2.1 Abstract Data Type and the C++ Class

Reviews of Data Type

- A ***data type*** is a collection of ***objects*** and a set of ***operations*** that act on objects
- Fundamental data type in C++:
  - char, int, float, double, …etc.
  - Modifiers: short, long, signed, unsigned
- Example: int data type
  - Objects: {0, +1, -1, +2, -2, …, MAXINT, MININT}
  - Operations: {+, -, *, /, ==, <=, …etc.}

Reviews of Data Type

- How to group data together?
- Arrays: int x[10];
  - Collection of elements of the same basic data type.
- structs (C) and classes (C++)
  - Collection of elements whose data types need not be the same.

Abstract Data Type (ADT)

- A data type that separates
  - The ***specification*** of objects from their ***representation***
  - The ***specification*** of operation from their ***implementation***.
- Major advantages
  - Simplification
  - Testing and debugging
  - Reusability
  - Flexibility

Example

Data Type 1	Data Type 2	Data Type 3
Objects: int data; Operations: int Value(void) { return data; } void Calculate(void) {data = 100; }	Objects: float data; Operations: float Value(void) { return data; } void Calculate(void) {data= exp(-10); }	Objects: Type1 data; int data2; Operations: int Value(void) { return data.Value()+data2; } void Calculate(void) {data. Calculate(); data2 = 128; }

Object Representation?

Data Type 1	Data Type 2	Data Type 3
Objects: int data; Operations: int Value(void) { return data; } void Calculate(void) {data = 100; }	Objects: float data; Operations: float Value(void) { return data; } void Calculate(void) {data= exp(-10); }	Objects: Type1 data; int data2; Operations: int Value(void) { return data.Value()+data2; } void Calculate(void) {data. Calculate(); data2 = 128; }

Object Specification?

Data Type 1	Data Type 2	Data Type 3
Objects: int data; Operations: int Value(void) { return data; } void Calculate(void) {data = 100; }	Objects: float data; Operations: float Value(void) { return data; } void Calculate(void) {data= exp(-10); }	Objects: Type1 data; int data2; Operations: int Value(void) { return data.Value()+data2; } void Calculate(void) {data. Calculate(); data2 = 128; }

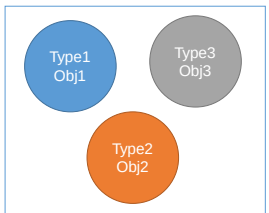
Operation Specification?

Data Type 1	Data Type 2	Data Type 3
Objects: int data; Operations: int Value(void) { return data; } void Calculate(void) {data = 100; }	Objects: float data; Operations: float Value(void) { return data; } void Calculate(void) {data= exp(-10); }	Objects: Type1 data; int data2; Operations: int Value(void) { return data.Value()+data2; } void Calculate(void) {data. Calculate(); data2 = 128; }

Operation Implementation?

Data Type 1	Data Type 2	Data Type 3
Objects: int data; Operations: int Value(void) { return data; } void Calculate(void) {data = 100; }	Objects: float data; Operations: float Value(void) { return data; } void Calculate(void) {data= exp(-10); }	Objects: Type1 data; int data2; Operations: int Value(void) { return data.Value()+data2; } void Calculate(void) {data.Calculate(); data2 = 128; }

A Software that uses ADTs




```
double Calculate1(void);
double Calculate2(void);
double Calculate3(void);
```

```
double Calculate1(void)
{ Obj1.Calculate();
  Obj2.Calculate();
  return (double)Obj1.Value()+
  (double)Obj2.Value();}

double Calculate2(void)
{ Obj2.Calculate();
  Obj3.Calculate();
  return (double)Obj2.Value()+
    (double)Obj3.Value();}
```


