EL2310 – Scientific Programming

Lecture 15: Inheritance and Polymorphism, STL

Andrzej Pronobis
(pronobis@kth.se)

Royal Institute of Technology – KTH

Overview

Lecture 15: Inheritance and Polymorphism, STL

- Wrap Up
- Additional Bits about Classes
- Overloading
- Inheritance
- Polymorphism and Virtual Functions
- The Standard Template Library (STL)

Wrap up of Course

Last time

- Classes, classes, classes

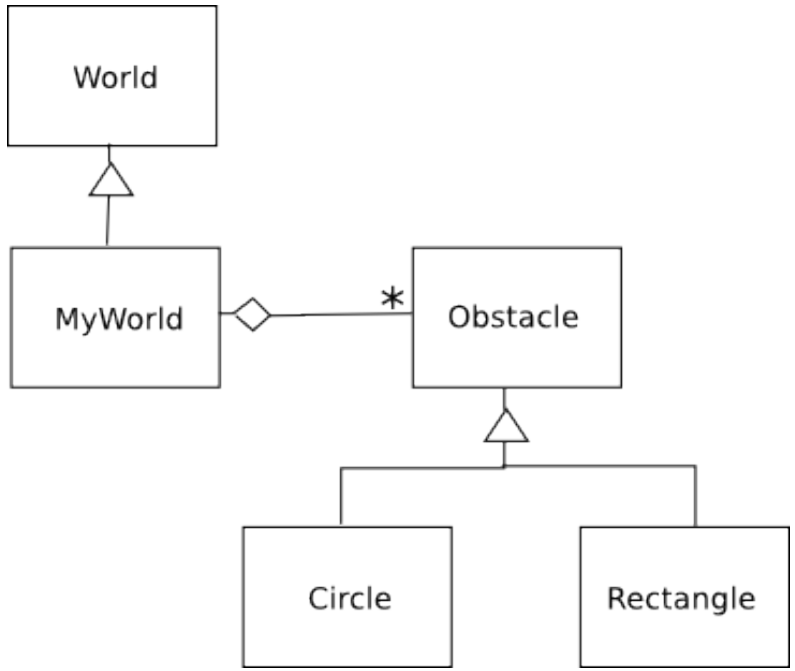
Today

- Even more about classes
- Function / operator overloading
- Inheritance
- Virtual functions
- STL

Repetition

- **Namespaces**
- **Call by Reference**: &
- **Classes**: “extension” of structs
  - data, functions
  - constructor, destructor
- **C++ Structs**
- Classes and **source/header files**

Class diagrams / UML



Dynamic allocation of objects

- One reason to use dynamic memory allocation (`new/delete`):
  - Moving around pointers to BIG chunks of memory (avoiding unnecessary copying)
- Makes sense not only for arrays
- Objects can also be BIG (e.g. database object can be 500MB!)
- Typically, we dynamically allocate objects
- We free memory when the object is no longer needed
- We pass objects by reference (* or &) to functions
- Example:

```
Database db = new Database("mydatabase.db");
useDb(db); // void useDb(Database *db)
delete db;
db = NULL;
```

`this` pointer

- Inside class methods you can refer to the object with `this` pointer
- The `this` pointer cannot be assigned (done automatically)
- Useful when we want to pass “this” class by reference:
- Example:

```
class Database
{
  void myMethod()
  {
    useDb(this); // void useDb(Database *db)
  }
}
```

`const` arguments

- Function arguments can be `const`
  Ex: `void fcn(const string &s);`
- String `s` is passed by reference and cannot be changed!

