EL2310 – Scientific Programming

Lecture 13: Intro to C++

Carl Henrik Ek
(chek@csc.kth.se)

Royal Institute of Technology – KTH

Overview

Lecture 13: Intro to C++

- Object Oriented Programming

C++ Basics

- Namespaces, Printing and User Input
- Reference and Pointers
- Classes
- Code Structure

Course so far

- MATLAB: Using program to achieve a goal
- C: Learning how to program

Rest of the course

- C++
  - Writing extendable programs
  - Using other peoples code
  - Extending other peoples code
  - Writing re-useable code

Today

- Object oriented programming
- Intro to C++

Lecture 13: Intro to C++

Object Oriented Programming

C++ Basics

- Namespaces, Printing and User Input
- Reference and Pointers
- Classes
- Code Structure

The Object-Oriented Paradigm

The motivation:

- Abstraction levels
- Intuitiveness
- Modularity
  - Division of labor and responsibilty
  - Interchangeability
  - Confidentiality
  - Decomposability
  - Code reuse

The Object-Oriented Paradigm, cont'd

The tools:

- Classes
- Methods
- Information hiding
- Polymorphism
- Inheritance

Lecture 13: Intro to C++

Object Oriented Programming

C++ Basics

Namespaces, Printing and User Input
Reference and Pointers
Classes
Code Structure

C is a subset of C++

- You can use all you learned in C in C++ as well
- Some constructions have a C++ version

C++ Compiler

- Use `g++` instead of `gcc`
- Syntax the same as for C
- Should print the version of the `g++`
- Also make sure you know how to use `make`

Naming of files

- We named files in C .c (source) and .h (header)
- In C++ the 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

Comments in C++

- Can write comments as in C, i.e. `/* ...*/`
- Single line comments after `//`

```
int main() {
  // This is a comment
  ...
}
```

Basic data types

- All data types from C can be used plus e.g.
- `bool`: boolean value `true/false`
- `string`: "real" string (need to `#include <string>`

Declaration of variables

- You no longer need to declare the variable at the beginning of the function (scope)
- Useful rule of thumb: Declare variables close to where they're used.

Finally

```
for(int i=0;i<N;i++){...}
```

- `i` only defined within loop
- “Save” specific names for counters, `i,j,k,...`

Lecture 13: Intro to C++

Object Oriented Programming

C++ Basics

Namespaces, Printing and User Input

Reference and Pointers

Classes

Code Structure

Namespaces

- In C all function share a common `namespace`
- This means that there can only be one function for each function name
- Namespaces offers a way around this
- Functions are placed in namespaces
- Syntax:

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

Accessing functions in a namespace

- To access a function `fcn` in namespace `A`
  `A::fcn`
- This way you can have more than one function with the same name but in different namespaces

Task 1

- Write a program to test the idea with namespaces
- Define two functions `void fcn();` inside namespaces `A` and `B`

Printing to screen

- In C++ we use so called *streams* for input and output
- Output is handled with the stream `cout` and `cerr`
- All basic data types have the ability to add themselves to a stream for printing
- We use the `<<` operator
- Ex: `cout << "Hello world";`
- To add a line feed use the "\n" as in C or the special `endl`
- Ex: `cout << "Hello world" << endl;`

Printing to screen cont'd

- You can mix data types easily
- In C:
```
printf("The value is %d\n", value);
```
- In C++:
```
cout << "The value is " << value << endl;
```
- The stream `cerr` is the error stream
- Compare `stdout` and `stderr` in C

Getting input from the user

- You can quite easily get input from the user
- Use the `cin` stream
- Ex:

```
int value;
cin >> value;
```

- Using `cin` will flush the `cout` stream

Hello world in C++

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

- `<iostream>` replaced `<stdio.h>`
- Standard C++ header files are included without the suffix (no .h at the end)
- Here the `std` namespace is used, where `cout` is found

Task 2

- Write a program that reads the name and age of a person
- It should then print this info on the screen

Call 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 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++ the support for call by reference is built-in

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);   std::cout << "x=" <<
x << std::endl; }
```

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

Pointers vs References

- Try to use references when possible
- Much less error prone constructions
- References need to be assigned constructed
- Ex: This is not allowed

```
int &x;
int y;
x = y;
```

Allocating memory

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

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

- If you allocate an array with `new` you need to delete with `delete []`

Classes

- C++ is an object oriented programming language
- Classes play a key role here
- A `class` is an "extension" of a `struct`
- A class can have both data member and function members

Why classes

- Encapsulate data (same motive as a `struct`)
- Ease of creation/maintenance
- Ease of understanding
- Create reusable components
- Bundle data and functions to process the data

Class definition

- Syntax:

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

- `m_X` is a member data
- `void fcn()` is a member function
- `public` is an access specifier telling that everything after it can be accessed from outside the object

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 object
- Ex: `m_X` above cannot be accessed from outside
- Will come back to `protected`

Classes and Objects

- Classes define data types
- Objects are instances of classes
- Objects correspond to variables

Classes and Namespace

- The class defines a namespace
- Hence function names inside a class do not name clash with other functions
- Example: the member variable `m_X` above is fully specified as `ClassName::m_X`

Constructor

- When an object of a certain class is created the so called *constructor* is called
- The constructor tells how to “setup” the objects
- The constructor that does not take any arguments is called the *default constructor*
- The constructor has the same name as the class and has no return type

Constructor

- Some types cannot be assigned, only initialized
- Ex.: references
- These data members should be initialized in the *initializer list* in the constructor

Constructor

```
class A {
public:
  A():m_X(1) {}
  int getValue() { return m_X; }
private:
  int m_X;
};
A a;
std::cout << a.getValue() << std::endl;
```

Constructor

- You can define several different constructors
- ```
  class MyClass {
  public:
    MyClass():m_X(1) {}
    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;
  std::cout << aa.getValue() << std::endl;
  ```
  ```

Destructor

- When an object is deleted the destructor is called
- The destructor should clean up things
- For example free up dynamically allocated memory
- There is only 1 destructor
- If not declared a default one is used which will not free up dynamic memory
- Syntax: `~ClassName();`
- ```
  Class A {
  public:
    A(); // Constructor
   ~A(); // Destructor
  ...
  };
  ```
  ```

Lecture 13: Intro to C++

Object Oriented Programming

C++ Basics

Namespaces, Printing and User Input
Reference and Pointers
Classes
Code Structure

Source and header file

- Normally you split the definition from the declaration like in C
- The definition goes into the header file .h
- The declaration goes into the source file .cpp
- Header file ex:

```
class A{
public:
  A();
private:
  int m_X;
};
```

- Source file ex:

```
#include "A.h"
A::A():m_X(0)
```

Task 3

- Implement a class the defines a Car
- Should have a member variable for number of wheels
- Should have methods to get the number of wheels
- Write program that instantiate a Car and print number of wheels

Task 4

- Write class `Complex` for a complex number
- Provide 3 constructor
  - default which should give value 0
  - one argument which should give a real value
  - two arguments, real and imaginary part

`this` pointer

- Inside an object's methods you can refer to the object with the `this` pointer
- The `this` pointer cannot be assigned (done automatically)

Static members

- Members (both functions and data) can be declared `static`
- A `static` member is the same across all objects; it's a member of the *class*, not any single object
- That is all instantiated objects share the same `static` member
- You can use a `static` class member without instantiating any object

Next Time

- Lecture: Tuesday 4th of October, 10-12, M36
- More on Object Oriented Programming
- C-project deadline Thursday 6th of October