Object Oriented Design and C++

Richard Bowden
CVSSP
http://www.ee.surrey.ac.uk/Personal/R.Bowden
r.bowden@surrey.ac.uk

Objectives

- Objectives
  - Understand object-oriented program design techniques
  - Implementation of OOD in C++
- Prerequisites
  - C Programming
  - Experience of UNIX environment

- Course Structure
  - Introduction to Object-Oriented Programming
  - Revision of C
  - OOD in C++
- Assessment
  - C++ Assignment (30%)
    - Design, Implementation and Documentation
    - Set in week 5
    - Design due week 8 - 4pm Mon 24/11/2014
    - Final submission week 11 - 4pm Mon 15/12/2014
    - Tutorials 5-6pm weeks 7-10
  - Examination (70%)
    - Section A - 20 Compulsory Multiple Choice Questions
    - Section B - Choose 2 out of 3 questions

- Lectures
  - Tues 11-1 22AA04
- Lab Sessions
  - Friday 2-4 Starting week 2
    - 32BB03
    - 34BB04
    - 34aBB04
    - 34BB03
- Webpage

`http://www.ee.surrey.ac.uk/Personal/R.Bowden/cpp`

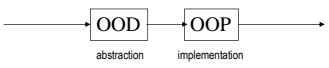
All notes, supplementary material and details of laboratory exercises are on the web page

Books

- C++
  - Lippman (good introduction)
    - C++ Primer
  - Stroustrup (good reference)
    - The C++ Programming Language
- OOD
  - Booch (background reading)
    - Object-Oriented Design
- Sams Teach yourself C++ in 21 days
  - Plenty of them cheap on ebay
- Webpage
  - http://www.ee.surrey.ac.uk/Personal/R.Bowden/cpp
- Online resources:
  - http://www.cplusplus.com/

Introduction to OOD

- Object Oriented Design (OOD)
  - Does not rely on any particular programming language
- Object Orientated Programming
  - Uses OOD to develop a program



OOD - Object Oriented Design

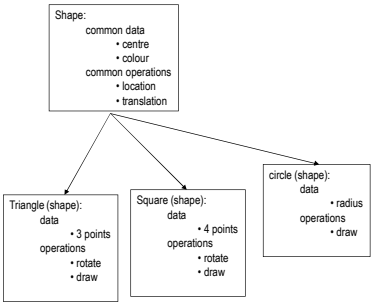
- Aims to produce a direct representation of programmers ideas

'design program as a set of interacting objects rather than a bunch of data structures with functions to process'

- Why OOD ?
  - Better management of large software projects
  - Development of software by multiple people
  - Reuse of code (efficient development and fewer bugs)
- Feature of OOD
  - Objects containing both data and operations
  - Object is accessed via a 'simple' interface
  - Hierarchical structure which inherit common properties

Example of OOD: Shapes

- Base Object
- Derived Objects



Thinking about programming

- 3 stages
  1. **Analysis**: Clear understanding of the problem
  2. **Design**: Identify key concepts involved in solution
  3. **Programming**: Express the solution in a program
- Problem and concepts are often not clearly understood before programming
  - Results in badly structured code which is difficult to modify, maintain and use.
  - Critical for large programs with multiple users

Thinking about programming

- **Key to writing good programs**:
  - Design objects (classes) that clearly represent a single concept
- **Focus on questions:**
  - How are objects created?
  - Can objects be copied/destroyed?
  - What operations are applied to an object?
  - How does the object interface to other objects?

Generates a clear concept of the object

Simple Example of OOD

- **Problem**:
  - Generation of exam marks for each student on the OOD course (exam mark + project mark)
- **Analysis**:
  - How many students ? Unknown
  - Input ? User supplies marks + n' of students
  - Output? Table of final marks

Simple Example of OOD

- **Design:**
  - Storage of exam marks for each student
    - Option 1: Allocate a 'big' array that is always bigger than the number of students
      - make sure I don't access outside the array
    - Option 2: Dynamically allocate an array to size
      - remember to delete
      - make sure I don't access outside the array
    - Option 3: Design an 'object' which is a dynamic array & manages creation/deletion/access
      - Simple user interface for array
      - safe memory management
- **Program:**
  - Use simple dynamic array object + input and output

Programming Paradigms

- Procedural
  - separate data and operations
- Modular
  - Organise data into modules
- Data Abstraction
  - Modules with associated operations
- Object Oriented
  - encapsulation of data and operations
  - hierarchical
- Generic
  - Common operations between different data

Procedural Programming

- original programming paradigm
- focus on the 'best algorithm' to perform an operation
- program based on performing a set of operations (functions) on the data
- pass data to function and return result
- typical of programs written in C/Fortran/Pascal

Procedural Programming

Example:

```
double square(double x)
{
  return x*x;
}

void some_function()
{
  …
  double sqr=square(2.0);
  …
}
```

