Programowanie Obiektowe

C++ szablony

Dariusz Brzeziński

Politechnika Poznańska, Instytut Informatyki

Powielanie kodu

```
int getMax (int a, int b) {
    return (a > b ? a : b);
}

float getMax (float a, float b) {
    return (a > b ? a : b);
}

long getMax (long a, long b) {
    return (a > b ? a : b);
}

double getMax (double a, double b) {
    return (a > b ? a : b);
}
```

Jak temu zaradzić?

Jak temu zaradzić?

```
#include <iostream>
using namespace std;

template <class T>
T getMax (T a, T b) {
   T result;

   result = (a > b) ? a : b;
   return (result);
}

int main () {
   int i = 5, j = 6, k;
   long l = 10, m = 5, n;

   k = GetMax<int>(i, j);
   n = GetMax<long>(l, m);

   cout << k << endl;
   cout << n << endl;
   return 0;
}
```

Szablony funkcji

```
template <class identifier> function_declaration;
template <typename identifier> function_declaration;
```

Kompilator generuje funkcję wzorcową na podstawie typów **argumentów wejściowych** tej funkcji:

```
int main () {
   int i = 5, j = 6, k;
   long l = 10, m = 5, n;

   k = GetMax(i, j);
   n = GetMax(l, m);

   cout << k << endl;
   cout << n << endl;
   return 0;
}
```

Szablony funkcji (2)

Można również definiować wzorce wykorzystujące więcej niż jeden typ:

```
template <class T, class U>
T GetMin (T a, U b) {
  return (a < b ? a : b);
}

int i,j;
long l;
i = GetMin<int, long> (j, l);
i = GetMin (j, l);
```

- Uwaga: w liście parametrów szablonu każdy typ występuje tylko **raz**
- Uwaga: nie mogą istnieć dwa lub więcej szablony funkcji różniące się między sobą jedynie typem wyznaczanej wartości

Szablony klas

- Nazwa wzorca klasy musi być unikalna w całym programie
- Wzorzec klasy musi być zdefiniowany w zakresie globalnym
- Nie można zagnieżdżać definicji wzorców

```
template <class T>
class nazwa_klasy{
   T zmienna1, zmienna2;
   nazwa_klasy (...);  //konstruktor
   ~nazwa_klasy (...); //destruktor
   T metoda1(...);
   void metoda2(T zmienna3);
   ...
};
```

Przykład

Wzorzec klasy o nazwie tablica umożliwiający przechowywanie tablicy liczb dowolnego typu. Rozmiar tablicy jest podawany w momencie tworzenia obiektu tej klasy. Każda tablica przechowuje również swój zadany rozmiar.

```
template <class typ>
class tablica{
   typ* dane; int rozmiar;

 public:
   tablica(int p_rozmiar) : rozmiar(p_rozmiar){
      dane = new typ [p_rozmiar];
   };
   ~tablica(){
      delete [] dane;
   };
   void wyswietl(){
      for (int i=0; i<rozmiar; i++)
         cout<<"dane["<<i<<"]="<<*(dane+i)<<endl;
   };
};
```

Definiowanie funkcji składowych wzorca klasy

```
template <class typ>
class tablica{
   typ* dane;
   int rozmiar;
  public:
   tablica(int p_rozmiar);
   void wyswietl();
};

template <class typ>
tablica<typ>::tablica(int p_rozmiar){
  rozmiar = p_rozmiar;
  dane = new typ [p_rozmiar];
};
```

```
template <class typ>
void tablica<typ>::wyswietl(){
   for(int i=0; i<rozmiar; i++){
      cout<<"dane["<<i<<"]= ";
      cout <<*(dane+i)<<endl;
   }
};
```

Specjalizacje szablonów

```
template <class T>
class mycontainer {
    T element;
  public:
    mycontainer (T arg) { element = arg; }
    T increase () { return ++element; }
};

template <>
class mycontainer <char> {
    char element;
  public:
    mycontainer (char arg) { element = arg; }
    char uppercase ()
    {
      if ((element >= 'a') && (element <= 'z'))
          element += 'A' - 'a';
      return element;
    }
};
```

Specjalizacje szablonów (2)

```
int main () {
  mycontainer <int> myint (7);
  mycontainer <char> mychar ('j');
  cout << myint.increase() << endl;
  cout << mychar.uppercase() << endl;
  return 0;
}
```

Ogólna składnia:

```
template <typy niespecjalizowane>
class mycontainer <specjalizacje typów/wartości> {
 ...
};
```

„Nietypowe” parametry

```
#include <iostream>
using namespace std;

template <class T, int N>
class mysequence {
    T memblock [N];
  public:
    void setmember (int x, T value);
    T getmember (int x);
};

template <class T, int N>
void mysequence<T,N>::setmember (int x, T value) {
  memblock[x] = value;
}

template <class T, int N>
T mysequence<T,N>::getmember (int x){
  return memblock[x];
}
```

```
int main () {
  mysequence<int, 5> myints;
  mysequence<double, 5> myfloats;
  myints.setmember (0, 100);
  myfloats.setmember (3, 3.1416);
  cout << myints.getmember(0);
  cout << myfloats.getmember(3);
  return 0;
}
```

std::vector

Biblioteka STL zawiera przydatną klasę `vector`, która działa jak samo rozszerzalna tablica. Poniżej przedstawiono podstawowe operacje, które można wykonać na obiektach tej klasy.

Operacja	Opis
`unsigned int size();`	Zwraca liczbę elementów znajdujących się w wektorze
`push_back(type element);`	Dodaje element na koniec wektora
`bool empty();`	Zwraca true jeśli wektor jest pusty
`void clear();`	Usuwa wszystkie elementy z wektora
`type at(int n);`	Zwraca element na n-tej pozycji wektora (sprawdzanie zakresu)
`=`	Zamienia zawartość wektora na zawartość przypisywanego wektora
`==`	Wynik porównania dwóch wektorów element po elemencie
`[]`	Bezpośredni dostęp do dowolnej pozycji wektora. Działanie podobne do tego znanego z tablic. Brak sprawdzania zakresu!

Przykład

```
#include <iostream>
#include <vector>

using namespace std;
int main()
{
    vector <int> example;          //Wektor typu int
    example.push_back(3);          //Dodaje 3 do wektora
    example.push_back(10);         //Dodaje 10 na koniec wektora
    example.push_back(33);         //Dodaje 33 na koniec wektora

    for (int x=0; x<example.size(); x++)
    {
        cout << example[x] << " ";    //Zwraca: 3 10 33
    }

    if(!example.empty())
        example.clear();           //Opróżnia wektorvector

    vector <int> another_vector;  //Tworzy nowy wektor typu int
    another_vector.push_back(10);
    example.push_back(10);

    if(example == another_vector) //Sprawdzanie operatora ==
        example.push_back(20);

    for (int y = 0; y < example.size(); y++)
    {
        cout << example[y] << " ";    //Zwraca 10 20
    }
    return 0;
}
```

Źródła

- http://www.cplusplus.com/doc/tutorial/templates/
- http://www.cprogramming.com/tutorial/stl/vector.html