2D1358 Object Oriented Program Construction in C++

Lecturer:
Frank Hoffmann hoffmann@nada.kth.se

Assistants:
Danica Kragic dank@nada.kth.se
Anders Oreback oreback@nada.kth.se
Peter Raicevic peter@nada.kth.se
Mikael Rosbacke rosbacke@nada.kth.se

Course webpages:
general : http://www.nada.kth.se/kurser/kth/2d1358
period 4 : http://www.nada.kth.se/kurser/kth/2d1358/00-01/F

Exercises & Labs

• group 1: Danica Kragic (in English)
• group 2: Anders Oreback
• group 3: Peter Raicevic
• group 4: Mikael Rosbacke
• First exercises on Tuesday 20/3 11:00
  • group 1 : room D32 (notice room change)
  • group 2 : room D33 (notice room change)
  • group 3 : room E36 (will be held by Frank Hoffmann until Tuesday 3/4)
  • group 4 : starts Monday 26/3 (first week attend either of the groups 1-3)
• lab groups 1-4, group 1-2 start Thursday 22/3 17:00
  group 3-4 start Friday 23/3 10:00

Course Literature

• C++ Programming Language, 3rd edition, Bjarne Stroustrup, Addison-Wesley
• C++ Primer, 3rd edition, Stanley B. Lippman, Josee Lajoie, Addison-Wesley
• The Waite's Group's Object-Oriented Programming in C++, 3rd edition, Robert Lafore, SAMS

Object Oriented Design and UML
• Design Patterns, Erich Gamma, Addison-Wesley
• The Unified Modeling Language User Guide, Grady Booch, Addison-Wesley
• UML Distilled, Martin Fowler, Addison-Wesley

Course Registration / Accounts

• Obtaining a UNIX account at NADA
  • sign up on the list with first and last name
  • visit DELPHI to create your account
    http://www.sgr.nada.kth.se/delfi
• Register for the course with RES
  • UNIX command : res checkin oopk01
  or
  • Via the web-form on the course webpage
**Course Requirements**

- regular attendance of lectures
- active participation in the exercises
- 7 lab exams
  - lab1: Matrices (optional but highly recommended)
  - lab2: Menu system (mandatory) : due 06.04.01
  - lab3: Drawing program (mandatory) : due 11.05.01
  - lab4: "Fia" Game (mandatory) : due 25.05.01
  - lab5: Handling options (4 points) : due 01.06.01
  - lab6: Window library (4 points) : due 01.06.01
  - lab7: Adventure Game (5 points) : due 01.06.01
- if you fail a lab you can redo it within one week (only if you took the lab exam before the deadline)
- if you miss a lab exam or fail twice you cannot take it earlier than the next "omtentamenperiod" in August 01

---

**Procedural Languages**

- Examples of procedural languages: C, Pascal, Fortran
- A program in a procedural language is basically a list of instructions
- As programs become larger they are usually broken down into smaller units, such as functions, procedures, subroutines
- Functions can be grouped together into modules according to their functionality, objectives and tasks.
- Structured programming is a programming paradigm that to a large extent relies on the idea of dividing a program into functions and modules.

---

**Problems with Structured Programming**

- Functions have unrestricted access to global data
  - Function A: local data
  - Function B: local data
  - Function C: local data
  - global data Y
  - global data Z

- Large number of potential connections between functions and data (everything is related to everything, no clear boundaries)
- makes it difficult to conceptualize program structure
- makes it difficult to modify and maintain the program
  - e.g.: it is difficult to tell which functions access the data

---

**Problems with Structured Programming**

- Data and function are considered as two separate entities
- makes it difficult to model things in the real world
- complex real world objects have both attributes and behaviours
  - attributes
    - people: name, date of birth, eye color, job title
    - cars: horse power, number of doors, color
  - behaviours
    - people: ask a person to bring you a beer
    - cars: apply the brakes, turn on the engine
- attributes and behaviors alone are sufficient to realistically model real world objects but a unified view is needed
Object Oriented Approach

- Encapsulation: integrate data and functions into one object
- Data hiding: data is hidden to the outside world and can only be accessed via the functions
- In C++ functions are called membership functions in other languages such as Smalltalk they are called methods
- Data items are called attributes or instance variables

Abstraction

- An abstraction is a named collection of attributes and behavior relevant to model a given entity for some particular purpose

Separation

- Independent specification of a visible interface and a hidden implementation
- Interface is some kind of contract between the object and the user of this object or module
- Separation is not restricted to object-oriented programming for example header files in standard C can be regarded as interfaces
**Structure of an Object**

<table>
<thead>
<tr>
<th>Interface</th>
<th>Implementation</th>
</tr>
</thead>
<tbody>
<tr>
<td>method</td>
<td>code</td>
</tr>
<tr>
<td>method</td>
<td>code</td>
</tr>
<tr>
<td>method</td>
<td>code</td>
</tr>
<tr>
<td>method</td>
<td>code</td>
</tr>
</tbody>
</table>

**Examples of Objects**

- physical objects
  - vehicles in a traffic-flow simulation
- electrical components in a circuit-design program
- elements of a computer user environment
- menus
  - graphic objects
- data-storage constructs
- arrays
- linked lists
- human entities
  - employees
  - students
- collections of data
  - an inventory
  - an address book
- user defined data types
  - time
  - complex numbers

