

# Alprogramok, paraméterátadás






Pataki Norbert



ELTE Informatikai Kar,  
Programozási Nyelvek és  
Fordítóprogramok Tanszék

October 24, 2016

# Programozási nyelvek

 <p>C++</p>	 <p>JavaScript</p>
 <p>Java/C#</p>	 <p>PHP(Without MySQL)</p>
 <p>Ruby</p>	 <p>Pascal</p>
 <p>Perl</p>	 <p>Lisp</p>
 <p>Visual Basic</p>	 <p>Haskell</p>
 <p>Python</p>	 <p>C</p>

# Alprogramok

- Függvények, eljárások
- Metódusok
- Korutinok
- stb.

# Alprogramok

- Alprogram: olyan nyelvi szerkezet, amelynek segítségével új nevet rendelhetünk egy kódrészlethez, hogy azt később, amikor csak akarjuk, egyszerűen végrehajthassuk.
- A kódrészlet végrehajtásának kezdeményezése, azaz az *alprogram meghívása* a kódrészlethez rendelt név (és esetleg paraméterek) megadásával történik.

# Példa

```
int max( int a, int b )  
{  
    if ( a < b )  
        return b;  
    else  
        return a;  
}
```

...

```
int k = 4;  
int l = 15;  
int m = 7;
```

```
std::cout << max( k, max( l, m ) ) << std::endl;
```

# Alprogramok előnyei

- Karbantarthatóság, újrafelhasználhatóság
- Olvashatóság: Azonosító - kifejezi a funkcionalitást
- Felhasznált változók láthatóságának csökkenése
- Spagetti kódok csökkentése
- Fordíthatóság, tesztelhetőség
- Könyvtárak

# Alprogramok: függvények, eljárások

- A függvények: a paramétereiből kiszámolnak valamilyen információt (pl. max, sin)
- Az eljárások: a paramétereket átalakítják, nem visszaadják a megváltoztatott információt; (pl. rendezés)
- A C/C++ nem különbözteti meg a függvényeket és az eljárásokat, minden alprogram „függvény”.
- Ha nem akarunk semmilyen információt visszaadni: `void` visszatérési típust adhatunk meg.
- A visszatérési értéket nem kötelező eltárolni a hívó oldalon.
- Tisztaság, mellékhatás, eredmény, megfontolások

# Példák

```
std::cout << "Hello" << std::endl;
```

```
printf( "Hello\n" );
```

```
int strlen( const char* s )  
{  
    const char* p = s;  
    while( '\0' != *p )  
    {  
        ++p;  
    }  
    return p - s;  
}
```

## ● Max-ot tároló konténer



# Bevezetés

- `void f( int s ); // int s: formális paraméter`
- `f( 5 ); // 5: aktuális paraméter`
- **Megfeleltetés**

# Fajták

- Érték-szerint
- Cím-szerint
- Eredmény, Érték/Eredmény-szerint
- Név-szerint

# Érték-szerinti paraméterátadás

- Az alprogram meghívásakor új lokális változó jön létre, ebbe másolódik bele az aktuális paraméter értéke
- Csak befele közvetít információt
- Költséges lehet
- Jellemző: C programozási nyelv

# Példa

```
int factorial( int n )
{
    if ( 0 == n )
    {
        return 1;
    }
    else
    {
        return n * factorial( n - 1 );
    }
}
```

# Cím-szerinti paraméterátadás

- Az alprogram a hívó által megadott változóval dolgozik
- Nincs másolat
- Információ iránya: kétirányú

# Referencia

- Álnév egy már létező tárterülethez
- Nem változhat meg, hogy minek az álneve
- C++-ban nincs „null referencia”
- Referencia, konstans referencia
- `const int& kahdeksan = 8;`

# Példa

```
void swap( int& a, int& b )  
{  
    int tmp = a;  
    a = b;  
    b = tmp;  
}
```

```
void swap( int a, int b )  
{  
    int tmp = a;  
    a = b;  
    b = tmp;  
}
```

# Példa

```
std::vector<int> read()
{
    std::vector<int> v;
    int i;
    while( std::cin >> i )
    {
        v.push_back( i );
    }
    return v;
}
```



# Példa

```
void read( std::vector<int>& v)
{
    int i;
    while( std::cin >> i )
    {
        v.push_back( i );
    }
}
```

# Példa

```
void f( int i );  
void g( const int& r );  
  
// ...  
int y = 2;  
f( 3 );  
f( y );  
f( y + 4 );  
  
g( 5 );  
g( y );  
g( y * 2 );
```

