KAPITAŁ LUDZKI
NARODOWA STRATEGIA SPÓJNOŚCI

UNIA EUROPEJSKA
EUROPEJSKI
FUNDUSZ SPOŁECZNY

Projekt współfinansowany ze środków Unii Europejskiej
w ramach Europejskiego Funduszu Społecznego

Programowanie w C++ - wybrane przykłady szablonów

Opracowanie:

dr hab. Mirosław R. Dudek, prof. UZ

Streszczenie

W tym rozdziale podamy kilka najprostszych przykładów programów korzystających z szablonów (*templates*). Często mamy do czynienia z funkcjami (metodami) które wyspecjalizowane są do działań na zmiennych jakiegoś zadanego typu np. **int** a jednocześnie te same funkcje które różnią się tylko zamianą typu zmiennej, np. zamiast **int** jest użyty typ **double**, użyte są w innej części kodu. Okazuje się że można napisać program w sposób bardziej uniwersalny gdzie zamiast konkretnej zmiennej mamy zmienną uniwersalną, tzw. zmienną szablonową, i programista bezpośrednio w programie rzutuje wykonywanie operacji na jednym z typów zmiennych w zależności od potrzeb. Tego rodzaju działania potrafią istotnie przyspieszyć wykonywanie się programu np. od strony numerycznej. Poniższe przykłady powinny ułatwić dalszą naukę programowania z użyciem szablonów. Przyjrzymy się też krótko standardowym bibliotekom języka C++ które korzystają z szablonów.

Wybrane przykłady

Na wstępie rozważmy programy **szablon1a0.cc**, **szablon1a1.cc**, **szablon1a2.cc**, których jedynym celem jest pokazanie na czym polega zastąpienie konkretnych typów zmiennych swojego rosdzaju abstrakcją typu zmiennej - zmienną szablonową. Programem wykorzystującym ideę zmiennej szablonowej jest rzeci z nich. Najpierw rozważmy pierwszy przykład polegający na znajdowaniu minimum spośród dwóch liczb całkowitych.

```
1  #include <iostream>
2  using namespace std;
3
4  //funkcja znajdujaca minimum pomiedzy wartosciami
5  //calkowitymi zmiennych a i b
6  int minimum(int a, int b)
7  {
8          if(a<=b) return a;
9   return b;
10 }
11
12 int main(){
13
14  int a=3,b=6;
15  cout<<"a="<<a<<"\t b="<<b<<":\t Minimum= "<<minimum(a,b)<<endl;
16 return 0;
17 }
```

Nastepnie popatrzmy na taki sam program ale zamiast zmiennych całkowitych mamy zmienne zmiennoprzecinkowe.

```
1  #include <iostream>
2  using namespace std;
3
4  //minimum pomiedzy dwoma wartosciami zmiennoprzecinkowymi typu double
5  double minimum(double a, double b)
6  {
7          if(a<=b) return a;
8   return b;
9  }
10
11 int main(){
12  double pi=3.1415,sq=1.42;
13  cout<<"pi="<<pi<<"\t sq="<<sq<<":\t Minimum= "<<minimum(pi,sq)<<endl;
14
```

```
15 return 0;
16 }
```

A teraz wprowadzimy zmienne szablonowe i jedną funkcję która w sposób uniwersalny porównuje albo wyłącznie wartości całkowite albo rzeczywiste.

```
1  #include <iostream>
2  using namespace std;
3
4  //przyklad z uzyciem zmiennej szablonowej P
5  //bedzie ona w naszym programie reprezentowac
6  //wartosci calkowite i wartosci typu double.
7
8  //konieczne jest napisanie prefiksa: template <class P>
9  //o symbolu P mo na np. myslec jako o double lub int.
10
11 template <class P> P minimum(P a, P b)
12 {
13         if(a<=b) return a;
14  return b;
15 }
16
17 int main(){
18
19  int a=3,b=6;
20  cout<<"a="<<a<<"\t b="<<b<<":\t Minimum= "<<minimum(a,b)<<endl;
21
22  double pi=3.1415,sq=1.42;
23  cout<<"pi="<<pi<<"\t sq="<<sq<<":\t Minimum= "<<minimum(pi,sq)<<endl;
24
25 return 0;
26 }
```

Tutaj wprowadzona została zmienna szablonowa P która w zależności od kontekstu staje się zmienną całkowitą lub zmienną zmiennoprzecinkową. Konieczne tutaj jest dodanie przed definicją funkcji prefiksa **template <class P>** i teraz typ argumentu nazywa się **P** zamiast int lub double i wartość którą funkcja zwraca w tym szczególnym przykładzie też jest typu **P**. Oczywiście fukcja może zwracać dowolną wartość lub być typu void. Co więcej funkcja może posiadać więcej argumentów i nie wszystkie muszą być typu **P**. Pokazujemy to w kolejnym przykładzie.

```
1  #include <iostream>
2  #include <cstring>
3  #include <cstdlib>
4
5  using namespace std;
6
7                                  //przyklad z szablonem funkcji
8                                  //ktora ma wiecej niz jeden parametr
9                                  //rozny od typu szablonowego P
10
11 template <class P> void Wydruk(P a, char *LancuchZnakow)
12 {
13   int k=strlen(LancuchZnakow);
14   cout<<a<<"\t"<<k<<"\t"<<LancuchZnakow<<endl;
15 }
16
```

```
17 int main(){
18 char Lancuch[6]="1234"; //{'1','2','3','4','\0'};
19 long int q;
20         q=atol(Lancuch);
21         Wydruk(q,Lancuch);
22
23 char Dancuch[7]="1.2345";
24 double d;
25         d=atof(Dancuch);
26         Wydruk(d,Dancuch);
27
28 return 0;
29 }
```