**Example of a Class in C++**

```cpp
class someobject    // declares a class
{
    private:
        int somedata;    // class data
    public:
        void setdata(int d)    // membership function to set data
        { somedata=d; }
    int getdata()    // membership function to get data
        { return somedata; }
};
```

**Classes versus Objects**

- A class is a prototype specification from which one can generate a number of similar objects.
- A class can be considered as an object factory.
- An object is said to be a member or instance of a class.
- A class can be considered as a more complex data structure than an ordinary built-in data type.
- Standard C already knows the struct command for user-defined data types:
  ```
  struct complex
  {
      double re;
      double im;
  };
  complex x;
  ```
### Instantiation of Objects

**Class**

- **person**
  - Data: name, p_number, address, date of birth
  - Methods: getAge(), changeAddress(newAddress)

**Objects**

- **person**
  - Data: Erik Olsson, 780605-4789, Hamngatan 3, male
- **person**
  - Data: Lena Brat, 761203-7111, Stureplan 4, female
- **person**
  - Data: Lars Backe, 671110-A562, Mälartorget 19, male

### Relationships among Objects

- **Attribute:**
  One object uses another object as an attribute, namely as member data, for example a Person contains an attribute Name. This type of relationship is also called a weak association or has-a relationship. Example: A Person has a Name.

- **Association:**
  One object uses another to help it carry out a task. Classes that collaborate are usually related through associations. This type of relationship is also called a uses relationship. Example: The object Driver invokes the method Brake of the object BrakingSystem.

- **Aggregation:**
  Aggregation means that one object contains other objects. Aggregation is also called part-of relationship. Example: The class Addressbook contains many People Objects.

- **Composition:**
  Composition is building objects from parts. It is a stronger type of aggregation in which the parts are necessary to the whole, namely they are permanently bound to the object and do not exist outside the object. Example: A class Processor contains a CPU, Memory and I/O-Ports.

- **Generalization:**
  Generalization is a relationship defined at the class level not the object level, which means that all objects of this class must obey the relationship. This is type of relationship is also called a is-a-kind-of relationship. Generalization in object oriented languages is realized by means of inheritance. Example: A car is a kind of vehicle.
Inheritance

• In our daily lives we use the concept of classes divided into subclasses, for example vehicles are divided into cars, trucks, buses and motor cycles.

• The principle in this sort of division is that each sub-class shares some common features with the base class from which it is derived, but also has its own particular features.

• A sub-class also shares common methods with its super-class but can add its own methods or overwrite the methods of its super-class.

Terminology:
• Car is a sub-class (or derived class) of Vehicle
• Car inherits from Vehicle
• Car is a specialization of Vehicle
• Vehicle is a super-class (or base class) of Car
• Vehicle is a generalization of Car

• In C++ an object of a sub-class is substitutable for an object of the super-class, in other words an object of class Car can be used whenever an object of class Vehicle is required.

Reusability

• Reusability means that a class that has been designed, created and debugged once can be distributed to other programmers for use in their own programs.

• Similar to the idea of a library of functions in a procedural language.

• The concept of inheritance provides an important extension to the idea of reusability, as it allows a programmer to take an existing class without modifying it and adding additional features and functionality. This is done by inheriting a new sub-class from the existing base class.
Polymorphism & Overloading

- Polymorphism: using functions and operators in different ways, depending on what they are operating on.
- Polymorphism allows it to manipulate objects without knowing their exact type but only their common property. For example, the classes Triangle and Circle both have their own (polymorphic) version of the method Draw, but a graphic routine that draws graphical elements does not have to know which object it manipulates.
- Overloading: an existing operator, such as + or = is given the capability to operate on a new data type, for example define the operator + for the class Complex such that it realizes the addition of two complex numbers.

Polymorphism

- Polymorphism means “having many shapes”
- In C++, it refers to a situation in which an object could have any of several types
- A polymorphic variable can refer to objects of different classes, for example, a graphic object can be either a circle or a triangle
- A polymorphic function or operator can take arguments of different types
- Example:
  - int max(int a, int b);
  - double max(double a, double b);

Polymorphism and Dynamic Binding

- Suppose we have a class Shape for graphical objects with subclasses Circle and Triangle
- Circle and Triangle both provide their own polymorphic method void Draw() to draw the shape on the screen
- The term dynamic binding refers to the process of identifying at run time what code should be executed as a result of a message
  ```
  Circle c;
  Triangle t;
  Shape &s;
  s=c;
  s.Draw(); // bounded to Circle Draw()
  s=t;
  s.Draw(); // bounded to Triangle Draw()
  ```

C++ and C

- C++ is derived from the language C
- C++ is a superset of C, that means almost every correct statement in C is also correct in C++
- The most important elements added to C are concerned with classes, objects and object-oriented programming
- New features of C++
  - Improved approach to input/output
  - Standard template library (STL) container classes for vectors, lists, maps, etc.
  - Reference type replaces pointers
  - Const variables replaces #define statements
  - String data type replaces C-style strings char[]
  - New comment style augments C-style comments /* */