# C vs. C++

```
void swap( int* a, int * b )  
{  
    int tmp = *a;  
    *a = *b; // a = b;  
    *b = tmp;  
}
```

```
int k = 4;  
int r = 7;  
swap( &r, &k );
```

# Tömbök

```
void reverse( int* p, int n )
{
    for( int i = 0; i < n / 2; ++i )
    {
        swap( &p[i], &p[n - 1 - i] );
    }
}
```

# Eredmény-szerint

- Új lokális változó jön létre
- Az alprogram a lokális változót használja
- Aktuális paraméter egy változó
- Függvényhívás végén visszamásolja a lokális változó értékét az aktuális paraméterbe
- Érték/Eredmény-szerint: függvényhívás elején bemásolja az aktuális paraméter értékét az új lokális változóba

# Példa

- Ada példa:

```
procedure swap( a, b : in out Integer ) is
  tmp : Integer := a;
begin
  a := b;
  b := tmp;
end swap;
```

# Név-szerint paraméterátadás

- Bonyolult
- Literál/konstans kifejezés: érték-szerint
- Skalár változó: cím-szerint
- Kifejezés: kiértékelődik
- Fortran, Perl, Preprocesszor

# Preprocesszor makrók

Nincs paraméterátadás...

```
#define INC(i) ++i
#define MAX(a,b) ((a)<(b))?(b):(a)

#define PRINT(x, n) for( int i = 0; i < n; ++i ) \
    std::cout << (x) << std::endl;

// ...
int i = 4;
PRINT( i, 7 );
```



# Deklarációk, definíciók

- Több fordítási egység
- Egy fordítási egységen belül
- Egy definíció, sok deklaráció, One Definition Rule
- Deklaráció: minimális információ, ami alapján a fordítóprogram a deklarált programegység típusozását kezelni tudja az aktuális fordítási egységben
- Definíció: pontos megadása a programegységnek

# Deklarációk

```
extern int x;
```

```
void f( int, double );
```

```
class Complex;
```

# Definíciók és definíciók

```
int x;  
  
void f( int i, double d )  
{  
    // ...  
}
```

# Definíciók és definíciók

```
class Complex
{
private:
    double re, im;

public:
    double abs();
    // ...
};
```

# Definíciók / definíciók

```
class Foo
{
private:
    // csak deklaráció:
    static int cnt;
};

// ford. egységben, def:
int Foo::cnt = 0;
```

# Példák

- `int* p[5];`
- `int (*q)[5];`
- `int *r(int);`
- `int (*s)(int);`

# Példák

```
void fv( int (*p)[ 6 ] )  
{  
    // ...  
}  
  
int t[ 4 ][ 6 ];  
fv( t );
```

# Azonosítók a lefordított tárgykódban

r.cpp:

```
int f()  
{  
}
```

```
int f( int )  
{  
}
```

```
int f( double )  
{  
}
```

```
nm r.o
```

```
00000000000000000f T _Z1fd  
000000000000000006 T _Z1fi  
000000000000000000 T _Z1fv
```

```
g++ -c r.cpp
```



# Azonosítók a lefordított tárgykódban

r.cpp:

```
int f()  
{  
}
```

```
double f( void* )  
{  
}
```

```
nm r.o
```

```
namespace nspace      000000000000000006 T _Z1fPv  
{  
  000000000000000000 T _Z1fv  
  int f( double )     000000000000000019 T _ZN6nspacelfEd  
  {  
  }  
}
```

```
g++ -c r.cpp
```

# Azonosítók a lefordított tárgykódban

r.cpp:

```
namespace
{
    int f( double )
    {
    }
}
```

```
inline int f()
{
}
```

```
double f( void*,
          char* )
{
}
```

```
nm r.o
```

```
00...0011 T _Z1fPvPc
```

```
00...0000 t _ZN12_GLOBAL__N_11fEd
```

# Azonosítók a lefordított tárgykódban

r.cpp:

```
class Class
{
public:
    int f(void*);
};
```

```
nm r.o
```

```
int Class::f(void*) 00...000 T _ZN5Class1fEPv
{
}
```

```
g++ -c r.cpp
```

# Azonosítók a lefordított tárgykódban

r.cpp:

```
int x;
```

```
namespace A
```

```
{  
    int x;  
}
```

```
class Class
```

```
{  
    static int c;  
};
```

```
int Class::c;
```

```
nm r.o
```

```
000...004 B _ZN1A1xE  
000...008 B _ZN5Class1cE  
000...000 B x
```

```
g++ -c r.cpp
```