Data Abstraction and Encapsulation

Data Abstraction and Encapsulation

- **Definition**: Data Encapsulation or Information Hiding is the concealing of the implementation details of a data object from the outside world.
- **Definition**: **Data Abstraction** is the **separation** between the specification of a **data object** and its **implementation**.
- **Definition**: A **data type** is a collection of **objects** and a set of **operations** that act on those objects.
- **Definition**: An **abstract data type (ADT)** is a data type that is organized in such a way that the **specification** of the objects and the **specification** of the operations on the objects **is separated from** the **representation** of the objects and the **implementation** of the operations.

Advantages of Data Abstraction and Data Encapsulation

- Simplification of software development
- Testing and Debugging
- Reusability
- Modifications to the representation of a data type

ADT Example

ADT *NaturalNumber* is

objects: An ordered sub-range of the integers starting at zero and ending at the maximum integer (MAXINT) on the computer.

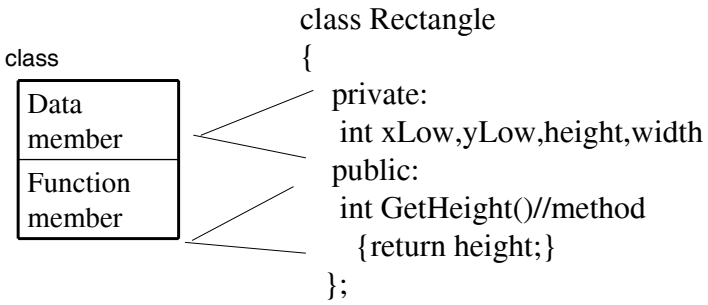
functions: for all x, y belong to *NaturalNumber;* TRUE, FALSE belong to *Boolean* and where +, -, <, ==, and = are the usual integer operations

Zero(): NaturalNumber	::=	0
IsZero(x): Boolean	::=	**if** (x == 0) *IsZero* = **TRUE** **else** *IsZero* = **FALSE**
Add(x, y): NaturalNumber	::=	if (x+y <= MAXINT) *Add* = x + y **else** *Add* = MAXINT
Equal(x, y): Boolean	::=	**if** (x == y) *Equal* = **TRUE** **else** *Equal* = **FALSE**
Successor(x): NaturalNumber	::=	**if** (x == MAXINT) *Successor* = x **else** *Successor* = x +1
Substract(x, y): NaturalNumber	::=	**if** (x < y) *Substract* = 0 **else** *Substract* = x – y

end *NaturalNumber*

ADT & C++ Class

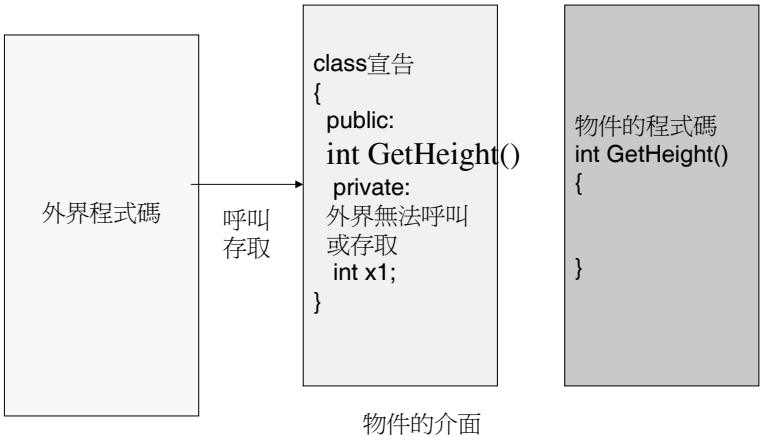
- A class name
- Data members
- Member functions (operations)



