02393 Programming in C++
Module 7: Classes and Objects III

Alberto Lluch Lafuente

Sebastian Mödersheim (slides author)

March 14, 2016

Lecture Plan

#	Date	Topic	Chapter *
1	1.2	Introduction	1
2	8.2	Basic C++	1
3	15.2	Data Types	2
4	22.2		
		Libraries and Interfaces	3
5	29.2		
6	7.3	Classes and Objects	4.1, 4.2 and 9.1, 9.2
7	14.3	Templates	4.1, 11.1
		Påskesferie	
8	4.4	Inheritance	14.3, 14.4, 14.5
9	11.4	Recursive Programming	5-7
10	18.4	Lists and Trees	10.5, 11, 13.1
11	25.4	Trees	13
12	2.5	Graphs	16
13	9.5	Summary	
	17.5	Exam	

OOP Basics—Summary

- A class is like a (struct) record with
  - ★ member variables and methods
- Object: instance of a class.
- Members can be public or private.
  - ★ Allows to realize ADTs: the user of a class cannot directly manipulate private members that implement the class, but only call public functions. Aka data encapsulation
  - ★ We can change the implementation without changing the calling program.
- Some special methods:
  - ★ Constructor: called when an object is created, e.g. a statically declared object or one dynamically allocated with `new`.
  - ★ Destructor: called when an object is deallocated, e.g. when the scope of a statically allocated object finishes or when a dynamically allocated object is deallocated with `new`.
  - ★ Assignment: there is an implicit assignment operator = but in some cases one needs to customize it (e.g. when the implementation uses dynamic allocation).

Templates in C++

- Templates are the main feature of the *generic programming* paradigm;
- The main idea is to allow us to write less code;
- In particular we can specify code that is generic with respect to some arguments (types, classes, numbers);
- The C++ Standard Library provides many useful functions based on templates (e.g. containers like `vector`, `set`);

Templates in C++: Function templates

The simplest form of a template is a function template.
For example, the function `max` can be implemented for a generic type:

```
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>(1,3);
char z = max<char>("a","b");
```

Note: some instances of the function template may not make sense or may lead to errors, e.g. `max<vector>(u,v)`.

Templates in C++: Specialization

Templates can be refined for specific cases.
For example, if we want to have a specific behaviour for max on vectors, we could write

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

Templates in C++: Class templates

Templates can also be used to define generic classes.
For example,

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

specifies a class of pairs of elements of generic types/classes `A` and `B`.