

A Programozás Alapjai 2

Objektumorientált szoftverfejlesztés

Dr. Forstner Bertalan

forstner.bertalan@aut.bme.hu

A C++ mint egy jobb C nyelv

Kipróbáljuk:

- Változódeklaráció, mint utasítás
- bool típus
- main függvény variáns

Függvéynév túlterhelése

- Mi azonosít egy függvényt C-ben?
- Mi azonosít egy függvényt C++ -ban?

Függvéynév túlterhelése

- Mi azonosít egy függvényt C++ -ban?
A neve és az argumentumlistája!
(Visszatérési érték nem!)

Függvénynév túlterhelése

- Példa

```
void drawPixel() {  
    printf("Potty\n");  
}  
void drawPixel(int x, int y)  
{  
    printf("Potty@%d,%d\n",x,y);  
}  
void drawPixel(double angle, double distance)  
{  
    printf("Potty %lf degree, %lf distance\n",  
angle, distance);  
}  
int main(int argc, char* argv[]) {  
  
    drawPixel();  
    drawPixel(5, 6);  
    drawPixel(5.1, 6.1);  
  
    ...  
}
```

Hogyan implementálták?

- Name mangling, a paraméterlista megjelenik a függvénynévben linkerszinten.
 - > C: egy aláhúzás a név elé (cfunc... ->_cfunc)
 - > C++: bonyolultabb, fordítófüggő

<code>int cppfunc()</code>	?cppfunc@@YAHXZ X:void (H:int – return value)
<code>double cppfunc(int a, double b)</code>	?cppfunc@@YANH@Z H:int N: double (N:double – return value)

Name mangling

- C és C++ függvények linker szinten másképp néznek ki!
- Hogy tudják hívni egymást, ha a C kódot C compiler fordította?
- Példa

Makrók és inline függvények

- Gyakran nagyon rövid kódrészeket is külön függvénybe teszünk (pl. max):
 - > olvashatóság, átláthatóság
 - > Módosíthatóság
- A függvényhívásnak megvan a maga költsége, lassítja a kódot. Pl.:

```
int max(int a, int b)
{
    return a>b ? a:b;
}
...
max(x, y);
...
```

Mi történik híváskor?

1. visszatérési cím a stack-re
2. paramétereknek hely a stack-en
3. ugrás a címre
4. lokális változóknak hely a stack-en
- 5. törzs végrehajtás**
6. visszatérés érték a stack-re
7. lokális változók „felszabadítása”
8. ugrás vissza
9. paraméterek és visszatérési érték „felszabadítása”

Megj: a foglalás és felszabadítás: SP és BP növelés és csökkentés.

C-s megoldás: Makrók!

- Szövegszerű behelyettesítés

```
#define MAX(a,b) ((a)>(b)?(a):(b))  
...  
int i = MAX(1, 2);  
printf("%d\n",i);
```

- Számos veszély!
- Nincs kontextusa, nem végez hibaellenőrzést

C-s makrók: problémák

- Szövegszerű behelyettesítés
- Nincs kontextusa, nem végez hibaellenőrzést

```
printf("%s\n", MAX("GYULA", "BELA"));
```

- Ha hiba van a makróban, annyiszor jelez a compiler hibát, ahány helyen használtuk

C-s makrók: problémák

- Szövegszerű behelyettesítés
- Nincs kontextusa, nem végez hibaellenőrzést

```
printf("%s\n", MAX("GYULA", "BELA"));
```



- Ha hiba van a makróban, annyiszor jelez a compiler hibát, ahány helyen használtuk

Inline függvények

```
inline int max(int a, int b) { ... }
```

- Írjuk a definíciónál a függvény neve elé az inline kulcsszót.
 - > (amikor deklarálom, nem kell az inline, de azzal is lefordul)
- Bemásolódik a függvény törzse a hívás helyére, emiatt gyorsabb
- A makrókkal szemben lokális környezete van a hívásnak és szintaktikai ellenőrzés is. Teljesen biztonságos.
- Ahányszor használom, annyiszor másolódik be a függvény törzse: nő a kód mérete.
- A példa átírása

Inline függvények

- Akkor van értelme használni, ha:
 - > a függvénytörzs végrehajtási ideje összemérhető a függvényhívás karbantartási műveletek idejével. $t(1..4, 6..9) \sim t(5)$
 - > egy-két soros függvények esetén

