Wstęp do programowania

Wykład 11. Szablony i STL

Plan wykładu

1. Preprocesor
2. Szablony
2. Biblioteka STL.

Problem

Chcemy napisać funkcję **`getMax(a, b)`**, która zwraca większą z podanych liczb, np.:

```
int getMax(int a, int b){
    return (a>b)?a:b;
}
```

Problem: nie chcemy jej pisać dla kązdego typu danych (**`int`**, **`long`**, **`float`**, **`double`**, ...) z osobna.

Rozwiązanie: #define

```
#define getMax(a,b) (a>b)?a:b

int main(){
    std::cout << (getMax(10,20)) << std::endl;
    std::cout << (getMax(10.5,20.67)) << std::endl;
    std::cout << (getMax(10.5,20)) << std::endl;
}
```

#define to tzw. dyrektywa preprocesora. Preprocesor to program, na który modyfikuje kod źródłowy zanim zostanie on poddany kompilacji. Dyrektywa:

#define cos cosinnego

Przeszukuje kod źródłowy i „inteligentnie” zmienia **cos** na **cosinnego**. W naszym przypadku **getMax(a,b)** zostanie zastąpione przez **(a>b)?a:b**, przy czym **a** i **b** mogą być dowolnymi ciągami. **#define** często wykorzystywana jest do definicji stałych:

#define N 100

Preprocesor

Inne, popularne dyrektywy:

`include` dołącza kod ze wskazanego pliku.

```
#ifdef symbol
…
#endif
```

dołącza kod zaznaczony kropkami tylko gdy został zdefiniowany symbol (za pomocą **`#define`**). Istnieje także wariant **`#ifndef...#endif`**.

```
#if warunek

…

#endif
```

Dołącza kod zaznaczony kropkami tylko wtedy, gdy spełniony jest warunek. Istnieje także wersja **`#if warunek … #elif warunek … #endif`**. Dyrektywy mogą być zagnieżdżane.

Rozwiązanie: szablony

```
template <class myType>
myType getMax (myType a, myType b) {
    return (a>b?a:b);
}


int main(){
    std::cout << getMax(10, 20) << std::endl;
    std::cout << getMax(10.5, 20.67) << std::endl;
    std::cout << getMax<double>(10.5,20) << std::endl;
}
```

Szablony pozwalają zdefiniować funkcję (metodę, klasę), dla której typ zmiennych nie jest ustalony. Typ jest określany w momencie kompilacji. Kompilator może to zrobić automatycznie, albo możemy podać jakiego szablonu ma użyć.

Rozwiązanie: szablony

```
template <class type1, class type2>
type1 getMax (type1 a, type2 b) {
    return (a>b?a:b);
}


int main(){
    std::cout << getMax(10, 20) << std::endl;
    std::cout << getMax(10.5, 20.67) << std::endl;
    std::cout << getMax(10.5,20) << std::endl;
}
```

Kompilator sam domyśla się typów. W rzeczywistości dla każdego wariantu użytych typów tworzona jest osobna funkcja, a ich wywołanie korzysta z mechanizmu przeciążania (przeładowania) funkcji.

Szablony: przykład

```
template<size_t N> class Vector;

template <size_t N>
class Vector{
private:
    double coordinates[N];

public:
    Vector();//constructor
    Vector(const Vector<N> &vec); //copy constructor
    Vector<N> operator+(const Vector<N> &vec); //addition
    Vector<N> &operator=(const Vector<N> &vec);
    double &operator[](size_t i);
    friend std::ostream& operator<< <>(std::ostream& os, const Vector<N> &v);
};
```

Szablony

```
...
template <size_t N>
Vector<N>::Vector(const Vector<N> &vec){
    for(size_t i=0; i<N; i++)
        this->coordinates[i] = vec.coordinates[i];
}

template <size_t N>
Vector<N> Vector<N>::operator+(const Vector<N> &vec){
    Vector<N> vRes;
    for(size_t i=0; i<N; i++)
        vRes.coordinates[i] = this->coordinates[i] + vec.coordinates[i];
    return vRes;
}

template <size_t N>
Vector<N> &Vector<N>::operator=(const Vector<N> &vec){
    for(size_t i=0; i<N; i++)
        this->coordinates[i] = vec.coordinates[i];
    return *this;
}
```

Szablony

```
template <size_t N>
double &Vector<N>::operator[](size_t i){
    return this->coordinates[i];
}

template<size_t N>
std::ostream& operator<<(std::ostream& os, const Vector<N> &v){
    os << "{";
    for(size_t i=0; i<N-1; i++)
        os << v.coordinates[i] << ", ";
    os << v.coordinates[N-1] << "}";
    return os;
}
```

Szablony

```
int main(){
    Vector<3> a3;
    a3[0] = 1; a3[1] = 2; a3[2] = 5;
    Vector<3> b3 = a3;
    Vector<3> c3 = a3+b3;
    std::cout << c3 << std::endl;

...

    Vector<5> a5;
    a5[0] = 1; a5[1] = 2; a5[2] = 5; a5[3] = 3; a5[4] = 4;
    std::cout << a5 << std::endl;
    Vector<4> a4 = a5; // to się nie skompiluje!
}
```

Vector<4> i **Vector<5>** to różne typy, nie związane ze sobą (np. relacją dziedziczenia).
To że powstały z jednego szablonu nie ma znaczenia.

Standard Template Library

Standard Template Library (STL) to biblioteka C++. Zawiera ona wiele przydatnych szablonów, związanych z popularnymi strukturami danych, oraz algorytmami, które na nich operują. Obecnie biblioteka STL znajduje się w standardzie C++, dlatego można z niej korzystać bez żadnych dodatkowych warunków.

std::vector

```cpp
#include <iostream>
#include <vector>

int main() {

    std::vector<int> vec;

    std::cout << "vector size = " << vec.size() << std::endl;

    for(int i = 0; i < 5; i++) {
        vec.push_back(i);
    }

    std::cout << "vector size = " << vec.size() << std::endl;

    for(int i = 0; i < vec.size(); i++) {
        std::cout << "value of vec [" << i << "] = " <<
                     vec[i] << std::endl;
    }
```

std::vector

```
    for(int i : vec) {
        std::cout << "next value of vec = " <<
        i << std::endl;
    }


    std::vector<int>::iterator v = vec.begin();
    while( v != vec.end()) {
        std::cout << "value of v = " << *v << std::endl;
        v++;
    }
    return 0;
}
```

Inne kolekcje

Przykładem innych kolekcji są **`std::list`** i **`std::set`**.

Proszę napisać podobne programy używając tych kolekcji a następnie porównać ich wydajność we wstawianiu, wyszukiwaniu i usuwaniu elementów.

Szablony

Dziękuję za uwagę