`const` methods

- Use `const` to state that a method doesn't change the object

```
class Database
{
  void fcn1(int arg) const;
  void fcn2(int arg);
}
```

- `fcn1` is not allowed to change anything in the object
- We can only call `const` methods from a const method or on a `const` object
- Example:

```
const Database d;
d.fcn1(1); // Ok, fcn1 is const
d.fcn2(1); // Error!, fcn2 is not const
```

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!**

Function overloading

- We can create functions and methods with the same name, but different arguments
- It is not possible to overload by changing return type
- Example:

```
void method();
void method(int a);
void method(int b, double c);
void method(int b); WRONG!
int method(int b); WRONG!
```

Operator overloading

- Operators behave just like functions
- Compare

```
Complex& add(const Complex &c);
Complex& operator+=(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 1

- 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)

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 be inherited by a truck or a bus
- Syntax:
  `class Car : public Vehicle`
  specifies that Car inherits from Vehicle

Inheritance and Constructors

- If you have three classes A, B and C,
- where
  - B inherits from A (`class B: public A`)
  - C inherits from B (`class C: public B`)
- When you create C:

  `C c;`

  the constructor from the base classes (B and A) will be run first
- Execution order
  1. Constructor of A
  2. Constructor of B
  3. Constructor of C

Access specifiers

- `private:` can be accessed from:
  - inside of the class
- `public:` can be accessed from:
  - inside of the class
  - subclasses
  - outside of the class
- `protected:` can be accessed from:
  - inside of the class
  - subclasses

Polymorphism

- A variable/function can have more than one form
- Example of polymorphism: operator/function overloading
- We can have sub-type polymorphism:
  a variable can be of more than one form
- A variable of a base type can hold an object of a sub-type
- In C++ implemented using references or pointers to base classes

Polymorphism example

- ```
  class Vehicle
  {...}
  class Car:  public Vehicle
  {...}
  ```
- `Vehicle *v1 = new Vehicle();`
- `Vehicle *v2 = new Car();`
- `v2` is a Car hidden inside a variable of type pointer to Vehicle!
- We can then write: `v1 = new Car();`
- So, `v1` can hold both a Car and a Vehicle (or even a Truck!) **Polymorphism!**
  ```

Subclasses as arguments to function

- If a function requires as argument a pointer/reference to an object of class `A`
- We can provide a pointer/reference to any subclass of `A`

Accessing methods

- ```
  class Vehicle
  {
    void drive();
  }
  class Car:  public Vehicle
  {
    void openTrunk();
  }
  ```
- `Vehicle *v = new Car();`
- `v->drive();` runs drive() from the Vehicle part of the Car
- `v->openTrunk();` NOT POSSIBLE!
- But: `((Car *)v)->openTunk();` WORKS!
  ```

Overloading in sub-classes

- We can overload a method in a sub-class

```
class Vehicle {
  void drive();
}
class Car:  public Vehicle {
  void drive();
}
```

- `Vehicle *v1 = new Vehicle();`
- `Vehicle *v2 = new Car();`
- `Car *c = new Car();`
- `v1->drive();` and `v2->drive();` run drive() from the Vehicle
- `c->drive();` runs drive() from the Car

`virtual` functions

- What if we want the object know what it "really" is and run the correct `drive()` method?
- Declare the method with the keyword `virtual`

```
class Vehicle {
  virtual void drive();
}
class Car:  public Vehicle {
  virtual void drive();
}
```

- `Vehicle *v1 = new Vehicle();`
- `Vehicle *v2 = new Car();`
- `v1->drive();` runs drive() from the Vehicle
- `v2->drive();` runs drive() from the Car

Polymorphism with `virtual` functions

- What `virtual` function to run is determined at run-time
- Depends on the “real” type of objects
- Works for both pointers and references

Template

- Templates offers a way to write code agnostic to data types
- You write a template for the code
- Compiler generates a version for each data type you use it with
- We can write both template classes and functions
- Example of a template function:

