Instrukcja laboratoryjna nr 3

Programowanie w języku C 2

(C++ poziom zaawansowany)

Szablony funkcji i klas (templates)

dr inż. Jacek Wilk-Jakubowski

mgr inż. Maciej Lasota

dr inż. Tomasz Kaczmarek

Wstęp

Szablony *(ang. Templates)* w innych obiektowych językach programowania (Java, C#) nazywane też **Typami Generycznymi** *(ang. Generic Types)* lub **Typami Uogólnionymi** to element języka C++ pozwalający na tworzenie kodu niezależnego od typów, algorytmów oraz struktur danych. Szablony są techniką realizacji polimorfizmu na innym poziomie niż za pomocą funkcji wirtualnych i dziedziczenia.

Mechanizm szablonów można rozumieć jako pewną formę makrodefinicji (preprocesora). Szablony ze względu na szerokie rozpowszechnienie się zastosowań biblioteki **STL** *(ang. Standard Template Library)* uważane są za jedną z najważniejszych właściwości języka C++. Dzięki szablonom mamy możliwości **programowania ogólnego** *(ang. generic programming)*. Tworząc szablon funkcji (podobnie jak szablon klasy) tworzymy kod działający na nieopisanych jeszcze typach – funkcja działa na typach ogólnych (które nie istnieją w języku C++), w momencie wywołania funkcji pod te typy ogólne podstawiane są konkretne typy, jak np. **int** czy **char**. Przekazujemy szablonowi typy jako parametry, podczas kompilacji następuje tak zwana **konkretyzacja szablonu** *(ang. template instantiation)*, podczas której kompilator na podstawie typów danych przekazanych wzorcowi generuje docelowy kod do obsługi danego typu.

Przykłady implementacji tej samej funkcji dla różnych typów (bez użycia szablonów):

```
// Wersja 1 int:
void Zamień(int &x, int &y) {
	int temp = x;
	x = y;
	y = temp;
}

// Wersja 2 double:
void Zamień(double &x, double &y) {
	double temp = x;
	x = y;
	y = temp;
}

// Wersja 3 string:
void Zamień(string &x, string &y) {
	string temp = x;
	x = y;
	y = temp;
}
```

Dzięki wykorzystaniu szablonów programista C++ może skupić się bardziej na algorytmach niż na danych jakie są przetwarzane. Użycie szablonów pozwala zredukować ilość nadmiarowego kodu ponieważ jedną funkcjonalność można zaprogramować dla wielu typów danych. Następnie kompilator w konkretnym przypadku precyzuje te typy i odpowiednio dostosowuje np. wywołania funkcji, metod lub tworzenia obiektów na podstawie klas.

Wadą działanie szablonów w języku C++ jest to że przypominają one bardzo zaawansowane makra preprocesora. Powoduje to, że kompilator ma zasadnicze trudności z wygenerowaniem prawidłowych, czytelnych komunikatów diagnostycznych w przypadku błędnego użycia poprawnego szablonu.

Szablony w C++ możemy podzielić na:

- **Szablony funkcji**,
- **Szablony klas**.

Szablony funkcji

Szablon funkcji to mechanizm umożliwiający automatyczną generację funkcji. Jest to schemat, według którego postępuje kompilator. Programista dostarcza jedynie ogólny zarys ciała funkcji bez określania konkretnych argumentów wywołania. Na tej podstawie kompilator jest w stanie wytworzyć dowolną ilość funkcji działających identycznie, a różniących się jedynie typem argumentów funkcji. Szablony zezwalają na definiowanie całych rodzin funkcji, które następnie mogą być używane dla różnych typów argumentów.

Definicja szablonu funkcji wygląda następująco:

```
template <typename T> T MojaFunkcja(T a, T b, ...) {
   //instrukcje
   ...
};

lub

template <class T> T MojaFunkcja(T &a, T &b, ...) {
   //instrukcje
   ...
};
```

Definicja umieszczona bezpośrednio przed definicją funkcji informuje że funkcja będzie korzystała z fikcyjnego typu o nazwie T. Definicja dotyczy tylko pojedynczej funkcji zdefiniowanej bezpośrednio po frazie „template”.

