

# C++ for Image Processing: The C++ part

## Lecture 1

## The course

- The C++ part of the course
  - 5 teaching hours
  - lab sessions to support lectures
- Aim
  - to make you aware of main issues
  - crash-course
  - aim is **not** to teach you C++ from scratch

## Today's session

- The aim is to understand how these concepts are used in C++, and to be able to write programs that use them:
  - Classes/Objects
  - Constructors/destructors
  - Protection
  - Inheritance

## Some introductory stuff

- C++ is a popular OOP language
  - however, it is not a *pure* OOP language
  - which is not a bad thing
- It is based on C
- You tend to have much greater control on the code than with e.g. Java
- Your code will generally be much faster than Java code

## Some C++ code

- Hello world !!

```
#include <iostream>  
using namespace std;
```

preprocessor  
includes header file

```
int main() {  
    cout<<"Hello world !"<<endl;  
}
```

## Some C++ code

- Hello world !!

```
#include <iostream>  
using namespace std;
```

specifies use of a  
given namespace

```
int main() {  
    cout<<"Hello world !"<<endl;  
}
```

## Some C++ code

- Hello world !!

```
#include <iostream>
using namespace std;
int main() {
    cout<<"Hello world !"<<endl;
}
```

main function for C++

## Some C++ code

- Hello world !!

```
#include <iostream>
using namespace std;
int main() {
    cout<<"Hello world !"<<endl;
}
```

C++ output

## Comparing with Java

```
#include <iostream>
using namespace std;
int main(int argc, char *argv[])
{
    cout << "Hello World\n";
}
```

```
public class Hello
{
    public static void main(String args[])
    {
        System.out.println("Hello World");
    }
}
```

- > You can have **global functions** that are not methods of a class
- > **#include** statement
- > Parameters in **main function**, and return type of the main function

## Components of C++ software

- header files (.h)
  - global declaration, class declarations
  - **#include** other files (should be guarded against multiple inclusions)
  - **#define** constants - forgetting parentheses can be bad:

```
#define TIMES(a, b) a * b
int x = TIMES(3 + 5, 4 + 2);
// Compiler sees: int x = 3 + 5 * 4 + 2;
// This is not what you want!
```

- implementation files (.cc or .cpp)
  - all implementation of code elements

## Basic C++ input and output

- standard input (cin)

```
int value,input;
cin>>value>>input;
```

- standard output (cout)

```
cout<<"Hello world!"<<endl;
```

## Basics : selection statements

```
switch(variable)
{
    case A:
        do something;
        break;
    default :
        do something else;
}

if(condition) { do something;}
else { do something else;}
```

if(variable == A) do something

if-else clause

## Basics : iterations

```
for(int i=0; i<10; i++)  
{  
    do something;  
}  
  
while(condition)  
{  
    do something;  
}
```

loop for 10 times

loop till condition holds

## :: Classes

```
[Foo.h]  
class Foo {  
public:  
    Foo();  
    ~Foo();  
    int myMethod(int a, int b);  
};
```

```
[Foo.cc]  
#include "Foo.h"  
#include <iostream>  
using namespace std;  
Foo::Foo()  
{ printf("I am a constructor\n");  
  int a = myMethod(5,2);  
  printf("a=%d\n",a); }  
Foo::~~Foo()  
{ printf("I am a destructor\n"); }  
int Foo::myMethod(int a, int b)  
{ return a+b; }
```

## :: Classes

- Splitting a program into a **header file** (Foo.h) and a **program file** (Foo.cc)
  - header file has class declarations
  - program file has method definitions
  - easy to identify the interface for a class (.h)
- Scope operator (::)**
  - used when defining methods
  - know for which class a method is defined

## Inlining

```
[Color.h]  
class Color  
{  
public:  
    Color();  
    int getRed() { return red; }  
    void setRed(int s_red) { red=s_red; }  
protected:  
    int red, green, blue;  
};
```

- Possible to declare and define methods in the same place
  - not for medium & large-sized methods, esp. if they are called frequently
- Inlining can lead to efficient or inefficient code
  - depends on how you use it

