EL2310 – Scientific Programming

Lecture 14: Object Oriented Programming in C++

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

Royal Institute of Technology – KTH

Overview

Lecture 14: Object Oriented Programming in C++

- Wrap Up
- Printing
- More on getting Input
- More on Classes and Members
- More on Object Oriented Programming

Tasks

- Lecture 13
- Lecture 14

Last time

- Intro to C++
- Some differences C vs C++

Today

- Printing and Getting Input
- Static members/data
- Review on Classes
- Object Oriented Programming

Lecture 14: Object Oriented Programming in C++

- Wrap Up
- Printing
- More on getting Input
- More on Classes and Members
- More on Object Oriented Programming

Tasks

- Lecture 13
- Lecture 14

Namespace

- Namespace container for naming giving additional abstration layer
- C has a single namespace
- C++ each class defines a namespace

Namespace

- Specifying the namespace gets old,
  ```
  std::cout << "Apa" << std::endl;
  ```
- Extending a specific namespace,
- Ex.
  ```
  using namespace std
  cout << "Apa" << endl;
  ```
- Avoid in headerfiles

Lecture 14: Object Oriented Programming in C++

Wrap Up

Printing

More on getting Input

More on Classes and Members

More on Object Oriented Programming

Tasks

Lecture 13

Lecture 14

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

Formatting output

- Just like in C you can format the output in a stream
- You can use

  `width` number of characters for output to fill
  `precision` number of digits
  `fill` pad with a certain character
- Syntax:

```
cout.precision(4);
cout.width(10);
cout.fill('0');
cout << 12.3456789 << endl;
```

- Will output 0000012.35
- Default precision=6, fill=' ' (space)

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
- If you want to read an entire line you can use `getline`
- Ex:

```
string line;
getline(cin, line);
cout << ``The input was `` << line << endl;
```

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);
  cout << ``x='' << x << endl;
}
```

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

new/delete

- 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 []`
- Ex:

  ```
  int *p = new int[10];
  p[0] = 42;
  delete [] p;
  ```

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 telling that everything after it can be access from outside the object
- `private` is an access specifier telling that everything after it is hidden from outside of the class

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
- Try to do as much of the initialization in the initialization list (“colon list”) rather than using assignment in the body of the constructor
- Double work otherwise, first default initialization and then assignment
- Note that variables are initialized in the orders they appear in the class definition

Destructor

- When an objects is deleted the destructor is called
- The destructor should clean up things
- For example free up dynamically allocated memory
- There is only destructor
- If not declared a default one is used which will not free up dynamic memory
- Syntax: `~ClassName();`
- Ex:

```
Class A {
public:
  A(); // Constructor
  ~A(); // Destructor
...
};
```

this pointer

- Inside an object you can refer to the object with the 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
- That is all instantiated objects share the same `static` member
- You can use a `static` without instantiating an 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`

Lecture 14: Object Oriented Programming in C++

- Wrap Up
- Printing
- More on getting Input
- More on Classes and Members
- More on Object Oriented Programming

Tasks

- Lecture 13
- Lecture 14

Object Oriented Programming (OOP)

- Encapsulation
  - Bundle data and the code to process it
  - Can create a “black-box” with well defined interface
  - Hiding the inside means you can not change the inside
  - this bundle or box is the *object*
- Polymorphism
  - “one interface, multiple methods”
  - Can have the same interface for many classes that do the same thing

Object Oriented Programming (OOP)

- Encapsulation
  - Bundle data and the code to process it
  - Can create a “black-box” with well defined interface
  - Hiding the inside means you can not change the inside
  - this bundle or box is the *object*
- Polymorphism
  - “one interface, multiple methods”
  - Can have the same interface for many classes that do the same thing

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

Operator overloading

- You can overload most operator
- This way you can make them behave in a certain way for a certain class
- It will not change the behavior for other classes only the new you add definition for

Inheritance

- Inheritance is a way to show a relation like “is a”
- Ex: A car is a vehicle
- A car inherits many of its properties from being a vehicle
- These same properties could also be inherited by a truck or a bus
- Syntax: `class Car : public Vehicle` to tell that Car inherits from Vehicle

Inheritance vs Aggregation

- Inheritance correspond to “is a” relations
- Ex:
  ```
  class Car :  public Vehicle ...
  ```
- Aggregation to “has a”
- Ex:
  ```
  class Car {
  ...
  Person m_Owner;
  ```

Inheritance and Constructors

- If you have three classes A, B and C,
- where B inherits from A and C from B
- When you create C the constructor from the base classes (B and A) will be run first
- Execution order
  1. Initialization list for A runs
  2. Body of A constructor runs
  3. Initialization list for B runs
  4. Body of B constructor runs
  5. Initialization list for C runs
  6. Body of C constructor runs

Constructors

- If you do not specify a constructor in the initialization list the default constructor will be called

Lecture 14: Object Oriented Programming in C++

- Wrap Up
- Printing
- More on getting Input
- More on Classes and Members
- More on Object Oriented Programming

Tasks

- Lecture 13
- Lecture 14

Task 13.4

- 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

Task 1

- Create a class hierarchy with Vehicle as base class and subclasses Car and Motorcycle
- What belongs in the base class and what goes into the subclasses?

Task 2

- 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);`
- Also look at the functions
  - `Complex add(const Complex &c1, const Complex &c2);`
  - `Complex add(Complex c1, Complex c2);`
- What is the difference between the functions?

Task 3

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

Next Time

- C Help Session: Today 15-16 Room 304
- Lecture: Wednesday 10th of October, 15-17, D34
- Inheritance, Virtual Functions and Templates
- C-project deadline Thursday 6th of October