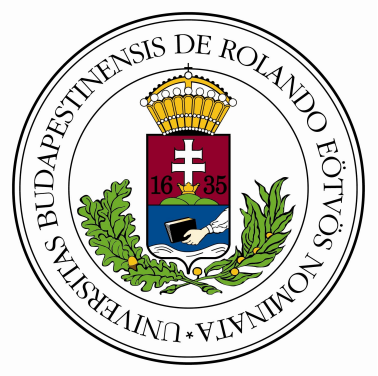
Imperatív programozás

Hatókör



Kozsik Tamás és mások

ELTE Eötvös Loránd Tudományegyetem

Programszerkezet

- Program tagolása – logikai/fizikai
- Programegységek (program units)
  - Pl. alprogramok (függvények)

Mellérendelt szerkezetek

- Fordítási egységek
- Programkönyvtárak
- Újrafelhasználhatóság

Alá-/fölérendelt szerkezetek

- Egymásba ágyazódás
- Hierarchikus elrendezés
- Lokalitás: komplexitás csökkentése

Hierarchikus programfelépítés

- Programegységek egymásba ágyazása
- Python: függvényben függvény
  - Blokkszerkezetes (block structured) nyelv
- Hatókör szűkítése: csak ott használható, ahol használni akarom

Hierarchia nélkül

```
def partition( array, lo, hi ): ...

def quicksort_rec( array, lo, hi ):
    if lo < hi:
        pivot_pos = partition(array,lo,hi)
        quicksort_rec(array,lo,pivot_pos-1)
        quicksort_rec(array,pivot_pos+1,hi)

def quicksort( array ):
    quicksort_rec(array,0,len(array)-1)

time = [2018, 10, 13, 22, 10]
quicksort(time)
quicksort_rec(time, 0, 4)
print(time)
```

Függvények egymásba ágyazása, lokális definíció

```
def quicksort( array ):

    def partition( array, lo, hi ): ...

    def quicksort_rec( array, lo, hi ):
        if lo < hi:
            pivot_pos = partition(array,lo,hi)
            quicksort_rec(array,lo,pivot_pos-1)
            quicksort_rec(array,pivot_pos+1,hi)

    quicksort_rec(array,0,len(array)-1)

time = [2018, 10, 13, 22, 10]
quicksort(time)
quicksort_rec(time, 0, 4)   # hibás!
```

Függvények egymásba ágyazása tetszőleges mélységben

```
def quicksort( array ):

    def quicksort_rec( array, lo, hi ):

        def partition( array, lo, hi ): ...

        if lo < hi:
            pivot_pos = partition(array,lo,hi)
            quicksort_rec(array,lo,pivot_pos-1)
            quicksort_rec(array,pivot_pos+1,hi)

    quicksort_rec(array,0,len(array)-1)
```

A C nem blokk-szerkezetes nyelv

Illegális 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);
}
```

Deklaráció és definíció

Gyakran együtt, de lehet az egyik a másik nélkül!

- Deklaráció: nevet 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);
}
```

Deklaráció hatóköre (scope)

Amíg a névvel elérhető az, amire hivatkozik

- Globális: legkívül van
- Lokális: valamin belül van

```
unsigned long int factorial( int n )   /* globális függvény */
{
    unsigned long int result = 1L;     /*  lokális változó  */
    int i;
    for( i=2; i<n; ++i )
    {
        result *= i;
    }
    return result;
}
```

Globális és lokális függvény

```
def quicksort( array ):  # globális

    def quicksort_rec( array, lo, hi ):   # lokális

        def partition( array, lo, hi ): ...   # lokális

        if lo < hi:
            pivot_pos = partition(array,lo,hi)
            quicksort_rec(array,lo,pivot_pos-1)
            quicksort_rec(array,pivot_pos+1,hi)

    quicksort_rec(array,0,len(array)-1)
```

Blokk (block)

A (statikus) hatóköri szabályok alapja

- Alprogram
- Blokk utasítás (C)

Törzs (body)

Több utasításból álló törzs:

C

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

Python – 1

```python
if lo < hi:
  pivot_pos = partition(array,lo,hi)
  quicksort_rec(array,lo,pivot_pos-1)
  quicksort_rec(array,pivot_pos+1,hi)
```

Python – 2

```python
if lo < hi: lo += 1; hi -= 1
```

Statikus hatóköri szabályok (static/lexical scoping)

