# An Overview of C++

| | |
|---|---|
| *Course* | C/C++ Network Programming on Linux |
| *Trainer* | Tuan Le |
| *Designed by* | Tuan Le |
| *Last updated* | January 2010 |

# Contents

- Programming Concept Evolution
- Object-Oriented Programming
- Characteristics of OOPL
- Operator Overloading

# Course Objectives

- Provide the student with the fundamental knowledge and skills to become a proficient C++ programmer.
- Students should be able to use the advanced features of the C++ programming language such as:
  - To design using Inheritance, as well as being able to implement it using C++
  - Polymorphism and Encapsulation.

# Advantages

- Object-oriented programming:
- Portability:
- Brevity:
- Modular programming:
- C Compatibility:
- Speed

# Disadvantages

- A complex language
- C++ is not 100% object-oriented

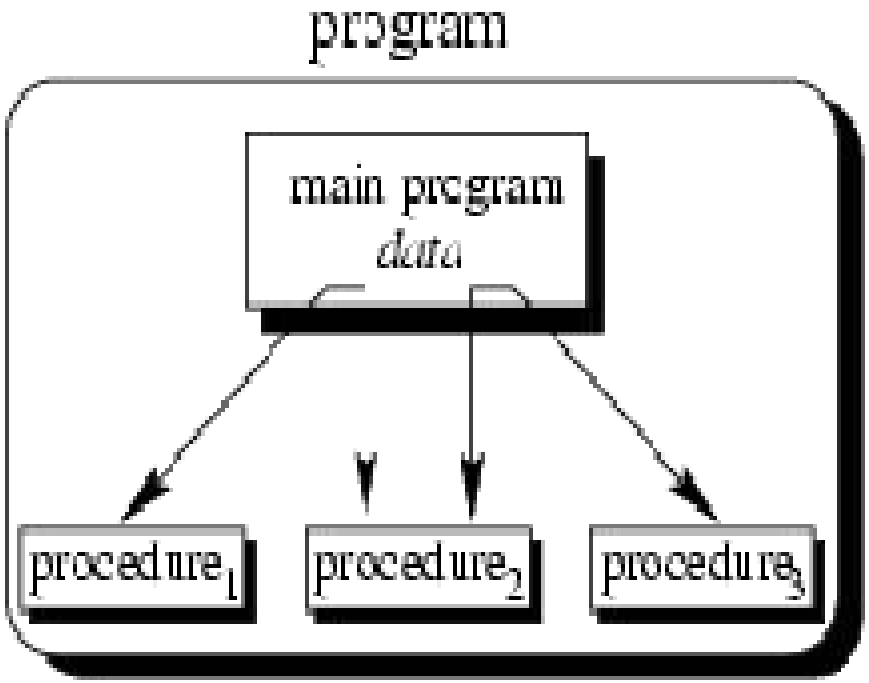
# What is Programming?

- Take
  - A *problem* (Find the area of a rectangle)
  - A set of *data* (length, width)
  - A set of *functions* (area = length * width)
- Apply ***functions*** to ***data*** to solve the ***problem***