Podobnie, może być więcej zmiennych szablonowych, np. zmienne P, T itd. Jeden warunek - trzeba je zadeklarować jako zmienne szablonowe. Jak to zrobić pokazuje kolejny przykład.

```
1  #include <iostream>
2  using namespace std;
3
4                                  //przyklad z szablonem funkcji
5                                  //ktora ma dwie zmienne szablonowe
6
7  template <class P, class T> void Wydruk(P a, T b)
8  {
9    cout<<"Typ P: "<<a<<"\t typ T:"<<b<<endl;
10 }
11
12 int main(){
13
14
15 double d=2.54;
16 int k=3;
17         Wydruk(d,k);
18
19 return 0;
20 }
```

Niestety, w przypadku gdy zmienne szablonowe reprezentują nie tylko standardowe zmienne numeryczne, np. typu **int** czy **double** ale zmienne typu zdefiniowanego przez użytkownika to np. dodawanie ich lub inne działanie trzeba dopiero zdefiniować przeciążając je. Przykładem może być kolejny program gdzie wczytujemy wartości liczbowe tak że są obiektami z klasy Liczba, którą sami definiujemy. Jeśli działania arytmetyczne lub strumieniowe cout i cin nie byłyby przeciążone to kompilator nie wiedzia lby co zrobić np. z poleceniem **c=a+b**.

```
1  #include <iostream>
2  using namespace std;
3
4                                  //Deklaracja klasy-szablonu
5  template <class P> class Liczba{   //Tutaj P jest parametrem oznaczajacym
6                                  //pewien dowolny typ. Parametrow moze
7                                  //byc wiecej niz jeden np. class P, class T
8                                  //Slowo class jest tutaj obowiazkowe.
9  private:
10    P wartosc;
11
12 public:
```

```
13
14
15      Liczba() {} //konstruktor bezparametrowy
16      Liczba(P zmienna){} //konstruktor z jednym parametrem
17
18  //przeciazony operator rzutowania
19      operator P(void) {return wartosc;}
20
21  //przeciazony operator dodawania
22
23      P operator+ (P zmienna){return wartosc+zmienna;}
24      P operator+ (Liczba& zmienna){return wartosc+zmienna.wartosc;}
25  //przeciazony operator przypisania
26      Liczba& operator= (Liczba& zmienna){wartosc=zmienna.wartosc; return *this;}
27      Liczba& operator= (P zmienna); //przeciazony operator przypisania
28
29      friend ostream& operator<< (ostream& cout, Liczba& zmienna){
30                                  return cout<<zmienna.wartosc;}
31    friend istream& operator>> (istream& cin, Liczba& zmienna){cin>>zmienna.wartosc;}
32
33  //klase trzeba uzupelnic o dodatkowe metody
34  //to jest tylko przyklad z kilkoma metodami
35  };
36
37
38
39  // definicja zmiennej szablonowej dla operatora przypisanaia
40  template <class P> Liczba <P>& Liczba<P>::operator=(P zmienna){
41      wartosc=zmienna;
42      return *this;
43  }
44
45
46  int main(){
47
48
49  Liczba<int> a,b,c;   //Tutaj w miejsce parametru P podstawiamy int
50  Liczba<float> d;     //Tutaj w miejsce P podstawiamy float
51  float f;
52
53  cout<<"\nWpisz dwie liczby calkowite a i b: ";
54  cin>>a>>b; //mamy przeciazony operator strumieniowy wejscia
55
56          c=a+b; //tutaj juz dodajemy obiekty bo
57                 //przeciazony jest operator dodawania i operator przypisania
58
59          cout<<"\nIch suma c= "<<c<<endl;
60          f=(float) c; //zamiana typu na float
61          d=f+0.1;
62          cout<<"\nZwiekszona o wartosc 0.1: "<<d<<endl;
63
64   return 0;
65  }
```

Teraz pokażemy przykład z działaniami na liczbach zespolonych gdzie korzysta się ze zmiennych szablonowych. Mam nadzieję że powoli wyłania się urok stosowania zmiennych szablono-

wych.

```
#include <iostream>
#include <cmath>
#include <complex> //biblioteka liczb zespolonych

using namespace std;


int main(){



        complex<double> z1(5.0,1.0);
        complex<double> z2(3.0,4.0);

        cout<<real(z1)<<endl;
        cout<<imag(z1)<<endl;
        cout<<z1<<endl;

        complex<double> z3;
        z3=z1*z2;
        cout<<z3<<endl;
        z3=z1/z2;
        cout<<z3<<endl;
        z3=z1+z2;
        cout<<z3<<endl;
        cout <<abs(z1)<<"\t"<<sqrt(real(z1)*real(z1)+imag(z1)*imag(z1))<<endl;

        complex<double> I(0,1),E(1,0);

        double a=-8.5,b=3.0,pi=M_PI;

        z1=a*E+b*I;
        cout<<z1<<endl;
        z3=cos(z1+b*I);
        cout<<z3<<endl;
        z3=cos(I*pi);
        cout<<z3<<endl;

return 0;

}
```