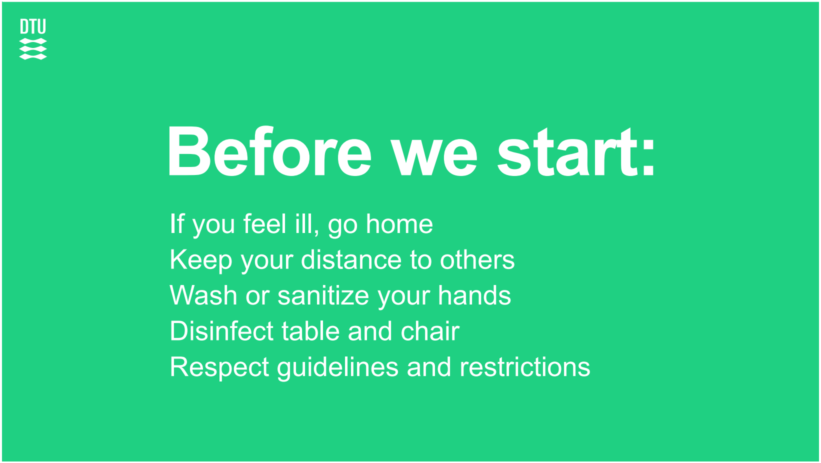
02393 Programming in C++



DTU

Before we start:

If you feel ill, go home

Keep your distance to others

Wash or sanitize your hands

Disinfect table and chair

Respect guidelines and restrictions

02393 Programming in C++
Module 7: Templates

Lecturer:

Alceste Scalas

(Slides based on previous versions by Andrea Vandin, Alberto Lluch Lafuente, Sebastian Mödersheim)

20 October 2020

Mid-Term Feedback

https://forms.office.com/Pages/ResponsePage.aspx?id=I_FR8s7JjkSSdzS7KFkR2biiX-4L0_RAnIPh1USR6YRUNkdPOTE2WTY5NlNZWVdRUVFMQU9EUTlBWS4u

The survey is anonymous

(but you will need to log in using your DTU credentials)

Lecture Plan

#	Date	Topic	Book chapter *
1	01.09	Introduction	
2	08.09	Basic C++	1
3	15.09	Data Types	2
4	22.09	Libraries and Interfaces	3
5	29.09		
6	06.10	Classes and Objects	4.1, 4.2 and 9.1, 9.2
		Autumn break	
7	20.10	Templates	4.1, 11.1
8	27.10	LAB DAY	Old exams
9	03.11	Inheritance	14.3, 14.4, 14.5
10	10.11	Recursive Programming	5
11	17.11	Linked Lists	10.5
12	24.11	Trees	13
13	01.12	Summary & Exam Preparation	
	07.12	Exam	

* Recall that the book uses sometimes ad-hoc libraries that are slightly different with respect to the standard libraries (e.g., strings and vectors).

OOP in C++: Recap

- A class is similar to a `struct`, but its members can be both variables and methods (a bit like functions)
- An object is an instance of a class
- Class members can be public or private
  - users of a class can only access public members (data encapsulation)

OOP in C++: Recap

- A class is similar to a `struct`, but its members can be both variables and methods (a bit like functions)
- An object is an instance of a class
- Class members can be public or private
  - ★ users of a class can only access public members (data encapsulation)

- Classes can have some **special methods**
  - ★ Constructor: called when an object is created (either statically, or dinamically using `new`)
  - ★ Destructor: called when an object is destroyed (either statically by exiting a scope, or dinamically using `delete`)
  - ★ Assignment: one can customise the behaviour of operator = (e.g., when the class internally uses dynamic allocation)

Vector of int, double, . . .

Last time we implemented our own vector class: `MyVector`

Objects of class `MyVector` can only contain `int` values

What if we need vectors of `double`s, or `bool`s, or `string`s. . . ?

Live coding

Vector of int, double, . . .