Python

a deklarációt közvetlenül tartalmazó blokk

```
def quicksort( array ):
    def quicksort_rec( array, lo, hi ):
        if lo < hi:
            pivot_pos = partition(array,lo,hi)    # hivatkozom
            quicksort_rec(array,lo,pivot_pos-1)
            quicksort_rec(array,pivot_pos+1,hi)
    def partition( array, lo, hi ): ...           # deklarálom
    quicksort_rec(array,0,len(array)-1)
```

C

a deklarációtól a deklarációt közvetlenül tartalmazó blokk végéig

Statikus hatóköri szabályok (static/lexical scoping)

Python

a deklarációt közvetlenül tartalmazó blokk

C

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

Mire nézve lokális?

```
def quicksort( array ):                          # globális

   def quicksort_rec( array, lo, hi ):  # lokális quicksort-ra

      def partition( array, lo, hi ):   # lokális quicksort_rec-re
         ...

      if lo < hi:
         pivot_pos = partition(array,lo,hi)   # lokális fv hívása
         quicksort_rec(array,lo,pivot_pos-1)
         quicksort_rec(array,pivot_pos+1,hi)

   quicksort_rec(array,0,len(array)-1)          # lokális fv hívása

time = [2018, 10, 13, 22, 10]
quicksort(time)                                  # globális fv hívása
```

Non-lokális deklaráció

```
def quicksort( array ):                    # globális

   def partition( array, lo, hi ):         # lokális quicksort-ra
      ...

   def quicksort_rec( array, lo, hi ):     # lokális quicksort-ra
      if lo < hi:
         pivot_pos = partition(array,lo,hi)  # nonlokális fv hívása
         quicksort_rec(array,lo,pivot_pos-1)
         quicksort_rec(array,pivot_pos+1,hi)

   quicksort_rec(array,0,len(array)-1)        # lokális fv hívása

time = [2018, 10, 13, 22, 10]
quicksort(time)                            # globális fv hívása
```

Lokális, non-lokális, globális deklaráció

- 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 C-ben

```c
int counter = 0;                                        /* globális */
int fun(void)
{
    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 */
    }
}
```

Globális deklarációk és definíciók C-ben

```c
int x;

extern int y;

int f(int p);        /* elhagyható az extern */

int g(void)
{
    x = f(y);
}
```

Fordítás

Minden fordítási egységben minden használt név deklarált kell legyen

Szerkesztés

Az egész programban minden globális név pontosan egyszer legyen definiálva

Elfedés (shadowing/hiding)

- Ugyanaz a név több dologra deklarálva
- Átfedő (tartalmazó) hatókörrel

```
void hiding(void)
{
    int n = 0;
    {
        int n = 1;
        printf("%d",n);
    }
    printf("%d",n);
}
```

- A „belsőbb'' deklaráció nyer
- Láthatósági kör: a hatókör része

Lokális változók deklarációja C-ben

ANSI C

- Blokk elején, egyéb utasítások előtt

C99-től

- Keverve a többi utasítással

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

- For-ciklus lokális változójaként

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

Elfedés Pythonban

```
def f(): return 'global'
def g():
    def f(): return 'local'
    print(f())
g()
```

Nonlokális definíció

Hibás Pythonban

```
def f(): return 'global'
def g():
    print(f())
    def f(): return 'local'

g()
```

Helyes C-ben

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

Deklarációk sorrendje

Helyes Pythonban

```
def g():
    print(f())
def f(): return 'global'
g()
```

Hibás C-ben

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

Általános fogalmak

- Deklaráció, definíció
- Blokk
- Hatókör
- Statikus hatóköri szabályok
- Lokális, non-lokális, globális deklarációk
- Elfedés

Nyelvspecifikus észrevételek

C és Python

- Globális változók és függvények

C

- Hatókör: deklarációtól
- Nem blokkszerkezetes
- Forward declaration, extern
- Fordítás/szerkesztés

Python

- Hatókör: blokk elejétől
- Blokkszerkezetes
- Egymásba ágyazott függvények
- Definíció korai elérése?

Érdekes különbségek

Hibás Pythonban

```
def f(): return 'global'
def g():
    print(f())
    def f(): return 'local'

g()
```

Helyes C99-ben

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

Hibás C-ben

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

Helyes Pythonban

```
def g():
    print(f())
def f(): return 'global'
g()
```

Változók Pythonban

- Általában nem kell deklarálni a változókat
- Az első értékadás definiálja és egyben deklarálja is

```
def factorial(n):
    result = 1                  # factorial lokális változója
    for i in range(2,n+1):      # az i is az
        result *= i
    return result
```

Python névfeloldás: dinamikus szemantikai szabályok

Lokális: UnboundLocalError

```
def foo(x):
    if x>0:
        y = 1
    print(y)    # okos dolog ez?


foo(5)  # idáig "értelmes" a program
foo(0)  # ez viszont futási hiba
```