```
template <class T>
T getMax (T a, T b)
{
  if(a>b) {return a}
  else {return b}
}
```

- `getMax<int>(4,5)` returns 5
- `getMax<double>(4.2, 4.1)` returns 4.2

Standard Template Library: STL

- The Standard Template Library (STL) provides classes for:
  - Collections: lists, vectors, sets, maps
- Defined as templates: can store data of any type!
- Examples:
  - `std::list<T>`
    Ex: `std::list<std::string> names;`
  - `std::vector<T>`
    Ex: `std::vector<double> values;`
  - `std::set<T>`
    Ex: `std::set<std::string> nameOfPerson;`
  - `std::map<T1,T2>`
    Ex: `std::map<int, std::string> nameOfMonth;`
    Ex: `std::map<std::string, int> monthNumberByName;`

Standard Template Library: STL

- Different collections are optimized for different use, e.g.:
  - `std::list<T>`
    Cannot access elements with `x[i]`, need to use so called *iterators* to step through the list, can add/remove elements at low cost
  - `std::vector<T>`
    Can access elements with `x[i]`, but resizing is more costly
  - `std::set<T>`
    Does not allow for redundant elements
  - `std::map<T1,T2>`
    Provides a mapping from one object to another
- More in C++ Library Reference, e.g.
  `http://www.cplusplus.com/reference/stl/list/`

Lecture 15: Inheritance and Polymorphism, STL

Wrap up of Course

MatLab - What you should have learned:

- Be comfortable working with MatLab
- Preparing scripts and functions using basic elements of programming (loops, branching, ...)
- Taking advantage of in-built functions (load data, plot data), especially the visualization capabilities.
- Translating a mathematical problem into MatLab code.

- Understand MatLab code if you see it.
- Know when (and how) to use MatLab in another course.

MatLab - What you should have learned:

- Be comfortable working with MatLab
- Preparing scripts and functions using basic elements of programming (loops, branching, ...)
- Taking advantage of in-built functions (load data, plot data), especially the visualization capabilities.
- Translating a mathematical problem into MatLab code.
- Understand MatLab code if you see it.
- Know when (and how) to use MatLab in another course.

MatLab - What you should have learned:

- Be comfortable working with MatLab
- Preparing scripts and functions using basic elements of programming (loops, branching, ...)
- Taking advantage of in-built functions (load data, plot data), especially the visualization capabilities.
- Translating a mathematical problem into MatLab code.

- Understand MatLab code if you see it.
- Know when (and how) to use MatLab in another course.

MatLab - What you should have learned:

- Be comfortable working with MatLab
- Preparing scripts and functions using basic elements of programming (loops, branching, ...)
- Taking advantage of in-built functions (load data, plot data), especially the visualization capabilities.
- Translating a mathematical problem into MatLab code.

- Understand MatLab code if you see it.
- Know when (and how) to use MatLab in another course.

C - What you should have learned:

- Working with C: how to write, compile, link, execute.
- Declaring and initializing variables, basic data types, pointers(!), memory allocation(!)...
- Preparing programs using basic elements of programming (loops, branching, ...)
- Taking advantage of libraries (e.g. for printing data)
- Understand C code if you see it.
- Know when (and how) to use C in another course.

C - What you should have learned:

- Working with C: how to write, compile, link, execute.
- Declaring and initializing variables, basic data types, pointers(!), memory allocation(!)...
- Preparing programs using basic elements of programming (loops, branching, ...)
- Taking advantage of libraries (e.g. for printing data)
- Understand C code if you see it.
- Know when (and how) to use C in another course.

C - What you should have learned:

- Working with C: how to write, compile, link, execute.
- Declaring and initializing variables, basic data types, pointers(!), memory allocation(!)...
- Preparing programs using basic elements of programming (loops, branching, ...)
- Taking advantage of libraries (e.g. for printing data)
- Understand C code if you see it.
- Know when (and how) to use C in another course.

C - What you should have learned:

- Working with C: how to write, compile, link, execute.
- Declaring and initializing variables, basic data types, pointers(!), memory allocation(!)...
- Preparing programs using basic elements of programming (loops, branching, ...)
- Taking advantage of libraries (e.g. for printing data)
- Understand C code if you see it.
- Know when (and how) to use C in another course.

C++ - What you should have learned:

- What of C you can use in C++ and what C++ has to offer more (or in a different way) ...
- especially, the Object Oriented Programming Paradigm(!): Encapsulation, Polymorphism, Inheritance.
- Declaring classes and instantiating objects, accessing members, ...
- Understanding of 'conceptual programming', i.e. hiding of functions, declaring of static, const, virtual ...
- Understand C++ code if you see it.
