Paradygmaty programowania

Paradygmaty programowania

Dr inż. Andrzej Grosser

Częstochowa, 2013

Spis treści

1. Zadanie 1

1.1. Wprowadzenie

Szablon jest klasą lub funkcją, która jest parametryzowana zbiorem typów lub wartości. Najczęściej są używane do implementacji ogólnych koncepcji, niezależnych od zastosowanych typów (poszczególne implementacje różnią się jedynie typem przyjmowanych argumentów, przechowywanych pól składowych – np. implementacje tablicy różniłyby się jedynie typem przechowywanych elementów). Na dzisiejszych zajęciach będziemy się zajmować szablonami klas. Ich podstawowa składnia w C++ wymaga zapisu przed nazwą klasy wymaganych parametrów:

```
1 template<typename T>
2 class Box {
3 public:
4     Box() : wsk(new T()) {}
5     explicit Box(T* w);
6     //...
7 private:
8     T* wsk;
9 };
```

W pokazanym powyżej przykładzie T jest nazwą parametru szablonu (tutaj jednym – możliwe jest oczywiście przygotowanie szablonu z większą liczbą parametrów). W trakcie kompilacji, w zależności od tego jak będzie szablon ukonkretniony, wszystkie wystąpienia T będą zastąpione odpowiednim typem. Na przykład:

1. Box<int> b1; - T będzie zastąpione typem całkowitym int,
2. Box<double> b2; - T będzie zastąpione typem rzeczywistym podwójnej precyzji,
3. Box<Point> b3; - T może być nawet zastąpione typem implementowanym przez użytkownika.

W pokazanym przykładzie macie również podany przykład implementacji metod - konstruktora domyślnego i konstruktora jednoargumentowego. Kod konstrukotra domyślnego jest zapisany wewnątrz definicji klasy - implementacja w tym miejscu praktycznie nie różni się od tego jak byłaby zrealizowana wewnątrz zwykłej klasy (poza użyciem parametru

szablonu). Implementacja na zewnątrz klasy byłaby nieco bardziej złożona, wymagałaby poinformowania kompilatora o użytych parametrach. Np dla konstruktora jednoargumentowego:

```
template<typename T>
Box<T>::Box(T* w) : wsk(new T(*w)) {}
```

Dlatego łatwiej jest implementować metody wewnątrz klasy.

1.2. Wskazówki do zadania

W zadaniu chodzi o przygotowanie typu ogólnego Box, który jest rodzajem pudełka przechowującego wskaźnik. Wszystkie operacje na przechowywanym pod tym wskaźnikiem obiektem są wykonywane za pośrednictwem metod szablonu Box. Ponieważ jest przechowywany w klasie wskaźnik pokazujący na obiekt dynamiczny, więc konieczna jest prawidłowa implementacja odpowiednich metod (jakich?). Domyślna implementacja tych metod dostarczana przez kompilator (płytkie kopiowanie obiektów) nie jest wystarczająca.

Na implementację Box jest narzucone dodatkowe wymaganie, żeby obiekty tej klasy zachowywały się jak wskaźniki, tzn. dostęp do wskazywanych elementów był realizowany za pośrednictwem operatora wyłuskania i operatora dostępu do składowej. Jest więc prostą implementacją czegoś co się nazywa sprytny wskaźnik. Pozwala to na uzyskanie korzyści w porównaniu ze zwykłymi wskaźnikami. Najlepiej sprawę wyjaśni porównanie implementacji z wykorzystaniem wskaźnika i klasy Box. Najpierw popatrzmy na wykorzystanie zwykłego wskaźnika:

```
int* w1 = new int;
//Wyłuskanie
*w1 = 10;
Point* w2 = new Point();
//Operator dostępu do składowej.
w2->x = 4;
//...
delete w1;
delete w2;
```

A teraz na wykorzystanie klasy szablonowej Box:

```
Box<int> b1;
//Wyłuskanie
*b1 = 10;
```

```
4 Boix<Point> b2;
5 //Operator dostępu do składowej.
6 b2->x = 4;
```

Proszę zwrócić uwagę, że nie jest konieczne jawne zwalnianie pamięci w przypadku klasy Box (dlaczego?).

Implementacja operator wyłuskania jest prosta (co powinien zwracać?), natomiast operatora dostępu do składowej jest niestety mniej intuicyjna. Przy przeładowaniu tego operatora uzyskuje on specjalne znaczenie - Zapis `b2->x` oznacza tak naprawdę wywołanie `b2.operator->()->x`. Czy na podstawie tego jesteście w stanie wywnioskować co ten operator powinien zwracać?