Uvod u programiranje

- predavanja -

listopad 2020.

Kontrola toka programa

- 3. dio -

Primjer

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

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

```
i =  1  2  3  4  5  6  7  8  9  10  11  12  13  14  15
```

  - definicija niza:

$a_1 = a_2 = 1$

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

Rješenje

```
#include <stdio.h>

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

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

   if (n >= 1) printf("%d ", 1);
   if (n >= 2) printf("%d ", 1);
   for (i = 3; i <= n; i = i + 1) {
      a_i = a_i_minus1 + a_i_minus2;
      printf("%d ", a_i);
      a_i_minus2 = a_i_minus1;
      a_i_minus1 = a_i;
   }
   return 0;
}
```

```
1 1 2 3 5 8 13 ...
```

Rješenje

```
#include <stdio.h>

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

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

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

Rješenje

```
#include <stdio.h>

int main(void) {
   int n, i, a_i_minus2 = 1, 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 = a_i_minus1 + a_i_minus2;
         a_i_minus2 = a_i_minus1;
         a_i_minus1 = a_i;
      }
      printf("%d ", a_i);
   }
   return 0;
}
```

```
1 1 2 3 5 8 13 ...
```

Rješenje

```
#include <stdio.h>

int main(void) {
   int n, i, a_i_minus2 = 1, 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 = a_i_minus1 + a_i_minus2;
         a_i_minus2 = a_i_minus1;
         a_i_minus1 = a_i;
      }
      if (i > 1) printf(", ");
      printf("%d", a_i);
   }
   return 0;
}
```

```
1, 1, 2, 3, 5, 8, 13, ...
```

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↵
```

Primjer

- Je li ovakvo rješenje prihvatljivo?

```
int stupac;
for (stupac = 1; stupac <= 10; stupac = stupac + 1)
   printf("%4d", 1 * stupac);
printf("\n");

for (stupac = 1; stupac <= 10; stupac = stupac + 1)
   printf("%4d", 2 * stupac);
printf("\n");

for (stupac = 1; stupac <= 10; stupac = stupac + 1)
   printf("%4d", 3 * stupac);
printf("\n");

... itd. za 4, 5, 6, 7, 8, i 9. redak

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

Rješenje

Dobro rješenje

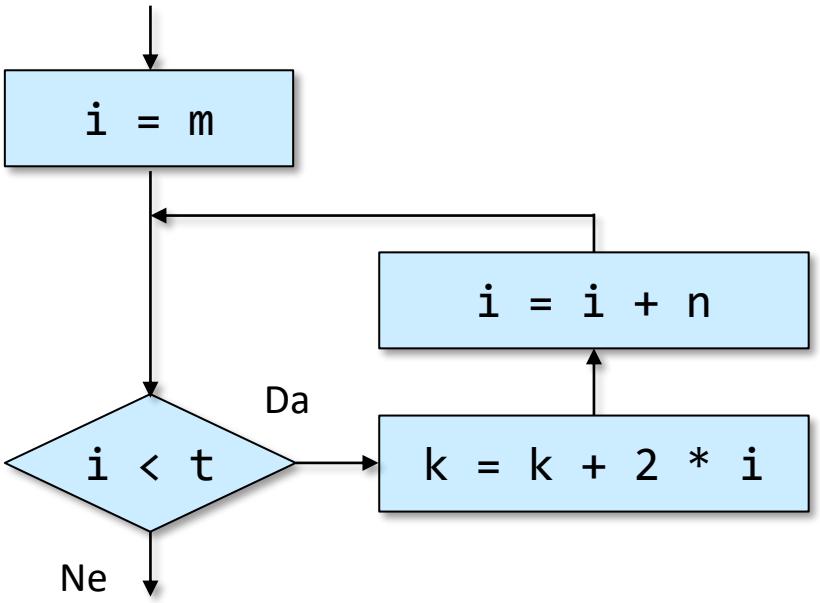
```
#include <stdio.h>

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;
}
```

```
int i;
for (i = 0; i < 100; i = i + 1) {
   printf("%4d", (i / 10 + 1) * (i % 10 + 1));
   if ((i + 1) % 10 == 0) {
      printf("\n");
   }
}
```

Rješenje s jednom petljom je loše, u prvom redu zato jer je teško razumljivo

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

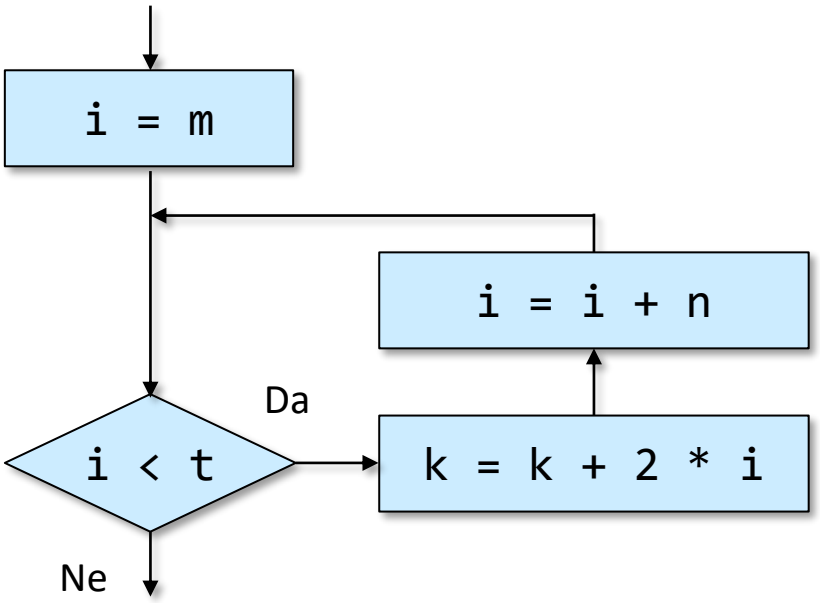
- Programski odsječak prikazan dijagramom toka treba realizirati petljom s ispitivanjem uvjeta ponavljanja na početku