- Know when (and how) to use C++ in another course.

C++ - What you should have learned:

- What of C you can use in C++ and what C++ has to offer more (or in a different way) ...
- especially, the Object Oriented Programming Paradigm(!): Encapsulation, Polymorphism, Inheritance.
- Declaring classes and instantiating objects, accessing members, ...
- Understanding of 'conceptual programming', i.e. hiding of functions, declaring of static, const, virtual ...
- Understand C++ code if you see it.
- Know when (and how) to use C++ in another course.

C++ - What you should have learned:

- What of C you can use in C++ and what C++ has to offer more (or in a different way) ...
- especially, the Object Oriented Programming Paradigm(!): Encapsulation, Polymorphism, Inheritance.
- Declaring classes and instantiating objects, accessing members, ...
- Understanding of 'conceptual programming', i.e. hiding of functions, declaring of static, const, virtual ...
- Understand C++ code if you see it.
- Know when (and how) to use C++ in another course.

C++ - What you should have learned:

- What of C you can use in C++ and what C++ has to offer more (or in a different way) ...
- especially, the Object Oriented Programming Paradigm(!): Encapsulation, Polymorphism, Inheritance.
- Declaring classes and instantiating objects, accessing members, ...
- Understanding of 'conceptual programming', i.e. hiding of functions, declaring of static, const, virtual ...
- Understand C++ code if you see it.
- Know when (and how) to use C++ in another course.

C++ - What you should have learned:

- What of C you can use in C++ and what C++ has to offer more (or in a different way) ...
- especially, the Object Oriented Programming Paradigm(!): Encapsulation, Polymorphism, Inheritance.
- Declaring classes and instantiating objects, accessing members, ...
- Understanding of 'conceptual programming', i.e. hiding of functions, declaring of static, const, virtual ...
- Understand C++ code if you see it.
- Know when (and how) to use C++ in another course.

In general:

- Have an understanding for basic concepts in programming
- Be skilled enough using MATLAB, so it does not pose a problem in other courses
- Solve problems and implement algorithms in C and C++
- Be able to read and understand existing code written in C or C++
- Know the importance of writing code which others can understand, change, correct and build upon

In general:

- Have an understanding for basic concepts in programming
- Be skilled enough using MATLAB, so it does not pose a problem in other courses
- Solve problems and implement algorithms in C and C++
- Be able to read and understand existing code written in C or C++
- Know the importance of writing code which others can understand, change, correct and build upon

In general:

- Have an understanding for basic concepts in programming
- Be skilled enough using MATLAB, so it does not pose a problem in other courses
- Solve problems and implement algorithms in C and C++
- Be able to read and understand existing code written in C or C++
- Know the importance of writing code which others can understand, change, correct and build upon

In general:

- Have an understanding for basic concepts in programming
- Be skilled enough using MATLAB, so it does not pose a problem in other courses
- Solve problems and implement algorithms in C and C++
- Be able to read and understand existing code written in C or C++
- Know the importance of writing code which others can understand, change, correct and build upon

In general:

- Have an understanding for basic concepts in programming
- Be skilled enough using MATLAB, so it does not pose a problem in other courses
- Solve problems and implement algorithms in C and C++
- Be able to read and understand existing code written in C or C++
- Know the importance of writing code which others can understand, change, correct and build upon

Summary

We have tools, but we haven't done much Computer Science yet

- Algorithms: *Sorting*, *Mapping*, ...
- Data structures: *Trees*, *Graphs*, ...
- Complexity
- Discrete Math
- ...

How to continue?

- The aim of this course was to get you started
- Hundreds of references and books
- You should know where to search for info when you need it
- Some more concentrated programming courses at KTH:
  - **DD2385** Programutvecklingsteknik
  - **DD2387** Programsystemkonstruktion med C++
  - **DD2390** Internet programming
  - **DD2257** Visualization
- Experience(!) - your own project.

Still to do:

- Your Evaluation
  - Finish C/C++ projects
  - Final submission deadline is next period
  - The course is only pass or fail
- Our Evaluation
  - Will be available through BILDA after the C++ project
  - For collecting feedback and opinions about the course

Still to do:

- Your Evaluation
  - Finish C/C++ projects
  - Final submission deadline is next period
  - The course is only pass or fail
- Our Evaluation
  - Will be available through BILDA after the C++ project
  - For collecting feedback and opinions about the course

Next Time

- Invited lecture!