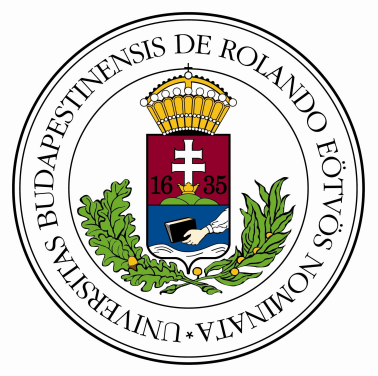
Imperatív programozás

Utasítások



Kozsik Tamás és mások

ELTE Eötvös Loránd Tudományegyetem

Eddig megismert utasítások

- Egyszerű utasítások
  - Változódeklaráció
  - Értékadás
  - Alprogramhívás
  - Visszatérés függvényből
- Vezérlési szerkezetek
  - Elágazás
  - Ciklus

Outline

1. Egyszerű utasítások
2. Vezérlési szerkezetek
   - Elágazások
   - Ciklusok
   - Nem strukturált vezérlésátadás
3. Rekurzió

Változódeklaráció

Python

- Nincs
- Az első értékadás?

C

- Minden változót az első használat előtt létrehozunk
- Érdemes már itt inicializálni

```
double m;
int n = 3;
char cr = '\r', lf = '\n';
int i = 1, j;
int u, v = 3;
```

Kifejezés-utasítás C-ben

(Mellékhatásos) kifejezés kiértékelése

```
<statement> ::= <expression> ;
              | ...
```

```
n = 1;
x *= y;
c++;
n > 0 ? --n : ++n;   /* nem idiomatikus! */
```

Tipikus példa: értékadások

Értékadás Pythonban

- Értékadó utasítások
- Nem kifejezések!

```
n = 2
n += 4

a, b = b, a  # szimultán értékadás

a = b = 1    # többszörös értékadás

a, b = t     # ha t egy pár (mintaillesztés)
```

Kifejezés-utasítás Pythonban

(Mellékhatásos) kifejezés kiértékelése

- Az értékadás nem tartozik ide
- Függvények hívása viszont igen
  - Csak mellékhatás: None visszatérési érték
  - Mellékhatás **és** visszatérési érték: inkább ne!

Tiszta függvény

```
def fact(n):
  result = 1
  for i in range(2,n+1):
     result *= i
  return result
```

Csak mellékhatás

```
def print_squares(n):
  for i in range(1,n+1):
     print(i*i)
  # no return statement
  # returns None
```

Függvények C-ben

- Deklarált visszatérési típus, megfelelő `return` utasítás(ok)
- Csak mellékhatás: `void` visszatérési érték, üres `return`

Tiszta függvény

```
unsigned long fact(int n)
{
  unsigned long result = 1L;
  int i;
  for( i=2; i<=n; ++i )
     result *= i;
  return result;
}
```

Csak mellékhatás

```
void print_squares(int n)
{
  int i;
  for( i=1; i<=n; ++i ){
     printf("%d\n",i*i);
  }
  return; /* elhagyható */
}
```

Kevert viselkedés

```
printf("%d\n", printf("%d\n",42));
```

Visszatérés

- Egy függvényben akár több `return` utasítás is lehet
- Nincs `return` ≡ üres `return` (C-ben: `void`)

C

```c
return 42;
return v + 3.14;


return;
```

Python

```python
return 42
return v + 3.14

return None
return
```

Több return utasítás

C

```c
int index_of_1st_negative( int nums[], int length ){
    int i;
    for( i=0; i<length; ++i )
        if( nums[i] < 0 )
            return i;
    return -1;    /* extremális érték */
}
```

Python

```python
def index_of_1st_negative(nums):
    for i in range(0, len(nums)):
        if nums[i] < 0:
            return i
    return -1     # extremális érték
```

Jobb extremális érték Pythonban

C

```c
int index_of_1st_negative( int nums[], int length ){
    int i;
    for( i=0; i<length; ++i )
        if( nums[i] < 0 )
            return i;
    return -1;    /* nem hagyható el */
}
```

Python

```python
def index_of_1st_negative(nums):
    for i in range(0, len(nums)):
        if nums[i] < 0:
            return i
    return None   # elhagyható, de így kifejezőbb
```

Index vagy érték?