## Constructors

```
[Foo.h]  
class Foo {  
public:  
    Foo();  
    Foo(int a_, int b_, double  
        x_, double y_);  
protected:  
    int a, b;  
private:  
    double x, y;  
};
```

```
[Foo.cc]  
#include "Foo.h"  
using namespace std;  
Foo::Foo()  
{  
    a=1; b=2; x=3.14; y=2.718;  
}  
Foo::Foo(int a_, int b_, double x_,  
         double y_)  
{  
    a = a_; b=b_; x=x_; y=y_;  
}
```

## Objects

- In C++ **object variables hold values, not object references**
  - `Foo f(3, 4, 2.34, 1.456);`
  - creates a new object `f` by calling the constructor
  - If you do not supply construction parameters, the object is constructed with the default constructor

## Objects

- Difference from Java
  - `Foo f1; /*construct f1 with Foo::foo() */`
    - in Java this creates an uninitialised reference
    - no **new** operator in C++
  - Object assignments (another difference)
    - `Foo f1 = f2;`
      - when we assign one object to another a **copy of the actual values** is created - modifying the copy does not change the original!!!
      - in Java this would only create a second reference to the object

## Protection

- **public, private, protected**
- **protected** members are only visible to subclasses
  - in Java a whole package can use protected members - no packages in C++
- Protection level needs to be specified in sections
  - you can have as many sections as you like
  - default level is **private**

## Protection: Friends

- It is possible for a non-member function to gain access to private members of a class
- **friend** has access to all private & protected members of the class for which it is a friend
- To declare a **friend** function, include its prototype within the class & precede it with the keyword **friend**

## Friends: Example

```
class myclass {
int a, b;
public:
    friend int sum(myclass x);
    void set_ab(int i, int j);
};
void myclass::set_ab(int i, int j)
{ a = i; b = j; }
int sum(myclass x)
/*because sum is a friend of myclass,
it can access a and b directly*/
{ return x.a + x.b; }
```

## Inheritance

- Very similar to Java
- You can have a **superclass** from where subclasses can inherit variables & methods
- Main difference:
  - in C++ you can have **multiple inheritance**
  - classes can inherit from more than one superclasses

## Inheritance example

```
class Manager : public Employee
{
public:
    Manager(string Name, double Salary, string Dept);
private:
    string Department;
};
```

> **class derived-class : access base-class** (access is optional)

> There is no use of **extends** keyword

> **protected** and **public** variables or members of the base class are all accessible in the derived class, but the **private** member variables not accessible by the derived class.

## Inheritance example

```
Manager : : Manager(string Name, double Salary, string Dept)
: Employee(Name, Salary) /*call superclass constructor*/
{
    Department = Dept;
}
```

- > You call the superclass constructor **outside** the body of the subclass constructor
- > To call a method from the superclass you use the name of the superclass and the :: operator - **Employee::print()**;

## What is the output?

```
#include <iostream>
using namespace std;
class base {
public:
    base() {cout << "Constructing base\n";}
    ~base() {cout << "Destructing base \n";}
};
class derived: public base {
public:
    derived() {cout << "Constructing derived\n";}
    ~derived() {cout << "Destructing derived\n";}
};
int main() {
    derived ob;
    return 0; }
```

## The answer

- After running it you should get:

```
Constructing base
Constructing derived
Destructing derived
Destructing base
```

## Some fine differences to Java

- C++ **compiler** will not check if **all local variables are initialised** before they are read
  - if left uninitialised, compiler will assign whatever value was in the memory location that the value occupies
- In C++ **constants can be declared everywhere** (not only static data of a class)
  - `const int DAYS_PER_WEEK = 7;`

## A Makefile for example

```
# Entries to compile c++ programmes to .obj files (Compilation)
main.o: main.cpp Point.h Rectangle.h
g++ -c main.cpp
Point.o: Point.cpp Point.h
g++ -c Point.cpp
Rectangle.o: Rectangle.cpp Rectangle.h Point.h
g++ -c Rectangle.cpp

# Entries to bring the executable up to date (linking)
main: main.o Point.o Rectangle.o
g++ -o main main.o Point.o Rectangle.o

# Entries to compile and link with all warnings
g++ -o main -Wall main.cpp Point.cpp Rectangle.cpp
```

## Next weeks

- Memory management
- Arrays
- Pointer arithmetic
- I/O, file types, image I/O
- Debugging