

EL2310 – Scientific Programming

Lecture 13: OOP and Classes in C++



Hakan Karaoguz (hkarao@kth.se)

Royal Institute of Technology – KTH

Overview

Lecture 13: OOP and Classes in C++

- Announcements

- Wrap Up

- Classes in Depth

- Overloading of Functions and Operators

Last time

- ▶ OOP concepts in C++
- ▶ Introduction to Classes

Today

- ▶ Classes in Depth
- ▶ Overloading of Functions and Operators

Announcements

- ▶ This week's schedule:
 - ▷ 3,4,5 October → We will have lecture
 - ▷ 10 October → We will have the help session for C++ project
- ▶ C++ project will be announced this week

Lecture 13: OOP and Classes in C++

Announcements

Wrap Up

Classes in Depth

Overloading of Functions and Operators

Passing Arguments by Reference

- ▶ Declaration: `void fcn(int &x);`
- ▶ Any changed to `x` inside `fcn` will affect the parameter used in the function call

- ▶ Ex:

```
void fcn(int &x)
{
    x = 42;
}
```

```
int main()
{
    int x = 1;
    fcn(x);
    cout << "x=" << x << endl;
}
```

- ▶ Will change value of `x` in the scope of `main` to 42

Dynamic Memory Allocation in C++

- ▶ In C we used `malloc` and `free`
- ▶ In C++ the `new` and `delete` operators are used
- ▶ Ex:

```
int *p = new int;  
*p = 42;  
...  
delete p;
```

Key Concepts of OOP

- ▶ Classes (types)
- ▶ Instances (objects)
- ▶ Functions (Methods)
- ▶ Interfaces
- ▶ Encapsulation
- ▶ Polymorphism
- ▶ Inheritance
- ▶ Access protection - information hiding

Class definition

► Syntax:

```
class ClassName {  
    public:  
        void fcn();  
    private:  
        int m_X;  
}; // Do not forget the semicolon!!!
```

- As a general rule, class names start with capital letter!
- `m_X` is a member data
- `void fcn()` is a member function
- `public` is an access specifier specifying that everything below can be access from outside the class
- `private` is an access specifier specifying that everything below is hidden from outside of the class

Constructor

- ▶ Constructor is a special kind of function.
- ▶ The constructor tells how to “setup” the objects
- ▶ *default constructor*:
 - ▷ It is a constructor without arguments
 - ▷ Implicitly defined by compiler when it is not explicitly defined by the user
- ▶ When an object of a certain class is created (instantiated), the so-called *constructor* is called first
- ▶ The constructor has the same name as the class and has no return type

```
class A {  
public:  
    A() {}  
};
```

Lecture 13: OOP and Classes in C++

Announcements

Wrap Up

Classes in Depth

Overloading of Functions and Operators

Source and header files

- ▶ OOP allows us to generate hierarchical code structure. As such classes are defined in separate files other than the main file to isolate different components.
- ▶ For classes, it is a good programming practice to separate the class *interface* from the class *implementation*.
- ▶ The class *interface* goes into the header file .h
- ▶ The class *implementation* goes into the source file .cpp
- ▶ A radio analogy → We use radio through buttons and knobs (interfaces) but we don't see how it runs in inside (implementation).
- ▶ The header and source files are named the same as the class name. Ex: If class name is Car, car.h and car.cpp

Source and header files

Header file ex A.h:

```
// Preprocessor guards
#ifndef A_H
#define A_H
class A{
public:
    A();
private:
    int m_X;
}; // Don't forget the
semicolon!!
#endif
```

Source file ex A.cpp:

```
#include "A.h"
A::A() {
    m_X=0;
}
```

Task 1

- ▶ Download the task1-examples.zip file from the course website
- ▶ Try to compile using the command:
`g++ task1.cpp gradebook.cpp -o task1`
- ▶ Try to fix the code such that it works

Access specifiers

- ▶ `private`: can be accessed from:
 - ▷ inside of the class
- ▶ `public`: can be accessed from:
 - ▷ inside of the class
 - ▷ subclasses
 - ▷ outside of the class
- ▶ `protected`: can be accessed from:
 - ▷ inside of the class
 - ▷ subclasses

Setters and Getters

- In order to modify/access to the private data members of a class, we use *set* and *get* functions.