```
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



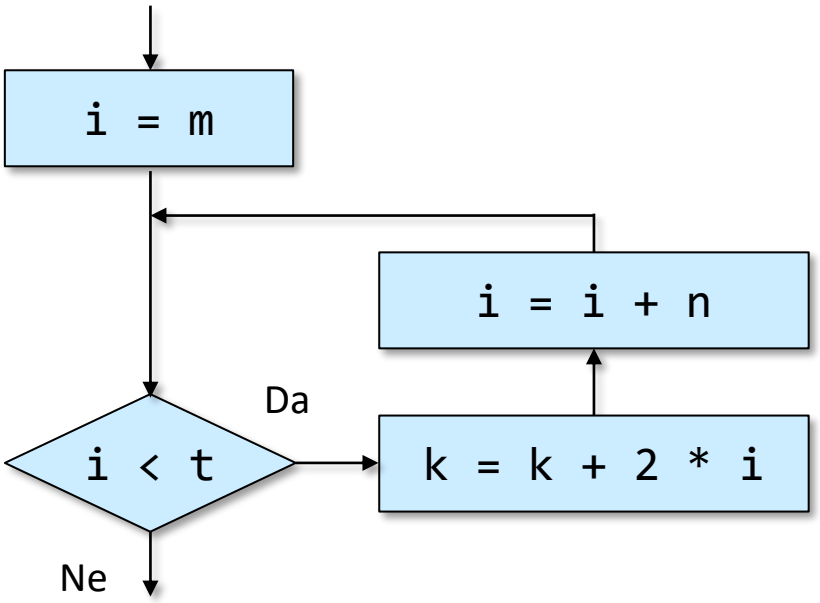

```
i = m;
while (i < t) {
    k = k + 2 * i;
    i = i + n;
}
```

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.

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

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

Neispravno!

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

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

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 poznatim brojem ponavljanja
    - petljom s ispitivanjem uvjeta na kraju
  - 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;
}
```

```
...
if (n >= 2) {
   fakt = 1;
   i = 2;
   while (i <= n) {
      fakt = fakt * i;
      i = i + 1;
   }
} else {
   fakt = 1;
}
...
```

Rješenje (2)

- 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;
}
```

```
i = 2;
while (i <= n) {
   fakt = fakt * i;
   i = i + 1;
}
```

Rješenje (3)

- 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

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 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;
}
```

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;
}
```

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 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 (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: 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 jednak nuli, ispisati učitani broj i prestati s učitavanjem brojeva
    - ako je učitani broj manji od nule, ispisati poruku "Nedopustena vrijednost" 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

- ako je broj == 0, ispisati broj i prestati
- ako je broj < 0, ispisati "Nedopustena vrijednost" i prestati
- ako je broj > 100, ispisati "Zanemarujem vrijednost"i nastaviti
- inače ispisati učitani broj i nastaviti

```
#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

- ako je broj == 0, ispisati broj i prestati
- ako je broj < 0, ispisati "Nedopustena vrijednost" i prestati
- ako je broj > 100, ispisati "Zanemarujem vrijednost"i nastaviti
- inače ispisati broj i nastaviti

```
#include <stdio.h>
int main(void) {
   int broj;
   do {
      printf("Upisite broj > ");
      scanf("%d", &broj);
      if (broj < 0)
         printf("Nedopustena vrijednost\n");
      else if (broj > 100)
         printf("Zanemarujem vrijednost\n");
      else
         printf("Upisani broj je: %d\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);
      ostaleNaredbe3;
   }
   ostaleNaredbe2;
}
ostaleNaredbe1;
```

Rješenje s naredbom goto

```
for (...; uvjet1; ...) {
   while (uvjet2) {
      do {
         ...
         if (uvjet4) goto izaPetlje;
         ...
      } while (uvjet3);
      ostaleNaredbe3;
   }
   ostaleNaredbe2;
}
izaPetlje:
ostaleNredbe1;
```

Rješenje bez naredbe goto

- može se riješiti i bez naredbe goto, ali je rješenje manje elegantno i 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;
      ostaleNaredbe3;
   }
   if (gotovo == 1) break;
   ostaleNaredbe2;
}
ostaleNaredbe1;
```

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

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