Program 2.1 Definition of the C++ class Rectangle

```
#ifndef RECTANGLE_H
#define RECTANGLE_H
// In the header file
class Rectangle {
public:                     // The following members are public
    Rectangle();            // Constructor
    ~Rectangle();           // Deconstructor
    int GetHeight();        // return the height of the rectangle
    int GetWidth(); // return the width of the rectangle
private:                    // The following members are private
    int x1, y1, h, w;
    // (x1, y1) are the coordinates of the bottom left corner of the rectangle
    // w is the width of the rectangle; h is the height of the rectangle
};
#endif
```


物件的運作



Program 2.2 Implementation of operations on Rectangle

```
// In the source file Rectangle.C
#include "Rectangle.h"

// The prefix "Rectangle::" identifies GetHeight() and GetWidth() as
    member functions belong to class Rectangle. It is required because the
    member functions are implemented outside their class definition

int Rectangle::GetHeight() {return h;}
int Rectangle::GetWidth() {return w;}
```

Constructor and Destructor

- Constructor: is a member function which **initializes** data members of an object.
  - Advantage: all class objects are well-defined as soon as they are created.
  - Must have the same name of the class
  - Must not specify a return type or a return value
- Destructor: is a member function which **deletes** data members immediately before the object disappears.
  - Must be named identical to the name of the class prefixed with a tilde ~.
  - It is invoked automatically when a class object goes out of scope or when a class object is deleted.

Examples of Constructor for Rectangle

```
Rectangle::Rectangle (int x, int y, int height, int width)
{
    x1 = x;          y1 = y;
    h = height;      w = width;
}
```

```
Rectangle::Rectangle (int x = 0, int y = 0, int height = 0, int width = 0)
: x1(x), y1(y), h(height), w(width)
{ }

Rectangle r(1, 3, 6, 6);
Rectangle *s = new Rectangle(0, 0, 3, 4);
```


Operator Overloading

- C++ can distinguish the operator == when comparing two floating point numbers and two integers. But what if you want to compare two Rectangles?

```
int Rectangle::operator==(const Rectangle &s)
{
    if (this == &s)  return 1;
    if ((x1 == s.x1) && (y1 == s.y1) && (h == s.h) && (w == s.w))   return 1;
    else return 0;
}
```


Function Overloading

- Function overloading is the practice of declaring the same function with different signatures. The same function name will be used with different number of parameters and parameters of different type.
- Example(next page)

Implement

```
class Rectangle
{
  private:
    int xLow,yLow,height,width; //data member
  public:
   Rectangle(int x, int y,int h,int w) //constructor
   {
        xLow=x;
        yLow=y;
        height=h;
        width=w;
   }
   Rectangle() //constrctor (function overloading)
   {
        xLow=0;
        yLow=0;
        height=1;
        width=1;
   }
int Rectangle::operator==(const Rectangle &s)  //operator overloading
   {
      if ((xLow == s.xLow) && (yLow == s.yLow) &&
          (height == s.height) && (width== s.width))       return 1;
     else      return 0;
   }
   public:
     int GetHeight()//method
   {return height;}
   public :
     int GetWidth()//method
   {return width;}
};
```

```
int main(void)
{
int gh, gw,gh1,gw1,te;
   string te2;
   Rectangle r(1,2,10,6);
   Rectangle r1;
   gh=r.GetHeight();
   gw=r.GetWidth();
   gh1=r1.GetHeight();
   gw1=r1.GetWidth();
   te=r==r1;
   if(te==1)
    te2="yes";
   else
   te2="no";
   cout<<"height of r is "<<gh<<endl;
   cout<<"width of r is "<<gw<<endl<<endl;
   cout<<"height of r1 is "<<gh1<<endl;
   cout<<"width of r1 is "<<gw1<<endl;
    cout<<"is the two same? "<<te2<<endl;
    system("pause");
   return 0;
}
```

Output

```
height of r is 10
width of r is 6

height of r1 is 1
width of r1 is 1
is the two same? no
```


Homework

- Implement and test the class Rectangle. Do Exercise 2.1 @P83
  - The input is (1,4,2,8) and (1,1,3,8)