naredbe goto, ali je rješenje manje elegantno i manje učinkovito zbog uvođenja pomoćnih varijabli i dodatnih testova

```
int gotovo = 0;
for (...; uvjet1; ...) {
   while (uvjet2) {
      do {
         ...
         if (uvjet4) {
            gotovo = 1;
            break;
         }
         ...
      } while (uvjet3);
      if (gotovo == 1) break;
      nastavak3;
   }
   if (gotovo == 1) break;
   nastavak2;
}
```

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.

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 > 12↵
12 nije prim broj↵
```

```
Upisite prirodni broj > 13↵
13 jest 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;
}
```

Alternativno rješenje s korištenjem break

```
#include <stdio.h>
int main(void) {
   int i, n;
   printf("Upisite prirodni broj > ");
   scanf("%d", &n);
   for (i = 2; i <= n - 1; i = i + 1) {
      if (n % i == 0) {
         break;                    // daljnja ispitivanja nisu potrebna
      }
   }
   if (i < n || n == 1)     // bio je break
      printf("%d nije prim broj\n", n);
   else                     // svi koraci petlje obavljeni bez break
      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;
}
```