Ciklusok használata

Kormányos Andor

Komplex Rendszerek Fizikája Tanszék

2020. szeptember 14.

A második program: Fibonacci-számsorozat

A sorozatot egy **iterációs formulával** adjuk meg

$$\begin{aligned} f_1 &= 1 \\ f_2 &= 1 \\ f_n &= f_{n-2} + f_{n-1} \end{aligned}$$

Adatmodell:

- a sorozatnak mindig csak az előző két elemét kell tárolni $\Rightarrow$ elegendő két `int` típusú változó
- "illetve kell még egy `int` segédváltozó

Iteratív algoritmus:

- egy paraméter, hogy a sorozat hány elemét kell előállítani
- egy iterációs lépés, ami a sorozat előző két eleméből előállítja a következőt
- **egy ciklus**, ami az iterációs lépést ismétli

A második program: Fibonacci-számsorozat

```
1  #include <stdio.h>
2  #include <stdlib.h>
3
4  void fibo(int n) {
5      int a = 0, b = 1;
6      int i = 0;
7      while (i < n) {
8          int c = a + b;
9          a = b;
10         b = c;
11         printf("%d\n", a);
12         i++;
13     }
14 }
15
16 int main(int argc, char* argv[])
17 {
18     // Process command-line arguments
19     if (argc < 2) {
20         printf("Missing argument.\n");
21         exit(-1);
22     }
23     int n = atof(argv[1]);
24     fibo(n);
25     return 0;
26 }
```

- a `main` függvény csak a parancssort dolgozza fel, majd meghívja a `fibo` függvényt

A második program: Fibonacci-számsorozat

```
1  #include <stdio.h>
2  #include <stdlib.h>
3
4  void fibo(int n) {
5      int a = 0, b = 1;
6      int i = 0;
7      while (i < n) {
8          int c = a + b;
9          a = b;
10         b = c;
11         printf("%d\n", a);
12         i++;
13     }
14 }
15
16 int main(int argc, char* argv[])
17 {
18     // Process command-line arguments
19     if (argc < 2) {
20         printf("Missing argument.\n");
21         exit(-1);
22     }
23     int n = atof(argv[1]);
24     fibo(n);
25     return 0;
26 }
```

- a `main` függvény csak a parancssort dolgozza fel, majd meghívja a `fibo` függvényt
- használhatjuk az `atoi` vagy az `atof` függvényt, az eredmény típusát a paraméter deklaráció adja meg

A második program: Fibonacci-számsorozat

```
1  #include <stdio.h>
2  #include <stdlib.h>
3
4  void fibo(int n) {
5      int a = 0, b = 1;
6      int i = 0;
7      while (i < n) {
8          int c = a + b;
9          a = b;
10         b = c;
11         printf("%d\n", a);
12         i++;
13     }
14 }
15
16 int main(int argc, char* argv[])
17 {
18     // Process command-line arguments
19     if (argc < 2) {
20         printf("Missing argument.\n");
21         exit(-1);
22     }
23     int n = atof(argv[1]);
24     fibo(n);
25     return 0;
26 }
```

- a `main` függvény csak a parancssort dolgozza fel, majd meghívja a `fibo` függvényt
- használhatjuk az `atoi` vagy az `atof` függvényt, az eredmény típusát a paraméter deklaráció adja meg
- a függvényben lokális változókat deklarálunk
  - ezek csak a `fibo` függvény számára láthatók
  - `a` és `b` a sorozat két elemét tárolja
  - `i` egy **ciklusváltozó**

A második program: Fibonacci-számsorozat

```
1  #include <stdio.h>
2  #include <stdlib.h>
3
4  void fibo(int n) {
5      int a = 0, b = 1;
6      int i = 0;
7      while (i < n) {
8          int c = a + b;
9          a = b;
10         b = c;
11         printf("%d\n", a);
12         i++;
13     }
14 }
15
16 int main(int argc, char* argv[])
17 {
18     // Process command-line arguments
19     if (argc < 2) {
20         printf("Missing argument.\n");
21         exit(-1);
22     }
23     int n = atof(argv[1]);
24     fibo(n);
25     return 0;
26 }
```