# Programming Concept Evolution

- Unstructured
- Procedural
- Object-Oriented

# Procedural Programming



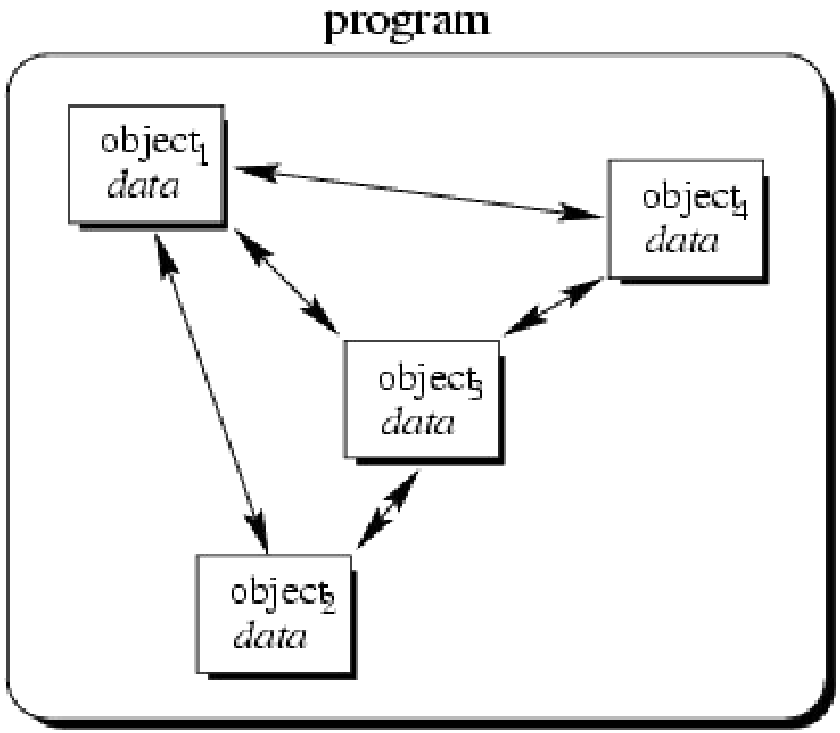

The main program coordinates calls to procedures and hands over appropriate data as parameters.

# Procedural Languages

- Usually there is a main function, which is aimed to solve the problem. In a function, arguments are passed to the subfunctions, and results are returned.
- Examples: C, Pascal, Basic, Assembly language…
- For the rectangle area problem, the procedure to solve it is:

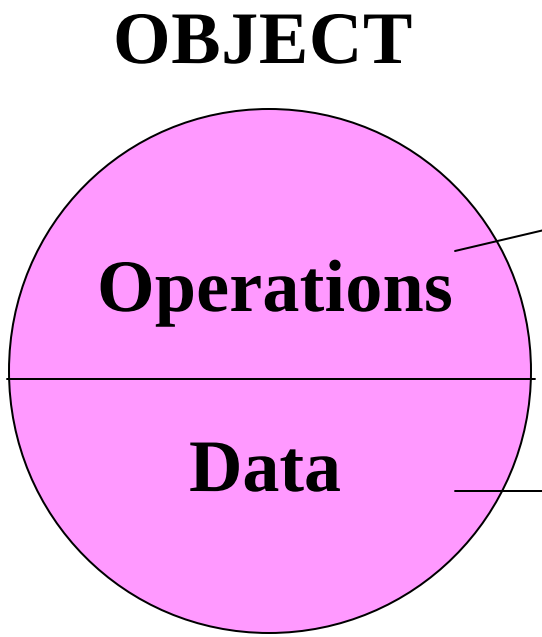
```
int compute_area (int l, int w)
{
        return ( l * w );
}
```

# Object-Oriented Programming




Objects of the program interact by sending messages to each other

# What is an object?




**set of methods**
**(member functions)**

**internal state**
**(values of private data members)**

# What is an object (cont.)

- **An object is an encapsulation of both functions and data (not one or the other individually)**
- ***Objects are the abstractions of real world entities***
  - *Classes are data structures that define common properties or attributes*
  - *Objects are instances of a class*
- ***Objects have States***
  - *Each property or attribute has a value at a particular time*
- ***Objects have Operations***
  - *associated set of operations called methods describe how to carry out operations*
- ***Objects have Messages***
  - *request an object to carry out one of its operations by sending it a message*
  - *messages are the means by which we exchange data between objects*

# Object-Oriented Programming - example

- The rectangle area problem
- Define a class: Rect
  - Data: width, length
  - Functions: compute_area()
- An object: an instance of the class Rect
  - To Solve the problem, create an object of Rect, and request this object to return the area of the rectangle

# Object-Oriented Programming - Example

```
class Rect {
      private:
         int width, length;
      public:
         Rect (int w, int l) {
                  width = w;
                  length = l;
               }
      int compute_area() {
          return width*length;
       }
}
```

```
main()
{
     Rect rect1(3,5);
    int x;
     x=rect1.compute_area();
     cout<<x<<endl;
}
```

# Characteristics of OOPL

- **Encapsulation:** Combining data structure with actions
  - Data structure: represents the properties, the states, or characteristics of objects
  - Actions: permissible behaviors that are controlled through the member functions

