Programowanie II

Lista 11

Szablony

Załóżmy, że potrzebujemy funkcji, która z dwóch liczb typu int wybierze i zwróci większą jako rezultat.

```
int wieksza (int a, int b)
{
if (a>b)
      return a;
else
      return b;
};
```

Oczywiście w przypadku, gdy będziemy porównywać liczby tybu np. double, funkcja będzie wyglądać analogicznie:

```
double wieksza (double a, double b)
{
if (a>b)
      return a;
else
      return b;
};
```

Można zatem powiedzieć, iż dla różnych typów nasza funkcja zwracająca wartość większą będzie budowana według schematu (szablonu):

```
jakiś_typ wieksza(jakis_typ a, jakiś_typ b)
{
if (a>b)
      return a;
else
      return b;
};
```

Taki sam szablon może zostać zaprezentowany kompilatorowi:

```
template <class jakis_typ>
jakis_typ wieksza(jakis_typ a,jakis_typ b)
{
if (a>b)
      return a;
else
      return b;
};
```

Taką konstrukcję należy czytać jako: to jest szablon, w którym zastępowanym tekstem będzie nazwa typu „jakis_typ”. Szablon jest mechanizmem do tworzenia

rodziny bardzo podobnych funkcji, które są identyczne w działaniu i różniących się typem argumentów.

Nic nie stoi na przeszkodzie aby szablony budowały funkcje dla dowolnej liczby dowolnych parametrów.

```
template <class jakis_typ, class jakis_typ_2>
jakis_typ wieksza(jakis_typ a,jakis_typ_2 b)
{
if (a>b)
      return a;
else
      return b;
};
```

Szablony klas

Jak wiadomo lenistwo jest matką wszelkich wynalazków, zatem gdy nie chce nam się pisać wielu niemal identycznych definicji klas, możemy skorzystać z szablonów klas.

Musimy jednak pamiętać, iż jedyne czym te klasy będą się różniły to ich nazwa i typ obiektów którymi się zajmują.

```
template <class jakis_typ>
class schowek
{
      jakis_typ zasobnik;
      public:
      void schowaj (jakis_typ chowany)
      {
            zasobnik = chowany;
      };
      jakis_typ oddaj()
      {
            return zasobnik;
      };
};
schowek<int> na_integer;
schowek<float> na_float;
na_integer.schowaj(10);
cout<<na_integer.oddaj();
na_float.schowaj(20.20);
cout<<na_float.oddaj();
```

Zadanie 1 (20 pkt)

Wykorzystując wszelką wiedzę na temat programowania i projektowania obiektowego zaproponuj rozwiązanie, które pozwoli na wykonywanie operacji arytmetycznych na macierzach zawierających elementy dowolnego typu.

Należy zaimplementować dodawanie, odejmowanie, mnożenie macierzy przez macierz i przez liczbę. Proszę przygotować prezentację rozwiązania działającego na macierzach przechowujących liczby całkowite, liczby zespolone i np. obiekty reprezentujące punkty w przestrzeni

3D (x, y, z) . Oczywiście, aby można było np. dodawać macierze liczb zespolonych to trzeba najpierw przeciążyć operator dodawania dla takich właśnie liczb.