EL2310 – Scientific Programming

Lecture 14: Object Oriented Programming in C++

Andrzej Pronobis
(pronobis@kth.se)

Royal Institute of Technology – KTH

Overview

Lecture 14: Object Oriented Programming in C++

- Wrap Up
- Introduction to Object Oriented Paradigm
- Classes
- More on Classes and Members
- Operator Overloading

Last time

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

Today

- Object Oriented Programming
- Classes

Lecture 14: Object Oriented Programming in C++

Wrap Up

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 []

Lecture 14: Object Oriented Programming in C++

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
- Interfaces
- Access protection - information hiding
- Encapsulation
- Composition / aggregation
- Inheritance
- Polymorphism

Lecture 14: Object Oriented Programming in C++

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
- Unlike structs, they also define how operators work on the new 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

Access specifiers

- There are three access specifiers:
  - `public`
  - `private`
  - `protected`
- No access specifier specified ⇒ assumes it is `private`
- Data and function members that are `private` cannot be accessed from outside the class
- Ex: `m_X` above cannot be accessed from outside
- `protected` will be discussed later

C++ Structs

- C++ also uses `struct`
- In C++ `struct` is just like a class (much more than the C `struct`!)
- The only difference is the default access protection:

```
class Name {
  int m_X; // Private
};
struct Name {
  int m_X; // Public
};
```

Classes and Objects

- Classes define data types
- Objects are instances of classes
- Objects correspond to variables
- Declaring an object:

```
ClassName variableName;
```

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`

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

Constructor

- When an object of a certain class is created the so called *constructor* is called
- Constructor is a special kind of method.
- 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

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

Constructor

- Some types cannot be assigned, only initialized, e.g. references
- These data members should be initialized in the *initializer list* of the constructor
- Try to do as much of the initialization in the initialization in the list rather than using assignment in the body of the constructor
- Variables are initialized in the order they appear in the list

```
class A {
public:
  A():m_X(1) {}
private:
  int m_X;
};
```

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
  ...
  };
  ```
  ```

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

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)
```

Lecture 14: Object Oriented Programming in C++

`this` pointer

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

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 in the object it belongs to
- Can only call `const` functions from a const function or with a `const` object

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`

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++

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
- This way you can make them behave in a “proper” way for your class
- It will not change the behavior for other classes only the one which overloads the operator
- Some operators are member functions, some are defined outside class

Task 4

- Use the Complex number class from before. Overload/implement:
- `std::ostream& operator<<(std::ostream &os, const Complex &c);`
- `Complex operator+(const Complex &c1, const Complex &c2)`
- `Complex operator+(const Complex &c);` (member function)
- `Complex& operator=(const Complex &c);` (member function)

Next Time

- C Help Sessions:
  - Wednesday 13-15 Room 304
  - Thursday 13-14 Room 523
- Inheritance, Virtual Functions and Templates
- C-project deadline Thursday 4th of October