Index

```
def index_of_1st_negative(nums):
    for i in range(0, len(nums)):
        if nums[i] < 0:
            return i
    return None
```

Érték

```
def find_1st_negative(nums):
    for n in nums:
        if n < 0:
            return n
    return None
```

Üres utasítás

Python

```
pass
```

C

```
;
```

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

```
int i, nums[] = {3,6,1,45,-1,4};
for( i=0; i<6 && nums[i]<0; ++i);

for( i=0; i<6 && nums[i]<0; ++i){
}
```

Egyszerű utasítások

- Változódeklarációs utasítás
- Üres utasítás
- Kifejezés-utasítás
- Értékadás
- Alprogramhívás
- Visszatérés alprogramból (`return`)

Outline

1. Egyszerű utasítások
2. Vezérlési szerkezetek
   - Elágazások
   - Ciklusok
   - Nem strukturált vezérlésátadás
3. Rekurzió

Vezérlési szerkezetek

- Elágazás
- Ciklus
  - Tesztelő
    - Elöltesztelő
    - Hátultesztelő
  - Léptető
- Nem strukturált vezérlésátadás
  - `return`, `break`, `continue`, `goto`
  - Kivételek, kivételkezelés

Strukturált programozás

- Szekvencia, elágázás, ciklus
- Minden algoritmus leírható ezekkel
- Olvashatóbb, könnyebb érvelni a helyességéről
- Csak nagyon alapos indokkal térjünk el tőle!

Szekvencia

- Utasítások egymás után írásával
- Pontosvessző: C és Python
- Blokk utasítás C-ben

```
<statement>      ::= { <statement-list> }
                   | ...

<statement-list> ::= ""
                   | <statement> <statement-list>
```

Vezérlési szerkezetek belseje

Python

margó szabály

C

- egy utasítás
- lehet a blokk-utasítás is

Elágazás

- if-else szerkezet
  - az else-ág opcionális
- C-ben: csellengő else

Többágú elágazás

Python

```
if x > 0:
    y = x
elif y > 0:
    x = y
else:
    x = y = x * y
```

C: idióma

```
if( x > 0 )
    y = x;
else if( y > 0 )
    x = y;
else
    x = y = x * y;
```

Többágú elágazás konvencionális tördelése

C: idióma

```
if( x > 0 )
    y = x;
else if( y > 0 )
    x = y;
else
    x = y = x * y;
```

Konvencionális tördelés

```
if( x > 0 )
    y = x;
else
    if( y > 0 )
        x = y;
    else
        x = y = x * y;
```

A kapcsos zárójelek nem ártanak

C: idióma

```
if( x > 0 ){
    y = x;
} else if( y > 0 ){
    x = y;
} else {
    x = y = x * y;
}
```

Konvencionális tördelés

```
if( x > 0 ){
    y = x;
} else {
    if( y > 0 ){
        x = y;
    } else {
        x = y = x * y;
    }
}
```

switch-case-break utasítás C-ben

egész típusú, fordítási idejű konstansok alapján

```
switch( dayOf(date()) )
{
    case  0: strcpy(name,   "Sunday"); break;
    case  1: strcpy(name,   "Monday"); break;
    case  2: strcpy(name,  "Tuesday"); break;
    case  3: strcpy(name,"Wednesday"); break;
    case  4: strcpy(name, "Thursday"); break;
    case  5: strcpy(name,   "Friday"); break;
    case  6: strcpy(name, "Saturday"); break;
    default: strcpy(name,"illegal value");
}
```

Adatban kódolt vezérlés

```
char *names[] = {"Sunday", "Monday", "Tuesday", "Wednesday",
                 "Thursday", "Friday", "Saturday"};
strcpy(name, names[dayOf(date())]);
```

Nem mindig kényelmes adatként

```
switch( key )
{
    case 'i': insertMode(currentRow,currentCol);
              break;
    case 'I': insertMode(currentRow,0);
              break;
    case 'a': insertMode(currentRow,currentCol+1);
              break;
    case 'A': insertMode(currentRow,length(currentRow));
              break;
    case 'o': openNewLine(currentRow+1);
              break;
    case 'O': openNewLine(currentRow);
              break;
}
```

Átcsorgás

```
switch( month )
{
    case  1:
    case  3:
    case  5:
    case  7:
    case  8:
    case 10:
    case 12: days = 31;
             break;
    case  2: days = 28 + (isLeapYear(year) ? 1 : 0);
             break;
    default: days = 30;
}
```

Nem triviális átcsorgás

```
switch( getKey() )
{
    case 'q': jump = 1;
    case 'a': moveLeft();
              break;
    case 'e': jump = 1;
    case 's': moveRight();
              break;
    case ' ': openDoor();
}
```

