AGH University of Krakow | Faculty of Mechanical Engineering and Robotics
| Department of Robotics and Mechatronics

MECHATRONIC ENGINEERING | I cycle | semester VI | 2025/2026

Object oriented programming and software engineering

Instruction IX:

Function templates, class templates, STL, algorithms and iterators

You will learn:

How to write generic code using templates in C++. You will understand how function templates and class templates allow you to create reusable components that work with different data types.

You will also work with the Standard Template Library (STL), which provides ready-to-use data structures and algorithms. You will learn how to store and process data using containers such as vectors and how to operate on them using iterators and standard algorithms.

By the end of the laboratory you will be able to combine templates and STL components to write more efficient and flexible programs.

Course supervisor:
dr inż. Lucjan Miękina, miekina@agh.edu.pl

Instruction author:
mgr inż. Joanna Koszyk, jkoszyk@agh.edu.pl

1. Initial information

Function template allows defining a function that works with different data types without rewriting the same code multiple times.

Class template defines a class that can operate on different data types.

STL (Standard Template Library) is a collection of containers (to store data), algorithms (to process data), and iterators (to access data).

Container stores collections of elements. Examples include:

std::vector – dynamic array
std::list – doubly linked list
std::queue – FIFO structure
std::set – ordered collections
std::map, std::multimap – key-value storage

STL containers differences:
std::vector – fast access, dynamic size
std::list – efficient insert/remove
std::queue – FIFO processing
std::set – unique sorted elements
std::map – key-value pairs
std::multimap – multiple values per key

Iterator is an object used to traverse elements in a container. It works similarly to a pointer.

Algorithms are functions provided by the STL to operate on data, such as sorting (std::sort), searching (std::find), and counting (std::count). Examples include:

std::find - find a specific value
std::count - count occurrences
std::count_if - count elements meeting a condition
std::min_element - get the minimum
std::max_element - get the maximum
std::sort - sort elements
std::reverse - reverse order
std::for_each - apply a function to all elements
std::transform - modify elements
std::replace - replace values
std::remove - remove elements
std::all_of - check if all elements meet condition
std::any_of - check if any element meets condition
std::none_of - check if none meet condition

2. Theoretical content

Example of a function template:

```
#include <iostream>
#include <string>

template <typename T>
T add(T a, T b) {
```

```
    std::cout<<"template"<<std::endl;
    return a + b;
}

template <typename T, typename U>
auto add(T a, U b) {
    std::cout<<"another template"<<std::endl;
    return a + b;
}

int main(){
    double x1 = 0.2;
    double y1 = 0.5;
    int x2 = 2;
    int y2 = 5;
    std::string var1 = "Hello ";
    std::string var2 = "world";
    std::cout<<add(x1,x2)<<std::endl;
    return 0;
}
```

Example of a class template:

```
#include <iostream>

template <typename T>
class Box {
private:
    T value;
public:
    void set(T v){
        value = v;
    }
    T get(){
        return value;
    }
};

int main() {
    Box<int> intBox;
    intBox.set(10);

    Box<double> doubleBox;
    doubleBox.set(1.2345);

    std::cout << intBox.get() << std::endl;
    std::cout << doubleBox.get() << std::endl;
}
```

std:vector sort() example:

```
#include <iostream>
#include <vector>
#include <algorithm>

int main(){
    std::vector<int> v = {3, 1, 4};
    std::sort(v.begin(), v.end());
    for(int x : v) {
        std::cout << x << " ";
    }
}
```

auto allows automatic type deduction:

```
auto x = 10;        // int
```

this pointer refers to the current object inside a class:

```
this->value = value;
```

File handling:

```
std::ofstream file("data.txt");
file << "Hello";
file.close();

std::ifstream file("data.txt");
std::string text;
file >> text;
```

3. Tasks

TASK 1.

Create a function template that performs operations of finding the maximum of two values and swapping two values. Use this function with different data types such as integers, floating-point numbers, and strings. Observe how the same function can operate on different types without modification.

Use auto where possible to simplify the code.

TASK 2.

Create a class template that stores elements using an STL container. The class should represent a sensor and store the measurements. The class should allow adding elements and retrieving them. Test the class with different data types and observe how it behaves.

TASK 3.

Create a program that stores data using multiple containers other than vector. Perform operations of adding elements, iterating using iterators and assigning values. Display the result.

TASK 4.

Add to Task3 STL algorithms for sorting elements, searching for values, count occurrences and other.

TASK 5.

Extend your program from previous tasks to save data to a file, load data from a file. Also use std::clog for logging and std::cerr for error handling.