Introduction to C++ Class

- C++ provides new mechanism, *class*, to support ***data abstraction*** and ***encapsulation***.

```
// In the header file Rectangle.h
#ifndef RECTANGLE_H
#define RECTANGLE_H
class Rectangle {
public:    // the following members are public
           // the next four members are member functions
           Rectangle ( );        // constructor
           ~Rectangle( );        // destructor
           int GetHeight ( );    // return the height of the rectangle
           int GetWidth ( );     // return the width of the rectangle
private:   // the following members are private
           // the following members are data member
           int xLow, yLow, height, width;
           // (xLow, yLow) are the coordinates of the bottom left corner of rec.
};
#endif
```


Class Architecture

- **A class name**: (e.g., Rectangle)
- **Data members**:
  - The data that makes up the class (e.g., *xLow, yLow, height*, *width*)
- **Member functions**:
  - The set of operations that apply to the objects (e.g., *GetHeight(), GetWidth()*)
- **Levels of program access** (data encapsulation):
  - **public**: data member (function) can be accessed from anywhere in the program.
  - **private**: data member (function) can be accessed only within its class or by a **friend** class
  - **protected**: data member (function) can be accessed only within its class, by a **friend** class or from its subclass (class inheritance)

2.1.2

Data Abstraction and Encapsulation

```
// In the header file Rectangle.h
#ifndef RECTANGLE_H
#define RECTANGLE_H
class Rectangle {
public:  // the following members are public
         // the next four members are member functions
         Rectangle ( );      // constructor
         ~Rectangle( );      // destructor
         int GetHeight ( );  // return the height of the rectangle
         int GetWidth ( );   // return the width of the rectangle
private: // the following members are private
         // the following members are data member
         int xLow, yLow, height, width;
         // (xLow, yLow) are the coordinates of the bottom left corner of rec.
};
#endif
```

Data abstraction (Specification) **Implementation?**

Data encapsulation

Data Abstraction

- ***Specification*** is placed in **header** file (e.g., Rectangle.h)
- ***Implementation*** is placed in **source** file (e.g., Rectangle.cpp)

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

/* The prefix "Rectangle::" identifies GetHeight() and GetWidth() are member
function of class Rectangle. It is required because the member functions are
implemented outside the class definition*/

int Rectangle::GetHeight() {return height;}
int Rectangle::GetWidth() {return width;}
```


Class Usage

```
// In a source file main.cpp
#include <iostream>
#include "Rectangle.h"

main() {
        Rectangle r, s;     // r and s are objects of class "Rectangle"
        Rectangle *t = &s;          // t is a pointer to class object s
        .
        .
        // use "." operator to access members of class objects.
        // use "->" operator to access members of class objects through
pointers.
        If ( r.GetHigh ( ) * r.GetWidth ( ) > t→GetHeight ( ) * t→GetWidth ( ) )
           cout << "r";
        else cout << "s";
        cout << "has the greater area " << endl;
}
```

Data Encapsulation

C++

```
class  Foo{
  private:
          int  x;
  public:
          int  y;
};

int main(void){
   Foo  obj1 ;
   obj1.x = 11; // compile ERROR
   obj1.y = 22; // access y
}
```

C

```
struct  Foo{

    int  x;
    int  y;

};

int main(void){
   struct Foo  obj1 ;
   obj1.x = 11; // access x
   obj1.y = 22; // access y
}
```


Constructors and Destructors

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

// constructor
Rectangle::Rectangle (void)
{
    xLow = 0; yLow = 0;
    height = 1; width = 1;
}

// destructor
Rectangle::~Rectangle (void)
{
   xLow = yLow = height = width = 0;
}

int Rectangle::GetHeight() {return height;}
int Rectangle::GetWidth() {return width;}
```


Constructors

- A member function to initialize the data members.
- Constructor (if defined) is invoked when an object is created, otherwise only the memory of data member is allocated.
- Must be declared as a **public** member.
- Must have the **same** name as the class.
- No return type or return value.
- A class can have multiple constructors, as long as their signature (the parameters they take) are not the same.