Modular Programming

- data hiding
- focus on organising data
- group procedures with related data 'module'
- user code is insulated from data representation
- necessary for large programs
- typical of programs in Modula2

Modular Programming

Example:

```
stack {
   data: set of data items on stack
}
operations:          stack_push(stack,double x)
                     double stack_pop(stack)

void some_function()
{
   stack_push(10);
   …
   double x=stack_pop();
   …
}
```

Data Abstraction

- modules not sufficient to express complex systems
- focus on user-defined types
- use modules to define types with associated operations
- provide a user interface for data
- user interface should be independent of internal data representation

Data Abstraction

Example:

```
stack {
    data: set of data items on stack
    operations:     construct(int size)
                    destruct()
                    push(double x)
                    double pop()
}

stack globalstack.construct(10);

void some_function()
{
    stack localstack.construct(5);
    …
    localstack.push(3);
    …
    double x=localstack.pop();
    globalstack.push(x);
    localstack.destruct();
}
```

Object Oriented Programming

- data abstraction + object hierarchy
- focus on common relationships between sets of objects
- structured programs which reuse code for common operations between a set of objects
- object oriented languages support object hierarchies
- supported by C++/Java => classes/inheritance

Object Oriented Programming

- Design
  1 Identify objects with data and operations
  2 Identify commonality between data/operations for sets of objects
  3 Define base objects with common data/operations
  4 Define derived objects for specific data/operations

Object Oriented Programming

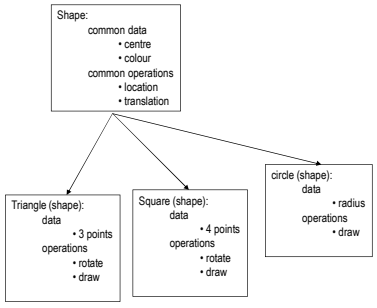
Example

```
class baseShapeC{
    private:
        point centre;
        colour col;
    public:
        baseShapeC(point p, colour c);// Constructor centre & colour
        ~baseShapeC();
        point getlocation();
        colour getColour();
        void translate(vector t);
};

class triangleC : public baseShapeC{
    private:
        point vertex1,vertex2,vertex3;
    public:
        triangleC(point v1, point v2, point v3, point p, colour c);
        void rotate(double angle);    // Rotate by angle
        void draw();
};
class circleC: public baseShapeC{
    private:
        double radius;
    public:
        circleC(double r, point p, colour c);
        void draw();
};
```

Example of OOD: Shapes

- Base Object
- Derived Objects



Generic Programming

- Objects which work for multiple types of data
- focus on expressing algorithm independent of data type
- used for containers, i.e. stack of `char/int/double` require common access functions.
- Supported by C++ as ‘templates’

Generic Programming

Example

```
template<class data> class stackC{
    private:
          data* stack;
    public:
          stackC(int size);
          void push(data);
          data pop();
};

void some_function()
{
    stackC<int> istack(5);              // stack of integers
    stackC<double> dtsack(10);          // stack of doubles
    stackC<triangleC> tstack(100);      // stack of triangles
    …
    istack.push(5);
    dstack.push(5.5);
    …
}
```

Elements of Object-Oriented Programming

- Modularity:
  - Identify 'objects; with data and operations
- Data Abstraction:
  - Define a user-interface for object independent of internal data representation
- Hierarchy:
  - Identify common operations/data between objects
    - Create hierarchy of base and derived objects
    - Base objects define common data/operations
    - Derived objects 'inherit' base object data/operations and add new data operations
- Generic
  - Identify common operations/structures for objects with different data
    - Define 'templates' for structures

The Standardisation of C++

- Although developed in the early 1980's
- Standardised in 1998
- 2003 saw some fixes
- 2005 recommended extensions to STL published in 2007
- 2011 latest major release
- 2014 some bug fixes
- Next major release planned for 2017

Year	C++ Standard	Informal name
1998	ISO/IEC 14882:1998	C++98
2003	ISO/IEC 14882:2003	C++03
2007	ISO/IEC TR 19768:2007	C++TR1
2011	ISO/IEC 14882:2011	C++11
2014	N3690 (working draft C++14)	C++14

C++03 vs C++11

- In mid-2011, the new C++ standard (dubbed C++11) was finished.
- The Boost library project made a considerable impact on the new standard
- New features included
  - regular expression support
  - a comprehensive randomization library
  - a new C++ time library
  - a standard threading library
  - a new for loop syntax (like foreach loops)
  - the auto keyword
  - new container classes……

Lets design a car database

We couldn't finish the lecture without a hello world example

- In C++ hello world looks something like this

```
#include <iostream>
using namespace std;

int main(int argc, char *argv[])
{
   cout << "Hello world!!" << endl;
}
```

NOTE, `cout` is your new best friend.
If I catch you using printf, Ill break your fingers :-)