Globális: NameError

```
x = 5
if x>0:
    y = 1
else:
    z = 1

print(y)
print(z)  # bumm!
```

Ciklusváltozó

C99: ciklusra lokális változó

```
for( int i=0; i<10; ++i )
{
    printf("%d",i);
}
printf("%d",i); /* fordítási hiba */
```

C: végtelen ciklus

```
signed char i;
for( i=0; i<=127; ++i )
{
    printf("%c",i);
}
```

C: 012345678910

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

Python

```
for i in range(1,10):
    print(i)

print(i)  # 9
```

Globális változó Pythonban

```
greeting = 'Hi '
def greet(name):
    print(greeting + name)
greet('Tom')
```

Globális változó Pythonban – ez szörnyű!

```
counter = 0
greeting = 'Hi '

def greet(name):
    print(greeting + name)
    counter += 1

greet('Tom')
greet('Jerry')
print(counter)
```

- local variable ‘counter’ referenced before assignment

Globális változó Pythonban – kijavítva

```
counter = 0
greeting = 'Hi '

def greet(name):
    print(greeting + name)
    global counter
    counter += 1

greet('Tom')
greet('Jerry')
print(counter)
```

Globális változó Pythonban – ez is szörnyű!

```
greeting = 'Hi '

def greet(name):
    print(greeting + name)
    global done
    done = True

greet('Tom')                    # ha felcseréljük,
if done: greet('Jerry')         # akkor hibás
```

Non-lokális változó Pythonban

```
def outer():
    n = 1
    def inner():
        print(n)
    inner()

outer()
```

Ismét vigyázni kell!

Helyes (de fura)

```
def outer():
    n = 1
    def inner():
        n = 2
        print(n)
    inner()
    print(n)

outer()
```

Hibás

```
def outer():
    n = 1
    def inner():
        print(n)
        n = 2
    inner()
    print(n)

outer()
```

Megtévesztő lehet

2–1–2

```
def outer():
    n = 1
    def inner():
        global n
        n = 2
        print(n)
    inner()
    print(n)

outer()
print(n)
```

2–2

```
def outer():
    n = 1
    def inner():
        nonlocal n
        n = 2
        print(n)
    inner()
    print(n)

outer()
print(n)  # name 'n' is not defined
```

Függvények átdefiniálása

- Függvény – változó: különböznek?

```
def foo(): return 1
print(foo())
def foo(): return 2
print(foo())
```

```
v = 1
print(v)
v = 2
print(v)
```

Függvények típusa is megváltozhat

```
def foo(): return 1
print(foo())
def foo(x): return x
print(foo(3))
print(foo())    # hibás
```

```
v = 1
print(v)
v = 'Pityu'
print(v + '!')
print(v + 3)    # hibás
```

Dinamikus típusellenőrzés

```
def foo(x):
    if x > 0:
        def bar(v): return v+1
        u = 23
    else:
        def bar(v): return len(v)
        u = 'lala'
    return bar(u)

print( foo(5) )   # 24
print( foo(0) )   # 4
```

Függvények átdefiniálása: global és nonlocal

```
def one(): return 1
def two(): return 2

def foo():
    global two
    def one(): return 'one'
    def two(): return 'two'

print( one(), two() )
foo()
print( one(), two() )
```

Definíció deklaráció nélkül

```
def double(x): return x + x
double(3)
double('tü')
```

```
(lambda x: x+x)(3)
(lambda x: 2*x)('tü')
```

```
double = lambda x: x+x
double(3)
double = lambda x: 2*x
double('tü')
```

Magasabbrendű függvények

Funkcionális programozási paradigma

```
def foreach(fun,list):
    for item in list: fun(item)

foreach( print,                  [1,2,3,4,5] )
foreach( lambda v: print(v,v), [1,2,3,4,5] )
```

Closure

```
def foreach(fun,list):
    for item in list: fun(item)

def multiplicity(pattern,list):
    counter = 0
    def match(item):
        nonlocal counter
        if item == pattern:
            counter += 1
    foreach(match,list)
    return counter

print( multiplicity( 2,  [2,4,5,4,3,2]) )
print( multiplicity('a', 'abracadabra') )
```

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(void)
{
    printf("%d\n",x);
    x = 2;
}
void f(void){
    int x = 3;
    g();
}
void main(){
    f();
    printf("%d\n",x);
}
```

Python

```
#!/usr/bin/python3
x = 1
def g():
    global x;
    print(x);
    x = 2

def f():
    x = 3;
    g()


f()
print(x)
```