

Szablony funkcji

- wyznaczenie większej wartości z dwóch liczb (max) wymaga napisania oddzielnej funkcji dla każdego możliwego typu jej argumentów

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

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

double maks(double a, double b)
{
    return a>b ? a : b;
}

float maks(float a, float b)
{
    return a>b ? a : b;
}
```

```
int main(void)
{
    int a1 = 3, b1 = 5;
    double a2 = 2.3, b2 = -1.2;
    float a3 = 4.1f, b3 = 1.6f;

    cout << "int: " << maks(a1,b1);
    cout << endl;
    cout << "double: " << maks(a2,b2);
    cout << endl;
    cout << "float: " << maks(a3,b3);
    cout << endl;
}
```

```
int: 5
double: 2.3
float: 4.1
```

Programowanie obiektowe (TZ1E2010)

Politechnika Białostocka - Wydział Elektryczny
Elektronika i telekomunikacja, semestr II
studia niestacjonarne I stopnia
Rok akademicki 2020/2021

Pracownia nr 9 (28.05.2021)

dr inż. Jarosław Forenc

Szablony funkcji - przykład

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

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

int main(void)
{
    int a1 = 3, b1 = 5;
    double a2 = 2.3, b2 = -1.2;
    float a3 = 4.1f, b3 = 1.6f;

    cout << "int: " << maks(a1,b1) << endl;
    cout << "double: " << maks(a2,b2) << endl;
    cout << "float: " << maks(a3,b3) << endl;
}
```

```
int: 5
double: 2.3
float: 4.1
```

Szablony funkcji

- tworząc **szablon funkcji** tworzymy kod działający na nieopisanych jeszcze typach
- funkcja działa na typach ogólnych - w momencie wywołania funkcji pod te typy ogólne podstawiane są konkretne typy (np. `int`, `char`)

```
template <typename T>
T MojaFunkcja(T a, T b, ...)
{
    // instrukcje
}
```

- słowo kluczowe **template** informuje, że funkcje będzie korzystała z fikcyjnego typu o nazwie `T`
- definicja dotyczy tylko jednej funkcji zdefiniowanej bezpośrednio po **template**

Szablony funkcji

- kompilator określa konkretny typ na podstawie typu argumentów w wywołaniu funkcji (proces ten nazywa się **konkretyzacją szablonu**)

```
int a1 = 3, b1 = 5;
cout << "int:   " << maks(a1,b1) << endl;
```

- jeśli typy nie zgadzają się, to wystąpi błąd kompilacji

```
cout << "inne:   " << maks(2.1f,5) << endl;
```

```
error C2782: 'T maks(T,T)' : template parameter 'T' is ambiguous
.\main.cpp(5) : see declaration of 'maks'
could be 'int '
or         'float'
```

- w takim przypadku należy jawnie określić typ

```
cout << "inne:   " << maks<float>(2.1f,5) << endl;
```

Szablony funkcji - definicja / deklaracja

- definicja szablonu funkcji:

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

- deklaracja szablonu funkcji:

```
template <typename T> T maks(T a, T b);
```

lub

```
template <typename T> T maks(T, T);
```

Szablony klas

- **szablony klas** (wzorce klas, klasy uogólnione) definiuje się na podobnej zasadzie jak szablony funkcji

```
template <class T>
class MojaKlasa
{
    // definicja klasy
};
```

- słowo kluczowe **template** informuje, że klasa będzie korzystała z fikcyjnego typu o nazwie **T**
- definicja dotyczy tylko pojedynczej klasy zdefiniowanej bezpośrednio po **template**

```
template <class T> class MojaKlasa
{
    // definicja klasy
};
```

Szablony klas - przykład

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

template <class T>
class MojaKlasa
{
public:
    T zmienna;
    MojaKlasa(T x)
    {
        zmienna = x;
    }
    void drukuj()
    {
        cout << zmienna;
        cout << endl;
    }
};
```

```
int main(void)
{
    MojaKlasa<int> MK_int(21);
    MK_int.drukuj();

    MojaKlasa<float> MK_float(3.5f);
    MK_float.drukuj();

    MojaKlasa<char> MK_char('a');
    MK_char.drukuj();

