

P00 - Templates



Templates


Templates - concept

Template-urile sunt instrumente ce sunt regăsite în C++. Ele se folosesc de conceptul de a trimite ca și parametru un tip de date.


Astfel putem refolosi o metodă pentru mai multe tipuri de date, folosind același cod. De exemplu dacă avem nevoie de mai multe metode de sort care să aibă tipuri de date diferite, în loc să scriem codul de mai multe ori folosim un template.

Astfel se introduc 2 termeni noi:

- template
- typename (care poate fi schimbat cu class)



Logica de templates se
realizează la compile time



```
1  #include <iostream>
2  using namespace std;
3
4  template <typename X>
5  X exMax(X x, X y)
6  {
7      return (x > y)? x: y;
8  }
9
10 int main()
11 {
12     cout << exMax<int>(10, 5) << endl; //int
13     cout << exMax<double>(2.54, 9.21) << endl; //double
14     cout << exMax<char>('o', 'r') << endl; //char
15
16     return 0;
17 }
```

Exemplu template max function

Exemplu sort

```
1 #include <iostream>
2 using namespace std;
3
4 template <class T>
5 void bubbleSort(T a[], int n) {
6     for (int i = 0; i < n - 1; i++)
7         for (int j = n - 1; i < j; j--)
8             if (a[j] < a[j - 1])
9                 swap(a[j], a[j - 1]);
10 }
11
12 int main() {
13     int a[5] = {10, 50, 30, 40, 20};
14     int n = sizeof(a) / sizeof(a[0]);
15
16     bubbleSort<int>(a, 5);
17
18     cout << " Sorted array : ";
19     for (int i = 0; i < n; i++)
20         cout << a[i] << " ";
21     cout << endl;
22
23     char b[5] = {'b', 'x', 'd', 'm', 'o'};
24     int m = sizeof(a) / sizeof(a[0]);
25
26     bubbleSort<char>(b, 5);
27
28     cout << " Sorted array : ";
29     for (int i = 0; i < m; i++)
30         cout << b[i] << " ";
31     cout << endl;
32
33     return 0;
34 }
```

Class Templates

La fel ca și template-urile care definesc funcții, putem să avem template-uri care definesc clase. Acestea sunt utile în momentul în care o clasă definește structuri de date care sunt independente de tipul de date.

```
1 #include <iostream>
2 using namespace std;
3
4 template <class T>
5 class myPair {
6     T a, b;
7     public:
8     myPair (T first, T second)
9         {a=first; b=second;}
10    T getMax ();
11 };
12
13 template <class T>
14 T myPair<T>::getMax ()
15 {
16     T retval;
17     retval = a>b? a : b;
18     return retval;
19 }
20
21 int main () {
22     myPair <int> myObject1 (100, 75);
23     cout << myObject1.getMax() << endl;
24     myPair <char> myObject2 ('r', 'v');
25     cout << myObject2.getMax() << endl;
26     return 0;
27 }
```

Exemplu class

Template specialization

Având în vedere faptul că template-urile primesc ca și parametrii tipuri de date, putem să realizăm o logică diferită în template în funcție de tipul de date primit.

```
1  #include <iostream>
2  using namespace std;
3
4  template <class T>
5  void example(T x)
6  {
7      cout << "Main template example: "
8          << x << endl;
9  }
10
11 template<>
12 void example(int x)
13 {
14     cout << "Specialized emplate for int type: "
15         << x << endl;
16 }
17
18 int main()
19 {
20     example<char>('a');
21     example<int>(100);
22     example<float>(3.14);
23 }
```

Exemplu template specialization functions

```
1 #include <iostream>
2 using namespace std;
3
4 template <class T>
5 class Example
6 {
7 public:
8     Example()
9     {
10         cout << "Main template" << endl;
11     }
12 };
13
14 template <>
15 class Example <int>
16 {
17 public:
18     Example()
19     {
20         cout << "Specialized template object" << endl;
21     }
22 };
23
24 int main()
25 {
26     Example<int> a;
27     Example<char> b;
28     Example<float> c;
29     return 0;
30 }
```

Exemplu template specialization classes

Template vs Overloading

Atât funcțiile supraîncărcate (overloaded) cât și template-urile sunt un bun exemplu de polimorfism în OOP.

Overloading e folosit când mai multe funcții realizează operații similare.

Template-urile sunt utilizate când mai multe funcții realizează operații identice.



În java se numesc generics





Summary

De Reținut

Template-urile sunt șabloane pentru funcții și clase care primesc ca parametrii tipuri de date.

Există template-uri în care putem realiza o logică diferită în funcție de tipul de date, iar conceptul se numește template specialization.

Template-urile sunt un exemplu de compile-time polimorfism.



Exerciții

Exerciții

1. **4.5p** Realizați un template pentru o funcție de concatenare care să trateze cazurile pentru char și int (exemplu: 1,2 -> 12 ; 'a','b' -> 'ab')
2. **4.5p** Realizați un template pentru o clasă de tip stivă (stack) care să aibă metodele de push și pop și să funcționeze atât pentru int-uri cât și pentru char-uri.