

PRZEDMIOT: Elementy programowania

KLASA: 1i gr. 2

Lekcja

Temat: Szablony (templates): Szablony funkcji i klas, specjalizacje, szablony w STL.

1 Szablony (templates) w C++

Szablony pozwalają pisać **uniwersalny kod**, który działa dla różnych typów danych **bez powielania kodu**. Czyli "napisz raz, a zastosuj dla wielu typów"

2 Szablony funkcji

✓ Składnia ogólna:

```
template <typename T>
T maksimum(T a, T b) {
    return (a > b) ? a : b;
}
```

- **T** - to **zmienna typu**, którą kompilator zastąpi konkretnym typem w momencie użycia funkcji lub klasy. Dzięki temu możesz pisać **jedną funkcję lub klasę**, która działa dla wielu typów danych.
- `template <typename T>` - deklaruje szablon z typem T
- Funkcja `maksimum` działa teraz dla **int**, **double**, **float**, itp.

✓ Przykład użycia:

```
#include <iostream>
using namespace std;
```

```
template <typename T>
T maksimum(T a, T b) {
    return (a > b) ? a : b;
}
```

```
int main() {
    cout << maksimum(5, 10) << endl;    // int
    cout << maksimum(3.14, 2.71) << endl; // double
}
```

3 Szablony klas

Szablony działają też dla **klas** — pozwalają tworzyć klasy działające na różnych typach danych.

```
#include <iostream>
using namespace std;
template <typename T>
class Para {
    T pierwszy, drugi;
public:
    Para(T a, T b) : pierwszy(a), drugi(b) {}
    T suma() { return pierwszy + drugi; }
};

int main() {
    Para<int> p1(3, 7);
    cout << p1.suma() << endl; // 10

    Para<double> p2(2.5, 3.5);
    cout << p2.suma() << endl; // 6.0
}
```

4 Specjalizacje szablonów

Czasem chcemy, aby szablon działał **inaczej dla konkretnego typu**.

```
#include <iostream>
using namespace std;

// Szablon klasy ogólny
template <typename T>
class Para {
    T pierwszy, drugi;
public:
    Para(T a, T b) : pierwszy(a), drugi(b) {}
    T suma() { return pierwszy + drugi; }
    void pokaz() { cout << pierwszy << " " << drugi << endl; }
};

// Specjalizacja szablonu dla typu char
template <>
class Para<char> {
    char pierwszy, drugi;
public:
```

```

Para(char a, char b) : pierwszy(a), drugi(b) {}
void pokaz() {
    cout << "Specjalizacja dla char: " << pierwszy << " " << drugi << endl;
}
};

int main() {
    // Użycie szablonu ogólnego dla int
    Para<int> p1(3, 7);
    cout << "Suma int: " << p1.suma() << endl;
    p1.pokaz();

    // Użycie szablonu ogólnego dla double
    Para<double> p2(2.5, 3.5);
    cout << "Suma double: " << p2.suma() << endl;
    p2.pokaz();

    // Użycie specjalizacji dla char
    Para<char> p3('A', 'B');
    p3.pokaz();

    return 0;
}

```

- To nazywamy **pełną specjalizacją**.
- Możemy też tworzyć **częściowe specjalizacje**, np. dla wskaźników.

5 Szablony w STL (Standard Template Library)

STL to **biblioteka standardowa w C++**, która w dużej mierze **opiera się na szablonach**.

Przykłady:

1. **vector** — dynamiczna tablica

```

#include <vector>
#include <iostream>
using namespace std;

int main() {
    vector<int> v = {1, 2, 3};
    v.push_back(4);

    for (int x : v)

```

```
        cout << x << " ";  
    }
```

2. **map** — mapa klucz-wartość

```
#include <map>  
#include <string>  
#include <iostream>  
using namespace std;  
  
int main() {  
    map<string, int> wiek;  
    wiek["Jan"] = 25;  
    wiek["Anna"] = 30;  
  
    for (auto &[k, v] : wiek)  
        cout << k << " ma " << v << " lat" << endl;  
}
```

3. **sort** w **<algorithm>** — szablon funkcji

```
#include <algorithm>  
#include <vector>  
#include <iostream>  
using namespace std;  
  
int main() {  
    vector<int> v = {3, 1, 4, 2};  
    sort(v.begin(), v.end()); // sort działa dla każdego typu porównywalnego  
    for (int x : v) cout << x << " ";  
}
```

W STL wszystko jest napisane przy użyciu **szablonów**, dlatego możesz używać **vector<int>**, **vector<double>**, **map<string,int>** itd., bez pisania osobnej klasy.