    MojaKlasa<string> MK_string("Txt 1");
    MK_string.drukuj();
}
```

```
21
3.5
a
Txt 1
```

Szablony klas - przykład

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

template <class T>
class MojaKlasa
{
public:
    T zmienna;
    MojaKlasa(T x);
    void drukuj();
};
```

```
template <class T>
MojaKlasa<T>::MojaKlasa(T x)
{
    zmienna = x;
}

template <class T>
void MojaKlasa<T>::drukuj()
{
    cout << zmienna;
    cout << endl;
}

int main(void)
{
    MojaKlasa<int> MK_int(21);
    MK_int.drukuj();

    ...
}
```

STL (ang. Standard Template Library)

- **iterator**
 - obiekt pozwalający przemieszczać się po elementach kontenera (przypomina wskaźnik używany do odwoływania się do elementów tablicy)
 - do iteratorów można stosować m.in. operatory: * (wyłuskania), ++, ==, !=
- **algorytmy**
 - grupa instrukcji opisujących sposób wykonywania konkretnego zadania, np. sortowanie, wyszukiwanie
 - szablony funkcji globalnych działające dla każdego kontenera (np. **sort**, **search**, **count**, **replace**)
 - algorytmy działają z iteratorami (ale także ze zwykłymi tablicami)

STL (ang. Standard Template Library)

- standardowa biblioteka szablonów (wzorców)
- opracowana w 1990 roku (Alex Stepanow, Meng Lee)
- w 1994 roku przyjęta jako standard - ANSI/ISO Standard C++
- zawiera szablony:
 - kontenerów (pojemników)
 - iteratorów
 - algorytmów
- **kontener**
 - jednostka umożliwiająca przechowywanie wielu wartości tego samego typu (kontenery w STL są homogeniczne - jednorodne)
 - przykłady: **vector**, **deque**, **list**, **set**, **multiset**, **map**, **multimap**
 - wybór kontenera zależy od problemu, który rozwiązujemy

STL (ang. Standard Template Library)

- **deque** (kolejka, double ended queue)
 - kontener sekwencyjny - pozycja elementu w kontenerze zależy od czasu i miejsca, gdzie został wstawiony, a nie od jego wartości
 - dynamiczna tablica dwukierunkowa (kolejka o dwóch końcach)
 - umożliwia szybkie dodawanie i usuwanie elementów na obu końcach
- **list** (lista)
 - kontener sekwencyjny, lista dwukierunkowa
 - brak indeksowania elementów - elementy można dodawać i usuwać wszędzie (na początku, na końcu, w środku)
- **set** (zbiór), **multiset** (multi-zbiór)
 - kontener asocjacyjny - aktualna pozycja elementu w pojemniku zależy od jego wartości (automatycznie sortuje wartości)
 - implementowany w postaci zbalansowanego drzewa binarnego (BST)
 - każdy element może wystąpić tylko raz (**set**) lub wiele razy (**multiset**)

STL (ang. Standard Template Library)

- **map** (mapa, słownik), **multimap** (multi-mapa)
 - zawiera pary elementów: klucz / wartość
 - kontener asocjacyjny (automatycznie sortuje elementy względem klucza)
 - implementowany w postaci zbalansowanego drzewa binarnego (BST)
 - każdy klucz może wystąpić tylko raz (**map**) lub wiele razy (**multimap**)

STL - kontener vector

- dynamiczna tablica (wektor) mogąca zawierać elementy dowolnego typu
- wymaga dołączenia pliku nagłówkowego: `#include <vector>`
- można stworzyć pusty wektor i rozszerzyć go (zmniejszyć) na bieżąco

```
vector<int> vec;
```

- można określić rozmiar wektora przy deklaracji

```
vector<double> vec(10);
```

- można określić rozmiar wektora przy deklaracji i zainicjalizować go 0

```
vector<double> vec(10, 0.);
```

- **vector** działa najszybciej, gdy z góry zarezerwujemy obszar pamięci - metoda **reserve()**

STL - metody kontenera vector

- **push_back()** - dodanie elementu na końcu
- **pop_back()** - usunięcie elementu z końca
- **size()** - zwraca liczbę elementów
- **at(i)** - zwraca element o indeksie i (sprawdza poprawność indeksu i)
- **[i]** - zwraca element o indeksie i (nie sprawdza poprawności indeksu i)
- **clear()** - usuwa wszystkie elementy
- **max_size()** - zwraca maksymalny rozmiar kontenera
- **empty()** - sprawdza czy kontener jest pusty
- **front()** - zwraca pierwszy element wektora
- **back()** - zwraca ostatni element wektora