OO Programozás

Sablonok

Dr. Hatwagner F. Miklós

Széchenyi István Egyetem, Győr

https://github.com/wajzy/GKxB_INTM085

2023. november 1.

Sablonok (template)

- Különböző típusokat kezelő függvények / osztályok csoportjának egyszerű létrehozása
- *Típusparaméter* (fordítási időben eldől) vs. függvény paraméter (futási időben adják át)
- Kulcsszavak:
  - Sablon → `template`
  - Típusparaméterek → `class` vagy `typename` (*majdnem* egyenértékűek)
- *Sablon példányosítás:* adott típusparaméterrel új változat előállítása → egyszerűbb, mint sok felültöltött függvényt kézzel létrehozni

Feladat:

Készítsünk `max` függvényt, ami két paramétere közül visszaadja a nagyobbat!

max1.cpp

```
3  // Overloaded max functions
4  int max(int a, int b) {
5    return a > b ? a : b;
6  }
7
8  double max(double a, double b) {
9    return a > b ? a : b;
10 }
```

max1.cpp

```
12 int main() {
13   const int x = 1;
14   const int y = 2;
15   std::cout << x << " es " << y << " kozul "
16             << max(x, y) << " a nagyobb.\n";
17
18   const double i = 1.5;
19   const double j = 2.5;
20   std::cout << i << " es " << j << " kozul "
21             << max(i, j) << " a nagyobb.\n";
22 }
```

Kimenet

```
1 es 2 kozul 2 a nagyobb.
1.5 es 2.5 kozul 2.5 a nagyobb.
```

max2.cpp

```
3  template<class T> T max(T a, T b) {
4    return a > b ? a : b;
5  }
6
7  int main() {
8    const int x = 1;
9    const int y = 2;
10   std::cout << x << " es " << y << " kozul "
11             << max(x, y) << " a nagyobb.\n";
12
13   const double i = 1.5;
14   const double j = 2.5;
15   std::cout << i << " es " << j << " kozul "
16             << max(i, j) << " a nagyobb.\n";
```

- Egy sablon több típusparamétert is fogadhat, de akkor ezeket mind használni is kell!
- Csak akkor állít elő kódot a fordító, amikor a függvény konkrét hívásával találkozik → a példányosításig a hibás sablon nem vált ki hibaüzenetet
- Pontosan olyan változatok jönnek létre, amikre szükség is van
- Típusonként pontosan egy változatot hoz létre, akkor is, ha létezik a típusok között implicit konverzió (pl. `int` → `long`)
- Mi történik, ha két *eltérő* típusú paraméterrel hívják a függvényünket?

max2.cpp

```
18    //const int k = 3;
19    //const double l = 4.5;
20    //std::cout << k << " es " << l << " kozul "
21    //          << max(k, l) << " a nagyobb.\n";
22    // error: no matching function for call to
23    // 'max(const int&, const double&)'
24    // note: candidate: template<class T> T max(T, T)
25    // note: template argument deduction/substitution failed:
26    // note: deduced conflicting types for parameter 'T' ('int' and 'double')
27  }
```

Probléma:

A két paraméter típusának mindenképpen egyeznie kell!

Félmegoldás:

Két típusparaméter használata (nem befolyásolható futásidőben, melyikben keletkezzen az eredmény)

max3.cpp

```
3  template<class T1, class T2> T1 max(T1 a, T2 b) {
4    return a > b ? a : b;
5  }
6
7  int main() {
8    const int i = 3;
9    const double d = 4.5;
10   std::cout << i << " es " << d << " kozul "
11             << max(i, d) << " a nagyobb.\n";
12   std::cout << d << " es " << i << " kozul "
13             << max(d, i) << " a nagyobb.\n";
14 }
```

Kimenet

```
3 es 4.5 kozul 4 a nagyobb.
4.5 es 3 kozul 4.5 a nagyobb.
```

Ha létezik függvény a szükséges paraméterekkel, nem jön létre példány a sablonból, és a korábbi hiba orvosolható.

max4.cpp

```
3  template<class T1, class T2> T1 max(T1 a, T2 b) {
4    return a > b ? a : b;
5  }
6
7  double max(int a, double b) {
8    return a > b ? a : b;
9  }
```

Kimenet

```
3 es 4.5 kozul 4.5 a nagyobb.
4.5 es 3 kozul 4.5 a nagyobb.
```

Fontosabb tudnivalók:

- A függvénysablonokhoz hasonlóan *osztálysablonokat* is létrehozhatunk.
- A típusparaméterek mellett *konstansparamétereket* is megadhatunk → bárhova helyettesíthetők, ahol konstans kifejezés állhat.
- Ha egy tagfüggvényt az osztályon kívül definiálunk, meg kell ismételni a `template` kulcsszót a paraméterekkel együtt. A definíciónak kivételesen a fejfájlban a helye, hogy a fordító mindig elérje.
- Osztálysablon példányosítása: a típus után < és > között meg kell adni a konkrét típusokat, konstansokat. Csak ezek együtt azonosítják egyértelműen a sablon alapján előállított osztályt!

Stack.h

```
6  template<class T, int I> class Stack {
7      T* array;
8      const int size;
9      int used;
10   public:
11     Stack() : size(I) {
12       array = new T[I];
13       used = 0;
14     }
15     void push(T data);
```

Stack.h

```
16      T pop() {
17        if(used > 0) {
18          return array[--used];
19        } else {
20          std::cerr << "Oops, the stack is empty.\n";
21          return array[0]; // something must be returned
22        }
23      }
24      bool isEmpty() {
25        return used == 0;
26      }
27  };
```

Stack.h

```
29 // It MUST be in the header!
30 template<class T, int I> void Stack<T, I>::push(T data) {
31   if(used < size) {
32     array[used++] = data;
33   } else {
34     std::cerr << "Stack full :(\n";
35   }
36 }
```

stackMain.cpp

```
10 #include <iostream>
11 #include "Stack.h"
12
13 int main() {
14   Stack<int, 3> si;
15   si.push(1); si.push(2); si.push(3);
16   si.push(4); // stack full
17   while(not si.isEmpty()) {
18     std::cout << si.pop() << std::endl;
19   }
```

stackMain.cpp

```
12    Stack<const char*, 10> scp;
13    scp.push("World!\n"); scp.push("Hello ");
14    std::cout << scp.pop();
15    std::cout << scp.pop();
16    std::cout << scp.pop(); // stack empty
17  }
```