Duff's device, Pigeon's device stb.

```
switch ( trafficlight ){
    case AMBER:  if( canSafelyStop() ){
    case RED:        stop();
                     break;
                 }
    case GREEN:  go();
}
```

A `switch` és a strukturált programozás

Strukturáltnak tekinthető

- Minden ág végén `break`
- Ugyanaz az utasítássorozat több ághoz

Nem felel meg a strukturált programozásnak

- Nem triviális átcsorgások
- Pl. ha egyáltalán nincs `break`

Elöltesztelő ciklus

Python

```
while i > 0:
    print(i)
    i -= 1
```

C

```
while( i > 0 )
{
    printf("%i\n", i);
    --i;
}
```

Olvashatóság

C

```c
while( i > 0 )
{
    printf("%i\n", i);
    --i;
}
```

C

```c
while( i > 0 )
    printf("%i\n", i--);
```

while – szintaxis

Python

```
<while-stmt> ::= while <expression> :
                     <body>
                 <optional-else>
<optional-else> ::= ""
                  | else : <body>
```

C

```
<while-stmt> ::= while ( <expression> ) <statement>
```

Hátultesztelő ciklus

C-ben

<do-while-stmt> ::= do <statement> while (<expression>);

Jellemző példa

```
char command[LENGTH+1];
do {
   read_data(command);
   if( strcmp(command,"START") == 0 ){
      printf("start\n");
   } else if( strcmp(command,"STOP") == 0 ){
      printf("stop\n");
   }
} while( strcmp(command,"QUIT") != 0 );
```

Átírás – 1

Milyen feltétel mellett igaz ez?

`do` σ `while ( )` ε `;` $\equiv$ σ `while ( )` ε σ

Átírás – 2

Milyen feltétel mellett igaz ez?

```
do σ while ( ε );
```

$\equiv$

```
int new_var = 1; ... while ( new_var ){ σ  new_var = ε; }
```

Az előző példa átírva

```
char command[LENGTH+1];
int new_var = 1;
...
while( new_var ) {
    read_data(command);
    if( strcmp(command,"START") == 0 ){
        ...
    } else if( strcmp(command,"STOP") == 0 ){
        ...
    }
    new_var = ( strcmp(command,"QUIT") != 0 );
}
```

Refaktorálva

```
char command[LENGTH+1];
int stay_in_loop = 1;
...
while( stay_in_loop ) {
    read_data(command);
    if(         strcmp(command,"START") == 0 ){
        ...
    } else if( strcmp(command,"STOP" ) == 0 ){
        ...
    } else if( strcmp(command,"QUIT" ) == 0 ){
        stay_in_loop = 0;
    }
}
```

Vége jelig való beolvasás idiómája

```
void cat(void)
{
    int c;
    while( (c = getchar()) != EOF )
    {
        putchar(c);
    }
}
```

Léptető ciklus

Python

```
<for-stmt> ::= for <target-list> in <expression-list> :
                    <body>
               <optional-else>
<optional-else> ::= ""
                  | else <body>
```

C

```
<for-stmt> ::= for ( <optional-expression> ;
                     <optional-expression> ;
                     <optional-expression> )
                   <statement>
<optional-expression> ::= "" | <expression>
```

(inicializáció; feltétel; léptetés)

Végtelen ciklus C-ben

```
while(1) ...

for(;;) ...
```

Karaktertábla készítése

```
unsigned char c;
for( c = 0; c <= 255; ++c )
{
    printf( "%d\t%c\n", c, c );
}
```

Célszerű így fordítani

```
gcc -ansi -W -Wall -pedantic ...
```

Átírások

Mindig megtehető

`while ( ` ε ` ) ` σ $\quad\Rightarrow\quad$ `for ( ; ` ε `; ) ` σ

Milyen feltétel mellett igaz ez?

`for ( ` ι `; ` ε `; ` λ `) ` σ $\quad\Rightarrow\quad$ ι`; while ( ` ε `){ ` σ λ`; }`

Strukturált programozás vezérlési szerkezetei

- Blokk utasítás
- Elágazások
  - `if–elif–else`
  - `switch–case–break`
- Ciklusok
  - Tesztelő ciklusok
    - Elöltesztelő (`while`)
    - Hátultesztelő (`do–while`)
  - Léptető ciklus (`for`)