- a `main` függvény csak a parancssort dolgozza fel, majd meghívja a `fibo` függvényt
- használhatjuk az `atoi` vagy az `atof` függvényt, az eredmény típusát a paraméter deklaráció adja meg
- a függvényben lokális változókat deklarálunk
  - ezek csak a `fibo` függvény számára láthatók
  - `a` és `b` a sorozat két elemét tárolja
  - `i` egy **ciklusváltozó**
- a `while(...)` addig ismétli az őt követő utasításokat, amíg a feltétel igaznak értékelődik ki

A második program: Fibonacci-számsorozat

```
1  #include <stdio.h>
2  #include <stdlib.h>
3
4  void fibo(int n) {
5      int a = 0, b = 1;
6      int i = 0;
7      while (i < n) {
8          int c = a + b;
9          a = b;
10         b = c;
11         printf("%d\n", a);
12         i++;
13     }
14 }
15
16 int main(int argc, char* argv[])
17 {
18     // Process command-line arguments
19     if (argc < 2) {
20         printf("Missing argument.\n");
21         exit(-1);
22     }
23     int n = atof(argv[1]);
24     fibo(n);
25     return 0;
26 }
```

- a `main` függvény csak a parancssort dolgozza fel, majd meghívja a `fibo` függvényt
- használhatjuk az `atoi` vagy az `atof` függvényt, az eredmény típusát a paraméter deklaráció adja meg
- a függvényben lokális változókat deklarálunk
  - ezek csak a `fibo` függvény számára láthatók
  - `a` és `b` a sorozat két elemét tárolja
  - `i` egy **ciklusváltozó**
- a `while(...)` addig ismétli az őt követő utasításokat, amíg a feltétel igaznak értékelődik ki
- az iterációs szabályhoz be kell vezetni egy segédváltozót, különben felülírnánk a sorozat valamelyik elemét

A második program: Fibonacci-számsorozat

```
1  #include <stdio.h>
2  #include <stdlib.h>
3
4  void fibo(int n) {
5      int a = 0, b = 1;
6      int i = 0;
7      while (i < n) {
8          int c = a + b;
9          a = b;
10         b = c;
11         printf("%d\n", a);
12         i++;
13     }
14 }
15
16 int main(int argc, char* argv[])
17 {
18     // Process command-line arguments
19     if (argc < 2) {
20         printf("Missing argument.\n");
21         exit(-1);
22     }
23     int n = atof(argv[1]);
24     fibo(n);
25     return 0;
26 }
```

- a `main` függvény csak a parancssort dolgozza fel, majd meghívja a `fibo` függvényt
- használhatjuk az `atoi` vagy az `atof` függvényt, az eredmény típusát a paraméter deklaráció adja meg
- a függvényben lokális változókat deklarálunk
  - ezek csak a `fibo` függvény számára láthatók
  - `a` és `b` a sorozat két elemét tárolja
  - `i` egy **ciklusváltozó**
- a `while(...)` addig ismétli az őt követő utasításokat, amíg a feltétel igaznak értékelődik ki
- az iterációs szabályhoz be kell vezetni egy segédváltozót, különben felülírnánk a sorozat valamelyik elemét
- az `i=i+1` léptetés tömören `i++`

Ciklusok

Ha a programnak többször meg kell ismételnie néhány utasítást, akkor ciklusokat írunk.

A C nyelvben összesen háromféle ciklus van

- `while(i < n) { }` – **elöltesztelős ciklus**, zárójelben az ismétlés feltétele
  - a feltétel egy kifejezés, ugyanazok igazak rá, mint az if esetében
  - a feltétel a ciklusmag lefutása **előtt** értékelődik ki
- `do { } while (i < n);` – **hátul tesztelős ciklus**
  - a feltétel a ciklusmag lefutása **után** értékelődik ki
  - a ciklusmag egyszer mindenképpen lefut!
  - figyelem! a végére kell `;`
- `for (...) { }` – iteratív ciklus

for ciklus

Nagyon gyakori, hogy egy ciklusváltozót lépésenként kell növelni (csökkenteni)

```
1  int i = 0;
2  while (i < 100) {
3    // do something
4    i++;
5  }
```

Az ennek megfelelő **for** ciklus (csak C99 szabvány!)

