

# EL2310 – Scientific Programming

## Lecture 12: Memory Management, Object Oriented Programming and Classes in C++

Hakan Karaoguz (hkarao@kth.se)

Credits: Ramviyas Parasuraman, Andrej Pronobis (U.Washington), Florian Pokorny (UC Berkeley)

KTH - Royal Institute of Technology

# Overview

## Lecture 12: Memory Management, OOP and Classes in C++

Wrap Up

Memory Management in C++

Introduction to Object Oriented Paradigm

Classes

## Wrap Up

# Introduction to Object Oriented Paradigm

## Classes

- ▶ History of C++
- ▶ Differences between C++ and C
- ▶ Printing and getting user input
- ▶ Namespaces

## C++ subsumes C

- ▶ You can use all you learned in C in C++ as well
- ▶ Some constructs/syntax have a C++ version
- ▶ In C++, the file ending is typically .cc or .cpp for source files and .h, .hh or .hpp for header files
- ▶ In this course we will use .cpp and .h
- ▶ `g++` is specific to C++ by (auto) linking to std C++ libraries
- ▶ Usage and command line options for `g++` are the same as for `gcc`
- ▶ Make sure you know how to use `make` for this part of the course!
- ▶ New variable `bool`: `true/false`
- ▶ `string`: “real” string (use `#include <string>`)

## Getting input from the user

- ▶ In C++ we use *streams* for input and output using the `<iostream>` library
- ▶ streams is also used to get input from console
- ▶ Ex:  

```
int value;  
std::cout<<"Please enter an integer value:  ";  
std::cin >> value;
```
- ▶ When reading multiple inputs, they are separated by spaces
- ▶ Ex: `std::cin >> a >> b >> c;`

# Hello KTH in C++ vs C

```
#include <iostream>
int main ()
{
    std::cout << "Hello
KTH!"<<std::endl;
    return 0;
}
```

```
#include <stdio.h>
int main()
{
    printf("Hello KTH! \n");
    return 0;
}
```

# Namespaces

- ▶ In C all functions share a common `namespace`
  - ▷ This means that there can only be one function for each function name
- ▶ In C++ functions can be placed in specific namespaces

- ▶ **Syntax:**

```
namespace NamespaceName {  
    void fcn(); ...  
}
```

# Announcements

- ▶ The C assignment deadline 4 October 23:55
- ▶ Next week's schedule
  - ▷ 3 October Monday Lecture
  - ▷ 4 October Tuesday Lecture
  - ▷ 5 October Wednesday Lab Session → Room 523 Teknikringen 14 Plan 5

# Today

- ▶ Memory Management in C++
- ▶ Introduction to OOP
- ▶ Introduction to Classes

## Lecture 12: Memory Management, OOP and Classes in C++

Wrap Up

**Memory Management in C++**

Introduction to Object Oriented Paradigm

Classes

## Passing Arguments by Reference

- ▶ Standard function calls are by value
- ▶ Value of the variable is copied into the function
- ▶ Pointers offered a way in C to do this call by reference
- ▶ Call by reference avoids the need to copy all the data
- ▶ Ex: Not so good to copy an entire 10Mpixel image into a function, better to give a reference to it (i.e. tell where it is)
- ▶ In C++ we can use references

## Passing Arguments by Reference, Cont'd

- ▶ 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;
```

# Dynamic Allocation of Arrays

- ▶ If you allocate an **array** with `new` you need to delete with `delete []`
- ▶ Ex:  

```
int *p = new int[10];  
p[0] = 42;  
delete [] p;
```
- ▶ A typical mistake: forgotten `[]`

# Task 1

- ▶ Please open `cpp.sh` from your web browser
- ▶ Dynamically allocate a 5 by 5 double matrix using C++
- ▶ Set the `matrix[4][3]` to a random value and then print the value by accessing to the matrix
- ▶ delete the matrix

## Lecture 12: Memory Management, OOP and Classes in C++

Wrap Up

Memory Management in C++

Introduction to Object Oriented Paradigm

Classes

# The Object-Oriented Paradigm

## Motivation:

- ▶ We are trying to solve complex problems
  - ▷ Complex code with many functions and names
  - ▷ Difficult to keep track of all details
- ▶ How can we reduce the complexity?
  - ▷ Grouping related things
  - ▷ Abstracting things away
  - ▷ Creating hierarchies of things
- ▶ Advantages:
  - ▷ Re-usable and reliable code
  - ▷ Ease of debugging

# Key Concepts of OOP

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

# Object Oriented Programming (OOP)

## ► Inheritance

- ▷ Support for hierarchies (most knowledge can be structured by hierarchical classifications)
- ▷ Ex: A car is a motor vehicle which is a vehicle which is a transportation system which is a . . .
- ▷ Subclass to inherit the properties of the base class

## Lecture 12: Memory Management, OOP and Classes in C++

Wrap Up

Memory Management in C++

Introduction to Object Oriented Paradigm

**Classes**

# Classes

- ▶ A `class` is an “extension” of a `struct`
- ▶ A class can have both data member and function members (methods)
- ▶ Classes bring together data and operations related to that data
- ▶ Like C structs, classes define new data types

# Class definition

► **Syntax:**

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

- **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**

# Classes and Objects

- ▶ Classes define data types
- ▶ Objects are instances of classes
- ▶ Objects correspond to variables
- ▶ Instantiating a class (Declaring an object):

```
ClassName variableName;
```

# Your First Class

```
#include <iostream>
#include <string>
using namespace std;
class Gradebook {
public:
    void displayMessage(){
        cout<<"Welcome to Gradebook Class!!"<<endl;
    }
};
int main() {
    Gradebook gradebook;
    gradebook.displayMessage();
    return 0;
}
```

## Task2

- ▶ Please open cpp.sh from your web browser
- ▶ Use the previously created Gradebook class
- ▶ Modify the member function `displayMessage` to accept a string as an argument and output "Welcome to Gradebook class for X" where X is the input string.
- ▶ Ask the user for a string input and pass the inputted string to `displayMessage` function
- ▶ Ex output: "Welcome to Gradebook class for EL2310"

# Access specifiers

- ▶ There are three access specifiers:
  - ▷ `public`
  - ▷ `private`
  - ▷ `protected`
- ▶ No access specifier specified  $\Rightarrow$  assumes it is `private`
- ▶ Data and function members that are `private` cannot be accessed from outside the class
- ▶ `protected` will be discussed later

# 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() {}  
};
```

# Constructor Example

```
class A {  
public:  
    A(int x){  
        m_X = x;  
    }  
    int getValue() { return m_X; }  
private:  
    int m_X;  
};
```

Now assume you are in main function:

```
A a;  
std::cout << a.getValue() << std::endl;
```

# Multiple Constructors

- ▶ You can define several different constructors

```
▶ class MyClass {  
    public:  
        MyClass() {}  
        MyClass(int value) {  
            m_X = value;  
        }  
        int getValue() { return m_X; }  
    private:  
        int m_X;  
};  
MyClass a; // Default constructor  
MyClass aa(42); // Constructor with argument  
std::cout << a.getValue() << std::endl;
```

# Task3

- ▶ Please open `cpp.sh` from your web browser
- ▶ Use the previously created `Gradebook` class
- ▶ Write a constructor for `Gradebook` class that takes a string as an argument and calls its member function *displayMessage* with that string argument
- ▶ In the main function create a `Gradebook` object with the constructor that you have just created