Last time we implemented our own vector class: `MyVector`

Objects of class `MyVector` can only contain `int` values

What if we need vectors of `double`s, or `bool`s, or `string`s. . . ?

Live coding

Almost the same code: copy & paste. . .

Not very maintanable: many copies of almost the same code

- What if we need to change a functionality?
- What if we discover a bug?

Templates in C++

Templates: a key feature of C++ enabling *generic programming*

Using templates, we can write code that is generic w.r.t. some arguments (types, classes, numbers, . . .)

- Payoffs: **write less code** and **avoid duplication**

The C++ Standard Library provides many facilities based on templates (e.g. containers like `vector`, `set`)

Templates in C++: Function templates

Example: the function `max` on `int`s

```cpp
int max(int a, int b) {
    if (a < b)
        return b;
    else
        return a;
}
```

Templates in C++: Function templates

Example: the function `max` on `int`s can be made generic:

```
int max(int a, int b) {
    if (a < b)
        return b;
    else
        return a;
}
```

⟹

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

Templates in C++: Function templates

Example: the function `max` on `int`s can be made generic:

```
int max(int a, int b) {
    if (a < b)
        return b;
    else
        return a;
}
```

⟹

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

We can then instantiate the function to our needs:

```
int x = max<int>(2, 3);
double y = max<double>(1.2, 3.5);
char z = max<char>('a', 'b');
```

Note: some instances of the function template may not make sense and/or may not compile unless *specialized* — see last slide

Templates in C++: Class templates

Templates can also be used to define *generic classes*

For example, the following code defines a class of pairs of elements having generic types `A` and `B`

```
template <class A, class B>
class Pair {
private:
     A a;
     B b;
      ...
}
```

Live coding

Templates in C++: Specialization

Templates can be refined for specific cases

E.g., if we have a templated max() and a BankAccount struct:

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

```
struct BankAccount {
    int amount;
};
```

If a and b are BankAccounts, then max(a,b) does not compile!

Templates in C++: Specialization

Templates can be refined for specific cases

E.g., if we have a templated `max()` and a `BankAccount` struct:

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

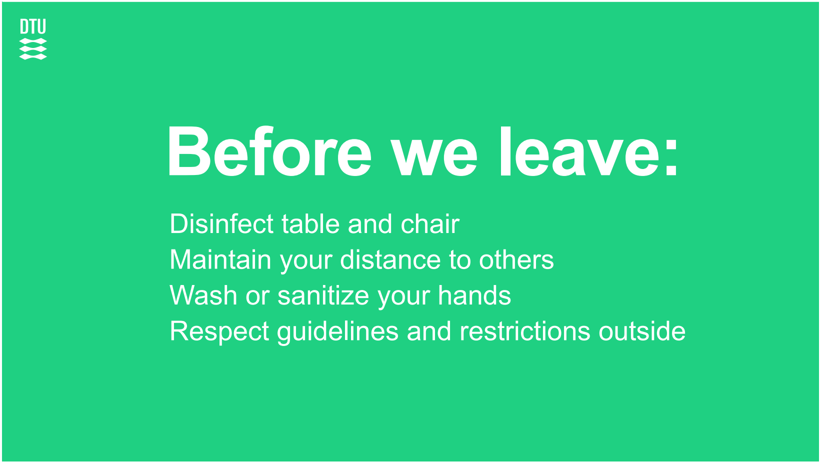
```
struct BankAccount {
    int amount;
};
```

If `a` and `b` are `BankAccount`s, then `max(a,b)` does not compile!

We can fix this by adding a specific behaviour for `max`:

```
template<>
BankAccount max(BankAccount a, BankAccount b) {
    if (a.amount < b.amount) return b;
    else return a;
}
```

02393 Programming in C++



Before we leave:

Disinfect table and chair
Maintain your distance to others
Wash or sanitize your hands
Respect guidelines and restrictions outside