EL2310 – Scientific Programming

Lecture 14: Object Oriented Programming in C++

Ramviyas Parasuraman (ramviyas@kth.se)

Royal Institute of Technology – KTH

Overview

Lecture 14: Object Oriented Programming in C++

Wrap Up

Classes (cont'd)

More on Classes and Members

Group presentations

Last time

- Intro to C++
- Differences between C and C++
- Intro to OOP

Today

- Classes and more on OOP

Lecture 14: Object Oriented Programming in C++

Wrap Up

Classes (cont'd)

More on Classes and Members

Group presentations

C++ Compiler

- Use `g++` instead of `gcc`
- Usage and command line options are the same as for `gcc`
- Make sure you know how to use `make` for this part of the course!

Declaration of variables

- You no longer need to declare the variable at the beginning of the function (scope), as was the case for pre C99
- Useful rule of thumb: Declare variables close to where they're used.
- For instance:

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

  `i` only defined within loop
- Use specific names for counters, e.g. `i,j,k,...`

Namespaces

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

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

- To access a function `fcn` in namespace `A`

```
A::fcn
```

- To avid typing namespace name in every statement:

```
using namespace std
```

Printing to Screen

- In C++ we use *streams* for input and output
- Output is handled with the stream `cout` and `cerr`
- In C:

  ```
  printf("The value is %d\n", value);
  ```
- In C++:

  ```
  cout << "The value is " << value << endl;
  ```
- Just like in C you can format the output in a stream
- You can use

  `cout.width(10)` number of characters for output to fill
  `cout.precision(3)` number of digits
  `cout.fill('0')` pad with a certain character

Getting input from the user

- Use streams also to get input from console
- Use the `cin` stream
  Ex:
  ```
  int value;
  cin >> value;
  ```
- If you want to read an entire line, use `getline`
  Ex:
  ```
  string line;
  getline(cin, line);
  cout << "The input was " << line << endl;
  ```

References

- “Constrained” and “safer” pointers
- Compare

```
int a;                int a;
int *pa = &a;         int &ra = a;
int *pa = NULL;       -
*pa = 10;             ra = 10;    => a==10
int b;                int b;
pa = &b;              -
int *pc;              -
pc = pa;              -
```

Passing Arguments by Reference in C++

- 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++ the `new` and `delete` operators are used
- In C we used `malloc` and `free`
- If you allocate an array with `new` you need to delete with `delete []`
- Ex:

  ```
  int *p = new int[10];
  p[0] = 42;
  delete [] p;
  ```
- Typical mistake, forgotten []

The Object-Oriented Paradigm

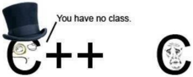
The motivation:

- We are trying to solve complex problems
  - Complex code with many functions and names
  - Difficult to keep track of all details
- How can we deal with the complexity?
  - Grouping related things
  - Abstracting things away
  - Creating hierarchies of things
- This also improves:
  - Code re-use
  - Reliability and debugging

Key Concepts of OOP

- Classes (types)
- Instances (objects)
- Methods (Actions)
- Encapsulation : Data manipulation
- Inheritance : Code re-use
- Polymorphism : Code efficiency

Classes



- A `class` is an “extension” of a `struct` : define new data types
- A class can have both data member and function members (methods)
- Classes bring together data and operations related to that data
- Classes works like a namespace

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` and
- `private` are access specifiers

Objects

- Objects are instances of classes
- Objects correspond to variables of type Class
- Create a new instance of a class:

  ```
  ClassName variableName;
  ```

Constructor

- Constructor is a special kind of method to initialize data members of the class (objects)
- *default constructor* (without arguments) + multiple user-defined constructors
- The constructor has the same name as the class and has no return type

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

Constructor Example

- What is the output?

-
  ```
  class A {
  public:
    int m_X;
    int m_Y;
  public:
    A() { m_X=5; m_Y=5; }
    A(int a) { m_X=a; m_Y=a; }
    A(int a,int b) { m_X=a; m_Y=b; }
  };
  A a,aa(10),aaa(10,20);
  std::cout << a.m_X << aa.m_X <<aaa.m_Y
  std::endl;
  ```

Constructor Example

- What is the output now?
-
```
class A {
public:
  int m_X = 40;
  int m_Y = 40;
public:
  A() { m_X=5; m_Y=5; }
  A(int a) { m_X=a; m_Y=a; }
  A(int a,int b) { m_X=a; m_Y=b; }
};
A a,aa(10),aaa(10,20);
std::cout << a.m_X << aa.m_X <<aaa.m_Y
std::endl;
```

Lecture 14: Object Oriented Programming in C++

Wrap Up

Classes (cont'd)

More on Classes and Members

Group presentations

Destructor

- When an object is deleted the destructor is called to clean up things
- For instance, to 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
  ...
  };
  ```

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)
```

Let's look at some examples

Task 1

- 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 2

- Write a class `Complex` for a complex number
- Provide 3 constructors
  - default - which should create a complex number with value 0
  - having one argument - should create a real value
  - having two arguments - should create a complex number with real and imaginary part

Lecture 14: Object Oriented Programming in C++

Wrap Up

Classes (cont'd)

More on Classes and Members

Group presentations

`this` pointer

- Inside class methods you can refer to the object with `this` pointer
- The `this` pointer cannot be assigned (done automatically)
- Example use in our toy constructor example.

const

- Can have `const` function arguments
- Ex: `void fcn(const string &s);`
- Pass the string as a reference into the function but commit to not change it
- For classes this can be used to commit to not change an object as well
- Ex: `void fcn(int arg) const;`
- The function `fcn` commits to not change anything (its class members) in the object it belongs to

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
- You need to define static data member
- Ex: (in source file) `int A::m_Counter = 0;` if `m_Counter` is a static data member of class `A`
- **Static methods can only use static data members!**

Task 3

- Start from the Complex class from last time
- Add a static int member
- Every time a new complex number is created the static variable should be incremented
- Implement the member function
  `Complex& add(const Complex &c);`
  which should add `c` to the object
- How does the number of created objects change if we change the function to
  `Complex& add(Complex c);`

Lecture 14: Object Oriented Programming in C++

- Wrap Up
- Classes (cont'd)
- More on Classes and Members
- Group presentations

Presentation today - Group 11

- How to interface between Matlab and C using MEX
- Calling Matlab functions in C program
- Calling C functions in Matlab

Presentation on 12th Oct (Monday)

- Group 10 - How to optimize C code. Explain with examples
- Group 12 - Introduce Genetic Algorithms (GA). Implement a GA solution for a problem in C++, e.g., Traveling Salesman Problem

Presentation on 14th Oct (Wednesday)

- Huffman Coding for compression
- Implement it in C++

Presentation on 15th Oct (Thursday)

- Group 14:
- Expectation-Maximization (EM) algorithm
- Monte Carlo Sampling for inference and approximation.
- Implement an example in C++
- Group 15:
- Introduce Multi-threading
- Show some implemented examples in C++

The C++ project

- Will be announced tomorrow
- Deadline: Monday 26.10.2014
- Lab/help session:
  Friday 16.10.2014, 1-3:00pm, Room 22:an, Teknikringen 14
- **Reminder: C project deadline tomorrow**