Paradygmaty programowania

Programowanie generyczne w C++

Dr inż. Andrzej Grosser

Częstochowa, 2014

Spis treści

1. Zadanie 5

1.1. Wprowadzenie

Algorytmy z STL-a z reguły są szablonami funkcji. Przyjmują iteratory i dostosowane do sytuacji funktory. Na przykład dla pierwszego zadania:

```
template <typename Iter1, typename Iter2, typename Pred>
Iter2 copy_if(Iter1 b1, Iter1 e1, Iter2 b2, Pred pred) {
//...
//Treść funkcji
}
```

W ten sposób można stworzyć ogólną wersję funkcji, która będzie działała niezależnie od typów przekazywanych iteratorów i predykatów. Oznacza to, dla przykładu, że ma działać zarówno z iteratorami dla listy jak i z iteratorami dla wektorów:

```
    int tab[] = {1,2,3,4};
    std::vector<int> vec(tab, tab + 4);
    std::list<int> lst, lst2(tab, tab + 4);
    copy_if(vec.begin(), vec.end(), back_inserter(lst),
            std::bind2nd(std::greater<int>(), 2));
    copy(lst.begin(), lst.end(), std::ostream_iterator<int>(std
       ::cout, " "));
    std::cout << "\n----------\n";
//Zarezerwowano pamięć z nadmiarem, ale będą wyświetlane jedynie
    elementy skopiowane
//Jak to jest możliwe?
    std::vector<int> vec2(4);
    copy(vec2.begin(),
         copy_if(lst2.begin(), lst2.end(), vec2.begin(),
                 std::bind2nd(std::greater<int>(), 2)),
         std::ostream_iterator<int>(std::cout, " "));
```

W pokazanym przykładzie można zauważyć, że przygotowany szablon, będzie działał zarówno przy kopiowaniu z wektora, jak i z listy. Zwróćcie uwagę, że w przeciwieństwie do szablonów funkcji, w większości przypadków nie ma potrzeby jawnego zapisywania typów argumentów szablonów funkcji. Kompilator sam na podstawie przekazywanych parametrów jest w stanie wywnioskować typ jaki ma podstawić do szablonu.

1.2. Wskazówki do zadania

Zadania przestaną być trudne, gdy o iteratorach przekazywanych do szablonu funkcji zaczniecie myśleć jako o swego rodzaju wskaźnikach. Można przesuwać się do przodu, wyłuskiwać wartości znajdujące się pod iteratorem. Funktory można traktować jak wskaźniki do funkcji.

Zadanie 1 jest bardzo proste, pozostaje jedynie kwestia odpowiedniego zapisu pętli i zwiększania iteratorów.

Zadanie 2 wymaga użycia szablonu `pair`. Szablon ten służy do przechowywania dwóch wartości - mogą być one różnych typów. Do jego utworzenia należy użyć albo jego konstruktora albo funkcji `make_pair`.

```
//...
//Tutaj Iter1 i Iter2 są parametrami szablonu
Iter1 iter1;
Iter2 iter2;
return pair<Iter1,Iter2>(iter1, iter2);
//lub
return make_pair(iter1,iter2);
//...
```

Zadanie 3 jest najtrudniejsze z listy (co oczywiście nie znaczy, że jest jakoś bardzo trudne :)). Kłopot w tym zadaniu polega na tym, że iterator listy dostarcza operatorów `++` i `--`, które modyfikują jego stan, nie można sięgnąć tak jak w przypadku wskaźników do określonego elementu (nie jest to iterator o dostępie swobodnym). Innymi słowi w przypadku iteratorów do listy nie zadziała:

```
*(wsk + 5) = 12.0;
```

Trzeba zrobić dwa pomocnicze iteratory.

Kontekst testujący macie pokazany z wykorzystaniem funktorów z biblioteki standardowej i wiązadeł. Na podstawie nazwy można w większości wywnioskować, co dany funktor robi. Wiązadło łączy argument funktora z określoną wartością. Np. `bind2nd` łączy drugi argument funktora z określoną wartością.