Imperatív programozás

Hatókör

Kozsik Tamás és mások

Eötvös Loránd Tudományegyetem

2022. szeptember 1.

ELTE
EÖTVÖS LORÁND
TUDOMÁNYEGYETEM

Tartalomjegyzék

Programszerkezet

- Program tagolása – logikai/fizikai
- Programegységek (program units)

Mellérendelt szerkezetek

- Fordítási egységek
- Programkönyvtárak
- Újrafelhasználhatóság

Alá-/fölérendelt szerkezetek

- Egymásba ágyazódás
- Hierarchikus elrendezés
- Lokalitás: komplexitás csökkentése

Hierarchikus programfelépítés

- Programegységek egymásba ágyazása
- Ha függvényben függvény: blokkszerkezetes nyelv
- Hatókör szűkítése: csak ott használható, ahol használni akarom

Hierarchia nélkül

```
int partition(int array[], int lo, int hi) { ... }

void quicksort_rec (int array[], int lo, int hi) {
  if (lo < hi) {
    int pivot_pos = partition(array,lo,hi);
    quicksort_rec(array, lo, pivot_pos - 1);
    quicksort_rec(array, pivot_pos + 1, hi);
  }
}

void quicksort(int array[], int length) {
  quicksort_rec(array, 0, length - 1);
}
```

Függvények egymásba ágyazása, lokális fv-definíció?

Nem valid C-kód!

```
void quicksort(int array[], int length)
{
  int partition(int array[], int lo, int hi) { ... }

  void quicksort_rec(int array[], int lo, int hi) {
    if (lo < hi) {
      int pivot_pos = partition(array, lo, hi);
      quicksort_rec(array, lo, pivot_pos - 1);
      quicksort_rec(array, pivot_pos + 1, hi);
    }
  }

  quicksort_rec(array,0,length-1);
}
```

Függvények egymásba ágyazása tetszőleges mélységben?

Nem valid C-kód!

```
void quicksort(int array[], int length)
{
  void quicksort_rec(int array[], int lo, int hi)
  {
    int partition(int array[], int lo, int hi) { ... }

    if (lo < hi) {
      int pivot_pos = partition(array, lo, hi);
      quicksort_rec(array, lo, pivot_pos - 1);
      quicksort_rec(array, pivot_pos + 1, hi);
    }
  }

  quicksort_rec(array, 0, length - 1);
}
```

Deklaráció és definíció

Gyakran együtt, de lehet az egyik a másik nélkül!

- Deklaráció: nevet (és típust) adunk valaminek
  - Változódeklaráció
  - Függvénydeklaráció
- Definíció: meghatározzuk, mi az
  - A változó létrehozása (tárhely foglalása)
  - Függvénytörzs megadása

```
unsigned long int factorial(int n);
int main() { printf("%ld\n", factorial(20)); return 0; }
unsigned long int factorial(int n) {
  return n < 2 ? 1 : n * factorial(n - 1);
}
```

Statikus hatóköri szabályok (static/lexical scoping)

Deklaráció hatóköre

A deklaráció hatóköre az a programrész, ahol névvel elérhető az objektum, amire a deklaráció hivatkozik.

- A (statikus) hatóköri szabályok alapja a blokk
  - Alprogram
  - Blokk utasítás
- A deklarációtól a deklarációt közvetlenül tartalmazó blokk végéig

```
int factorial(int n) {
  int result = n, i = result - 1; /* nem cserélhető fel */
  while (i > 1) {
    result *= i;
    --i;
  }
  return result;
}
```

Globális – lokális deklaráció

- Globális: ha a deklarációt nem tartalmazza blokk
- Lokális: ha a deklaráció egy blokkon belül van

- Lokális egy blokkra nézve: abban a blokkban van
- Nonlokális egy blokkra nézve
  - Befoglaló (külső) blokkban van
  - De az aktuális blokk a deklaráció hatókörében van
- Globális: semmilyen blokkra nem lokális

Lokális, non-lokális, globális változó

```
int counter = 0;                                    /* globális */
void fun()
{
  int x = 10;                                     /* lokális fun-ra */
  while (x > 0)
  {
    int y = x / 2;          /* y lokális a blokk utasításra */
    printf("%d\n", 2 * y == x ? y : y + 1);
    --x;                   /* nonlokális változó hivatkozható */
    ++counter;    /* nonlokális (globális) v. hivatkozható */
  }
}
```

Elfedés (shadowing/hiding)

- Ugyanaz a név több dologra deklarálva
- Átfedő (tartalmazó) hatókörrel
- A „belsőbb” deklaráció nyer

```
void hiding()
{
  int n = 0;

  {
    printf("%d", n);    /* 0 */
    int n = 1;
    printf("%d", n);    /* 1 */
  }

  printf("%d",n);       /* 0 */
}
```

Lokális változók deklarációja

ANSI C

- Blokk elején, egyéb utasítások előtt

C99-től

- Keverve a többi utasítással

```
{
  printf("Hello\n");
  int i = 42;
  printf("World\n");
}
```

- for-ciklus lokális változójaként

```
for (int i = 0; i < 10; ++i) printf("%d\n", i);
```

Deklarációk sorrendje

Hibás

```
void g() {
  printf("%c", f());
}
char f() { return 'G'; }
```

Helyes

```
char f();   /* forward deklaráció */
void g() {
  printf("%c", f());
}
char f() { return 'G'; }
```

Definíció deklaráció nélkül

```
double x = x + x
six = double 3
zoo = double 10.0
```

```
six = (\x -> x+x) 3
```

```
double = \x -> x+x
six = double 3
```

Magasabbrendű függvények

Funkcionális programozási paradigma

```
filter predicate (x:xs)
  | predicate x = x : filter predicate xs
  | otherwise   = filter predicate xs
filter _ [] = []

filter even [1..10]
filter (\x -> x > 4) [1..10]
filter ( > 4 ) [1..10]
```

Dinamikus hatóköri szabályok

Bash

```
#!/bin/bash
x=1
function g() {
  echo $x;
  x=2;
}
function f() {
  local x=3;
  g;
}

f
echo $x
```

C

```
#include <stdio.h>
int x = 1;
void g() {
  printf("%d\n", x);
  x = 2;
}
void f() {
  int x = 3;
  g();
}
int main() {
  f();
  printf("%d\n", x);
}
```