- két sorral tömörebb
- a ciklusváltozót nem tudjuk véletlenül a cikluson kívül használni

```
1  for (int i = 0; i < 100; i++) {
2    // do something
3  }
```

A `for` ciklus részletes szintaktikája

```
1  for ( [init] ; [condition] ; [increment] ) {
2    // do something
3  }
```

A két zárójelen belüli pontosvessző kötelező

Zárójelen belüli tagok:

1. **init**: utasítás, ami az első iteráció előtt hajtódik végre
   - jellemzően a ciklusváltozó inicializálása
   - de tetszőleges utasítás lehet

A `for` ciklus részletes szintaktikája

```
1  for ( [init] ; [condition] ; [increment] ) {
2    // do something
3  }
```

A két zárójelen belüli pontosvessző kötelező

Zárójelen belüli tagok:

1. **init**: utasítás, ami az első iteráció előtt hajtódik végre
   - jellemzően a ciklusváltozó inicializálása
   - de tetszőleges utasítás lehet
2. **condition**: kifejezés, minden iteráció előtt kiértékelődik
   - ha értéke != 0, akkor a ciklusmag lefut
   - elöltesztelős ciklus, mint a `while`

A `for` ciklus részletes szintaktikája

```
1  for ( [init] ; [condition] ; [increment] ) {
2    // do something
3  }
```

A két zárójelen belüli pontosvessző kötelező

Zárójelen belüli tagok:

1. **init**: utasítás, ami az első iteráció előtt hajtódik végre
   - jellemzően a ciklusváltozó inicializálása
   - de tetszőleges utasítás lehet
2. **condition**: kifejezés, minden iteráció előtt kiértékelődik
   - ha értéke != 0, akkor a ciklusmag lefut
   - elöltesztelős ciklus, mint a `while`
3. **increment**: utasítás, ami a ciklusmag legvégén hajtódik végre
   - tipikusan a ciklusváltozó növelésére, csökkentésére
   - csak akkor fut le, ha a feltétel teljesült, és a ciklusmag is lefutott

Néhány példa for ciklusra

Ciklusok egymásba ágyazhatóak

```
1  for (int i = 0; i < 100; i++) {
2    for (int j = 0; j < 20; j++) {
3      // do something
4    }
5  }
```

```
1  for (int i = 0; i < 100; i++) {
2    for (int j = 0; j < i; j ++) {
3      // do something
4    }
5  }
```

```
1  for (int i = 99; i >= 0; i--) {
2    // do something
3  }
```

Pár tipp

Egy nagyon fontos szabály

- C nyelvben a ciklusokat mindig 0-tól $n-1$-ig futtatjuk
- ez a memóriacímzés miatt van így

Gyakorlófeladat

- Írjuk át a Fibonacci-sorozatot előállító programot `for` ciklusra és hátultesztelős `do – while` ciklusra!

A `break` és a `continue` utasítás

A `break` utasítás kilép a (legbelső) ciklusból

- általában valamilyen feltétel teljesülése esetén
- nem iteráció végén, hanem speciális esetekben használjuk
- `for` ciklusnál is működik, de többnyire `while`-nál

A `break` és a `continue` utasítás

A `break` utasítás kilép a (legbelső) ciklusból

- általában valamilyen feltétel teljesülése esetén
- nem iteráció végén, hanem speciális esetekben használjuk
- `for` ciklusnál is működik, de többnyire `while`-nál

A `continue` utasítás átugorja a ciklusmag hátralevő részét

- ezt is feltétellel együtt használjuk
- a ciklus maga folytatódik, csak a ciklusmag nem fut tovább
- nem az iteráció végén, hanem speciális esetekben használjuk

Figyelem!

- `for` ciklusnál ilyenkor lefut az inkrementáló rész (`i++`)
- `while` ciklusnál ügyelni kell, hogy ne legyen végtelen ciklus

```
1  while (hasnext()) {
2    // do something
3
4    // exit loop if escape key is hit
5    if (isescape()) {
6      break;
7    }
8
9    // do something else
10 }
```

Első komolyabb program: Püthagoraszi számhármasok

Készítsünk programot, ami kilistázza a püthagoraszi számhármasokat, tehát az $a^2 + b^2 = c^2$ feltételnek eleget tevő a, b, c pozitív egész számhármasokat az alábbiak szerint!