Type of Constructors

- Default constructor
  - A constructor with no arguments

```
Rectangle ( );    // default constructor
```

- Augmented constructor
  - A constructor with arguments

```
Rectangle (int, int, int, int);      // augmented constructor
```

- Copy constructor
  - Must be specified if the STL containers are used to store your class object.

```
Rectangle (const Rectangle&); // copy constructor
```


Augmented Constructor

- Implementation

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

- May use member ***initialization list*** (more efficient)

```
Rectangle::Rectangle ( int x, int y, int h, int w )
: xLow (x), yLow (y), height (h), width (w)
{ }
```


Copy Constructor

- Implementation

```
Rectangle::Rectangle (const Rectangle& _src)
{
        xLow = _src. xLow;
        yLow = _src. yLow;
        height = _src. height ;
        width = _src. width ;
}
```

A Constructor Example

```
// In a source file main.cpp
#include <iostream>
#include "Rectangle.h"

main() {
        // r1 and r2 are initialized using default constructor
        Rectangle r1;
        Rectangle *r2 = new Rectangle;

        // r3 and r4 are initialized using augmented constructor
        Rectangle r3(1, 3, 6, 6);
        Rectangle *r4 = new Rectangle(0, 0, 3, 4);

        // r5 is initialized using r4 through copy constructor
        Rectangle r5(*r4);
}
```


Frequently Made Mistakes

- To specify an ***augmented*** constructor, one MUST also specify a ***default*** constructor.

```
/* The following statement results in a compile time error if
an augmented constructor is defined but default constructor
is missing */
Rectangle t;
```

- Possible solution: use default value for arguments.

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


Destructor

- A member function to delete data members when the object disappears.
- The destructor is **automatically** invoked when a class object is out of scope or is deleted.
- Must be declared as a **public** member.
- Must have the same name as class with the prefix "~".
- No return type or return value.
- Take no arguments.
- Only one destructor for a class.

Operator Overloading

- C++ can define customized "**operators**" for a class object.

```
// In a source file main.cpp
#include <iostream>
#include "Rectangle.h"

main() {
    Rectangle r1, r2(1, 3, 6, 6), r3(0, 0, 3, 4);

     r1 = r2;    // compile time error, no operator "=" is defined for Rectangle class
    if(r2==r3) // compile time error, no operator "==" is defined for Rectangle class
    {...}
}
```


Operator Overloading

```
// In the header file Rectangle.h
#ifndef RECTANGLE_H
#define RECTANGLE_H
class Rectangle {
public:
  Rectangle ( );    // constructor
  ~Rectangle( );    // destructor
  int GetHeight ( );          // return the height of the rectangle
  int GetWidth ( ); // return the width of the rectangle

  // the function prototype for operator overloading is fixed
  bool operator== (const Rectangle&);          // overloading operator "=="
  Rectangle& operator= (const Rectangle&);  // overloading operator "="
private:
  int xLow, yLow, height, width;
};
#endif
```

http://en.wikipedia.org/wiki/Operator_overloading

Operator Overloading

```
bool Rectangle::operator == (const Rectangle& _rhs)
{
  if (this == &_rhs) return true;
  if ((xLow == _rhs.xLow) && (yLow == _rhs.yLow) &&
     (height == _rhs.height) && (width == _rhs.width) ) return true;
  else return false;
}

Rectangle& Rectangle::operator = (const Rectangle& _rhs)
{
  if (this == &_rhs) return (*this);
  xLow = _rhs. xLow;
  yLow = _rhs. yLow;
  height = _rhs. height ;
  width = _rhs. width ;
  return (*this);
}
```

Important Note!!!
"this" is a reserved keyword indicating a pointer to the class object itself

I/O Operator Overloading

```
ostream& operator <<(ostream &os, const Rectangle &r);
{
  // need to implement additional GetX() and GetY member functions
  os << "Position is : " << r.GetX() << "  ";
  os << r.GetY() << endl;
  os << "Height is: " << r.GetHeight() << endl;
  os << "Width is: " << r.GetWidth() << endl;
  return os;
}

main() {
  Rectangle r1, r2(1, 3, 6, 6), r3(0, 0, 3, 4);
  std::cout << r1 << r2 << r3 << std::endl;
}
```

Ref: C++ Primer 5th chapter 14