A.h:

```
Class A {  
    public:  
        A(); // Constructor  
        void setm_X(int x);  
        int getm_X();  
    private:  
        int m_X;  
};
```

A.cpp:

```
A::A() {m_X = 0;}  
void A::setm_X(int x) {  
    m_X = x;  
}  
int A::getm_X() {  
    return m_X;  
}
```

Task 2

- ▶ Use the previously defined Gradebook class
- ▶ Add a *private* data member `string courseName`
- ▶ Add the necessary *get* and *set* functions to access/modify `courseName`.
- ▶ Try these functions in the main

Constructor with default arguments

► Ex:

```
Class Time {  
    public:  
    // Constructor with default arguments  
    Time(int hour = 0, int minute = 0, int second = 0);  
    ...  
};
```

► In main:

```
int main() {  
    Time t1; // All arguments are defaulted  
    Time t2(12); // Hour is initialized, others are  
    defaulted  
    Time t3(12,15); // Hour and minute are initialized,  
    second is defaulted  
    Time t4(12,15,20); // All arguments are initialized  
    return 0; }
```

Destructor

- ▶ To perform cleanup an object is deleted
- ▶ Only 1 destructor in a class
- ▶ Called before the object is destroyed
- ▶ Syntax: `~ClassName();`
- ▶

```
Class A {  
    public:  
        A(); // Constructor  
        ~A(); // Destructor  
        ...  
};
```

keyword `const`

- ▶ To make some functions, data members and objects as "read-only"
- ▶ `const` function type:
- ▶ Ex: `void fcn(int arg) const;`
- ▶ `const` data members:
- ▶ Ex: `const int m_X;`
- ▶ `const` data members cannot be modified by assignment.
- ▶ `const` objects:
- ▶ Ex: `const A a;`
- ▶ `const` objects can only use `const` member functions
- ▶ *constructors* cannot be `const`!! But they can be used to initialize `const` objects

Static members

- ▶ A `static` member (data/function) is the same across all objects.
- ▶ It's a special member of a *class* that can be accessed even if there is no object of that class!
- ▶ Ex: `int A::m_Counter = 0;` if `m_Counter` is a static data member of class `A`
- ▶ `static` member functions cannot be `const`. Because `const` member functions only work for the object that it operates.

this pointer

- ▶ Inside class methods you can refer to the object with `this` pointer
- ▶ The `this` pointer cannot be assigned (your program decides it run-time)
- ▶ Static members or functions cannot be referred by `this`!

Dynamic allocation of objects

- ▶ One reason to use dynamic memory allocation (`new/delete`):
 - ▷ Moving around pointers to BIG chunks of memory (avoiding unnecessary copying)
- ▶ Makes sense not only for arrays
- ▶ Objects can also be BIG (e.g. database object can be 500MB!)
- ▶ Typically, we dynamically allocate objects
- ▶ We free memory when the object is no longer needed
- ▶ We pass objects by reference (`*` or `&`) to functions
- ▶ Example:

```
Database db = new Database("mydatabase.db");  
useDb(db); // void useDb(Database *db)  
delete db;  
db = NULL;
```

Lecture 13: OOP and Classes in C++

Announcements

Wrap Up

Classes in Depth

Overloading of Functions and Operators

Function overloading

- ▶ We can create functions and methods with the same name, but different arguments
- ▶ It is not possible to overload by changing return type
- ▶ Example:

```
void method();  
void method(int a);  
void method(int b, double c);  
void method(int b); WRONG!  
int method(int b); WRONG!
```

Operator overloading

- ▶ Operators behave just like functions

- ▶ Compare

```
Complex& add(const Complex &c);
```

```
Complex& +=(const Complex &c);
```

- ▶ You can overload (provide your own implementation of) most operators
- ▶ It will not change the behavior for other classes only the one which overloads the operator

Task 3

- ▶ Use the previously defined Gradebook class
- ▶ Overload the function `displayMessage` to accept a string and an integer.
- ▶ In the main function, initialize a normal Gradebook object with using constructor such as `Gradebook gradebook("EL2310")` and use its member function `displayMessage()` to output information.
- ▶ Now dynamically initialize a Gradebook object, and use the overloaded `displayMessage` function to output information.
- ▶ Don't forget to delete dynamically initialized object before finishing execution.