Nem strukturált vezérlésátadás

- `return`
- `break` és `continue`
- `goto`
- kivételek, kivételkezelés

break utasítás

- Kilép a legbelső ciklusból (vagy switch-ből)

```
while( !destination(x,y) ){
    drawPosition(x,y);
    dx = read(sensorX);
    if( dx == 0 ){
        dy = read(sensorY);
        if( dy == 0 ) break;
    } else dy = 0;
    x += dx;
    y += dy;
}
```

- Python: a while/for else-ága sem hajtódik végre

continue utasítás

- Befejezi a legbelső ciklusmag végrehajtását

```
while not destination(x,y):
    drawPosition(x,y)
    dx = read(sensorX)
    if dx == 0:
        dy = read(sensorY)
        if dy == 0: continue
    else: dy = 0
    if validPosition( x+dx, y+dy ):
        x += dx; y += dy
```

- for-ciklusnál végrehajtja a léptetést

goto utasítás C-ben

- Egy függvényen belül a megadott címkéjű utasításra ugrik

```
<statement> ::= ...
              | goto <label>
              | <label> : <statement>
<label> ::= <identifier>
```

Keressünk nulla elemet egy mátrixban

goto-val

```
int matrix[SIZE][SIZE];
...
int found = 0;
int i, j;
for( i=0; i<SIZE; ++i ){
  for( j=0; j<SIZE; ++j ){
    if( matrix[i][j] == 0 ){
      found = 1;
      goto end_of_search;
    }
  }
}
/* --i; --j; */
end_of_search:;
```

szabályosan

```
int matrix[SIZE][SIZE];
...
  int found = 0;
  int i=-1, j;
  while( i<SIZE-1 && !found ){
    j = -1;
    while( j<SIZE-1 && !found ){
      if( matrix[i+1][j+1] == 0 ){
        found = 1;
      }
      j++;
    }
    i++;
  }
```

Outline

1. Egyszerű utasítások
2. Vezérlési szerkezetek
   - Elágazások
   - Ciklusok
   - Nem strukturált vezérlésátadás
3. Rekurzió

Rekurzív alprogramok

Python

```python
def factorial(n):
    if n < 2:
        return 1
    else:
        return n * factorial(n-1)
```

C

```c
int factorial( int n ){
    if( n < 2 ){
        return 1;
    } else {
        return n * factorial(n-1);
    }
}
```

Másképpen fogalmazva

Python

```python
def factorial(n):
    return 1 if n < 2 else n * factorial(n-1)
```

C

```c
int factorial( int n )
{
    return n < 2 ? 1 : n * factorial(n-1);
}
```

Számítási lépések ismétlése

Imperatív programozás

- Iteráció (ciklus)
- Hatékony

```
def factorial(n):
    result = 1
    for i in range(2,n+1):
        result *= i
    return result
```

Funkcionális programozás

- Rekurzió
- Érthető

```
def factorial(n):
    if n < 2:
        return 1
    else:
        return n * factorial(n-1)
```

Rekurzió imperatív nyelvben

- A legtöbb nyelvben támogatott
- Ritkán használják a gyakorlatban
  - Hatékonyság
  - Stack overflow

Van, amikor kényelmes

```
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);
}
```

Kiváltása

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

Ciklussal, végrehajtási vermet emulálva

```
def quicksort( array ):
    todo = ((0,len(array)-1),)  # feldolgozandó intervallumok
    while len(todo) > 0:        # amíg van feldolgozandó:
        (lo,hi), *todo = todo     # kivesszük az elsőt
        if lo < hi:               # ha kell vele valamit csinálni
            pivot_pos = partition(array,lo,hi)      # feldolgozzuk
            todo = (lo,pivot_pos-1), (pivot_pos+1,hi), *todo
```

Végrekurzív függvény (tail-recursion)

- Vannak eleve végrekurzív módon megadottak
- De mesterségesen is átírhatók (accumulator)

Kézenfekvő

```
def factorial(n):
    if n < 2: return 1
    else: return n * factorial(n-1)
```

Végrekurzív

```
def factorial(n): return factorial_acc(n,1)

def factorial_acc(n,acc):
    if n < 2: return acc
    else: return factorial_acc(n-1,n*acc)
```

A fordítóprogram optimalizálhatja

Végrekurzív

```
int fact_acc(int n, int acc){
   if (n<2) return acc;
   else return fact_acc(n-1,n*acc);
}
```

Optimalizált

```
int fact_acc(int n, int acc){
   START: if (n<2) return acc;
   else {
      acc *= n;
      n--;
      goto START;
   }
}
```

Strukturáltan

```
int fact_acc(int n, int acc){
   while( n>=2 ){

      acc *= n;
      n--;
   }
   return acc;
}
```