Inline függvények

- Az inline csak egy javaslat a fordítónak, ő felül tudja bírálni. Kizáró okok is vannak:
 - > rekurzió: önmagát hívja vagy két függvény hívja kölcsönösen egymást
 - > használom a címét a függvénynek (függvény pointer)
 - > címkét használok benne (goto)
 - > Egyebek
- Tegyük a definíciót (törzset) is a header-be
 - > Különbén: **Linker: unresolved external symbol**
 - > **Inline** függvélynél nem baj, hogy a definíció esetleg többször beinclude-olódik (mindaddig, amíg ugyanaz a törzs)

Modern fordítók és az inline

- A korszerű fordítók jobbak annak felderítésében, hogy megéri-e a hívás helyére a függvénytörzset másolni
- Gyakran teljesen ignorálják a programozó inline kulcsszavát
- Jó eséllyel nem jelent teljesítményromlást, ha elhagyjuk az inline kulcsszót

Szóval manapság leginkább arra használjuk, hogyha headerben akarunk függvénytörzset megadni.
A CPP17 már inline változókat is megenged ugyanezért (pl. konstans (pi) a headerben)

Alapértelmezett argumentumok

- Nézzünk példát!

```
void showWindow(int id, int x, int y, char* title)
{
    printf("Uj ablak (%d) a %d,%d koordinatan:
           '%s'", id,x,y,title);
}
```

Alapértelmezett argumentumok

- Hátulról előrefelé haladva alapértelmezett értéket adhatunk meg
- Híváskor hátulról sorban elhagyhatjuk
 - > Fordító automatikusan lenyomja helyettünk a stacken

```
void showWindow(int id, int x=320, int y=480,  
                const char* title="Hiba")  
{  
    printf("Új ablak (%d) a %d,%d koordinatan:  
          '%s'", id,x,y,title);  
}
```

Konstansok

- C-ben:

```
#define BASE_YEAR_SALARY_MILLION 6
```

- Szövegszerű behelyettesítés.
- Nem szabad ;-t
- Nem típusos (nem adtuk meg, hogy int), ez veszélyes.
- Példa

Konstansok

- C++-ban:

```
const double BASE_YEAR_SALARY_MILLION = 6.0;
```

(C++11: *constexpr* ha fordítási időben rendelkezésre áll az érték)

- Típusos.

- Inicializálni kell

- Mi az értelme? Minél inkább megkötjük a programozó kezét, annál kevésbé fog (vagy fogunk mi) hibázni.

Konstans kifejezések

- A konstans kifejezés lehet, hogy már fordítási időben kiszámítható.

```
const int i{ 5 }; //egyértelmű  
const int ii{ i + 3 }; //ez is, de „utána kell járni”
```

- Van, ahol nem

```
int size;  
scanf_s("%d", &size);  
const int iii{ size };
```

- Van, ahol muszáj lenne tudni (pl. tömb):

```
const int size{ 5 };  
int array[size]; //Baj, ha fordítás időben nem ismert
```

„constant expression”

Vs.

„runtime expression”

Konstans kifejezések

- Ahhoz, hogy a fordítónak ne kelljen utánajárni, hogy egy konstans fordítási időben már előáll:

```
constexpr int size{ 5 };  
constexpr int sizePlusOne{ size + 1 };
```

- A constexpr inicializálása kötelező konstans kifejezéssel.

Konstans pointerek

- Külön törődést és gondolkodást igényel
 - > Ki a konstans? A mutató, vagy amit mutatunk?
- Példa:

Konstans pointererek

- Külön törődést és gondolkodást igényel
- Példa:

```
char szo[] = { 'L', 'a', 'p', 'o', 's', '\0' };

const char* p1 = szo;
/*p1 = 'W'; //hiba!
p1++; //OK, 'a'-ra mutat

char* const p2 = szo;
*p2 = 'W'; // OK
//p2++; //Hiba!

const char* const p3 = szo;
/*p3 = 'W'; // Hiba
//p3++; //Hiba!
```

Konstans paraméterek

- Nagyobb méretű változót referenciaként adjunk át függvénynek, mert gyorsabb.
- Milyen problémákat vet ez fel?
 - > Nézzük meg! Példa
 - Fejlesszünk nyomtató „drivert” a HR rendszerhez

Automatikus konverzió

- Automatikus konverzió const-ról nem const-ra nincs
 - > ekkor nem lenne értelme a const-nak
- fordítva van konverzió

Automatikus konverzió

```
void f1(char* p) {  
    ...  
}  
  
void f2(const char* p) {  
    ...  
}  
  
main() {  
    char t[10];  
    char* p=t;  
    const char* pc=t;  
    f1(p); // nincs konv.  
    f2(p); // aut. konv.  
    f1(pc); // hiba: cannot convert const char* to char*  
    f2(pc); // nincs konv.  
}
```

Összefoglalás

- Függvénynév túlterhelés
- Inline függvények
- Alapértelmezett paraméterek
- Konstansok