  Data hiding: Process of making certain data inaccessible
- **Inheritance:** Ability to derive new objects from old ones
  - permits objects of a more specific class to inherit the properties (data) and behaviors (functions) of a more general/base class
  - ability to define a hierarchical relationship between objects
- **Polymorphism:** Ability for different objects to interpret functions differently

# Encapsulation

```
class Circle
{
    private:
        int radius
    public:
         Circle(int r);

        // The area of a circle
        int compute_area();
};
```

```
class Triangle
{
    private:
        int edgea, edgeb, edgec;
    public:
         Triangle (int a, int b, int c);

        // The area of a triangle
        int compute_area();
};
```

# Abstract Types

```
class Shape
{
      public:
                Shape();

                // Calculate the area for
                // this shape
                virtual int compute_area() = 0;
};
```

# Inheritance and Polymorphism

```
class Circle : public Shape
{
    private:
        int radius;
    public:
        Circle (int r);
        int compute_area();
};
```

```
class Triangle : public Shape
{
    private:
        int edgea, edgeb, edgec;
    public:
        Triangle (int a, int b, int c);
        int compute_area();
};
```

```
int sum_area(Shape s1, Shape s2) {
        return s1.compute_area() + s2.compute_area();
// Example of polymorphism
}
```

# Virtual Functions

- Virtual Functions overcome the problem of run time object determination
- Keyword virtual instructs the compiler to use late binding and delay the object interpretation
- How ?
  - Define a virtual function in the base class. The word virtual appears only in the base class
  - If a base class declares a virtual function, it must implement that function, even if the body is empty
  - Virtual function in base class stays virtual in all the derived classes
  - It can be overridden in the derived classes
  - But, a derived class is not required to re-implement a virtual function. If it does not, the base class version is used

# Why Operator overloading

- Programmer can use some operator symbols to define special member functions of a class
- Provides convenient notations for object behaviors

# Operator overloading

```
int i, j, k;          // integers
float m, n, p;        // floats
k = i + j;
    // integer addition and assignment
p = m + n;
    // floating addition and assignment
```

The compiler overloads the + operator for built-in integer and float types by default.

We can make object operation look like individual int variable operation, using operator functions

```
Date a,b,c;
c = a + b;
```

# Operator Overloading - example (cont.)

```
class Fred {
public:
  friend Fred operator+ (const Fred& x,
       const Fred& y);
  friend Fred operator* (const Fred& x,
      const Fred& y);
  void setValue(int x){
        m_x = x;
  }
private:
      int m_x;
};
```

```
 Fred operator+ (const Fred& x, const
Fred& y) {
        Fred a;
        a.m_x = x.m_x + y.m_x;
        return  a;
 }
 Fred operator* (const Fred& x, const
Fred& y) {
        Fred a;
        a.m_x = x.m_x * y.m_x;
        return  a;
 }
int _tmain(int argc, _TCHAR* argv[]){
       Fred Fred1, Fred2;
       Fred1.setValue(2);
       Fred2.setValue(4);
       Fred Fred3 = Fred1 * Fred2;

       return 0;
}
```

# What is 'Friend'?

- Friend declarations introduce extra coupling between classes
  - Once an object is declared as a friend, it has access to all non-public members as if they were public
- Access is unidirectional
  - If B is designated as friend of A, B can access A's non-public members; A cannot access B's
- A friend function of a class is defined outside of that class's scope

# Summary

- There are many different kinds of programming paradigms, OOP is one among them.
- In OOP, programmers see the execution of the program as a collection of dialoging objects.
- The main characteristics of OOPL include encapsulation, inheritance, and polymorphism.

# Exercises

- Refer to:

  01. An overview of C++ - Lab assignment.doc

# References

- Thinking in C++
  - http://mindview.net/Books/TICPP/ThinkingInCPP2e.html
- C++ tutorial for C users
  - http://www.4p8.com/eric.brasseur/cppcen.html
- C++ Programming Tutorial
  - http://cplus.about.com/od/beginnerctutorial/l/blcplustut.htm
- Complete C++ language tutorial
  - http://www.cplusplus.com/doc/tutorial/
- Introduction to Object-Oriented Programming Using C++
  - http://www.desy.de/gna/html/cc/Tutorial/tutorial.html