Przykłady implementacji funkcji dla różnych typów z użyciem szablonów:

```
// Listing 1
template <typename T> T Sprawdz(T a, T b) {
   return (a>b)?a:b;
};

// Listing 2
template <typename T> T Minimum(T a, T b, T c) {
   if (a <= b && a <= c) return(a);
   if (b <= a && b <= c) return(b);
   return(c);
};

// Listing 3
template <class T> void Zamień(T &x, T &y) {
   T temp = x;
   x = y;
   y = temp;
};
```

Wyrażenie template<typename T> lub template<class T> oznacza, że mamy do czynienia z szablonem, który posiada jeden parametr formalny nazwany T. Słowo kluczowe typename/class oznacza, że parametr ten jest typem (nazwą typu) lub klasy. Nazwa tego parametru może być następnie wykorzystywana w definicji funkcji w miejscach, gdzie spodziewamy się nazwy typu/klasy. I tak powyższe wyrażenie (Listing 1) definiuje funkcję, która przyjmuje dwa argumenty typu T i zwraca wartość typu T, będącą wartością większego z dwu argumentów. Typ T jest na razie niewyspecyfikowany. W tym sensie szablon definiuje całą rodzinę funkcji. Konkretną funkcję z tej rodziny wybieramy poprzez podstawienie za formalny parametr T konkretnego typu będącego argumentem szablonu. Takie podstawienie nazywamy konkretyzacją szablonu. Argument szablonu umieszczamy w nawiasach <> za nazwą szablonu.

Szablony klas

Na podobnej zasadzie co szablony funkcji można także definiować szablony klas (inne określenia: wzorce klas, klasy ogólne). Szablony klas podobne są do makr, tyle że wykonywane są przez kompilator, a nie przez preprocesor. Cechują je pewne własności, których jednak nie ma preprocesor, np. można tworzyć rekurencyjne wywołania. Podobnie jak w przypadku szablonów funkcji, szablon klasy definiuje nam w rzeczywistości całą rodzinę klas.

Definicja szablonu klasy wygląda następująco:

```
template <class T> class MojaKlasa {
   //definicja klasy
};
```

Definicja umieszczona bezpośrednio przed definicją klasy informuje że klasa będzie korzystała z fikcyjnego typu o nazwie T. Definicja dotyczy tylko pojedynczej klasy zdefiniowanej bezpośrednio po frazie „template”.

Przykłady implementacji i użycia klas z użyciem szablonów:

```
// Listing 4
template <class T> class c_klasa {
public:
     T zmienna;
     c_klasa(T l) { zmienna = l; }

     void wyswietl() { cout << zmienna << endl; }
};

int main() {
     c_klasa<int> calkowita(2);
     calkowita.wyswietl();

     c_klasa<char> znak('b');
     znak.wyswietl();
}

// Listing 5
template <class T> class c_klasa {
public:
     T zmienna;
     c_klasa(T l);
     void wyswietl();
};

template <class T>
c_klasa<T>::c_klasa(T l) : zmienna(l) {}

template <class T>
void c_klasa<T>::wyswietl() { cout << zmienna << endl; }
```

```
int main() {
	c_klasa<int> calkowita(2);
	calkowita.wyswietl();

	c_klasa<char> znak('b');
	znak.wyswietl();
}
```

Zadania do wykonania

1. Napisać program, w którym zostaną zaimplementowane 3 funkcje sortujące (algorytmem bąbelkowym, przez wstawianie oraz quicksort) z wykorzystaniem szablonów funkcji. Należy zaprezentować ich zastosowanie dla różnych rodzajów danych wejściowych. Funkcje powinny przyjmować jako parametr tablice do posortowania oraz liczbę elementów w tablicy.

2. Napisać program, w którym zostanie zaimplementowana klasa operująca na **stosie** z wykorzystaniem szablonów klas. Należy zaprezentować działanie tak stworzonego stosu dla różnych rodzajów danych wejściowych.

3. Napisać program, w którym zostanie zaimplementowana klasa operująca na **liście** z wykorzystaniem szablonów klas. Należy zaprezentować działanie tak stworzonej listy dla różnych rodzajów danych wejściowych.

4. Napisać program, który będzie modyfikacją zadania nr 3 (laboratorium nr 2) implementującego słownik angielsko-polski przy pomocy kontenera map. Należy tak zmodyfikować program aby używane były w nim szablony funkcji/klas. Należy zaprezentować działanie tak stworzonego słownika dla różnych rodzajów danych wejściowych.