Module 11: Programming in C++

Classes and Objects

Sourangshu Bhattacharya

Department of Computer Science and Engineering
Indian Institute of Technology, Kharagpur

sourangshu@cse.iitkgp.ac.in

Slides taken from NPTEL course on Programming in C++

by **Prof. Partha Pratim Das**

Module Objectives

- Understand the concept of classes and objects in C++

Module Outline

- Classes
- Objects
- Data Members of a class
- Member functions of a class
- `this` Pointer
- State of an Object

Classes

- A class is an implementation of a *type*. It is the only way to implement **User-defined Data Type (UDT)**
- A class contains **data members** / **attributes**
- A class has **operations** / **member functions** / **methods**
- A class defines a **namespace**
- Thus, classes offer **data abstraction** / **encapsulation** of **Object Oriented Programming**
- Classes are similar to structures that aggregate data logically
- A class is defined by `class` keyword
- Classes provide **access specifiers** for members to enforce **data hiding** that separates **implementation** from **interface**
  - `private` — accessible inside the definition of the class
  - `public` — accessible everywhere
- A class is a **blue print** for its instances (objects)

Objects

- An **object** of a class is an **instance** created according to its **blue print**. Objects can be automatically, statically, or dynamically created
- A object comprises **data members** that specify its **state**
- A object supports **member functions** that specify its **behavior**
- Data members of an object can be accesses by "." (dot) operator on the object
- Member functions are invoked by "." (dot) operator on the object
- An implicit `this` pointer holds the address of an object. This serves the **identity** of the object in C++
- `this` pointer is implicitly passed to methods

Program 11.01/02: Complex Numbers: Attributes

C Program

```
// File Name:Complex_object.c:
#include <stdio.h>


typedef struct Complex { // struct
    double re, im; // Data members
} Complex;

int main() {
    // Variable n1 declared, initialized
    Complex n1 = {4.2, 5.3};
    printf("%d %d", n1.re, n1.im); // Use
    return 0;
}
-----
4.2 5.3
```

- `struct` is a keyword in C for data aggregation
- The `struct Complex` is defined as composite data type containing two double (re, im) data members
- `struct Complex` is a derived data type used to create Complex type variable n1
- Data members are accessed using '.' operator
- `struct` only aggregates

C++ Program

```
// File Name:Complex_object_c++.cpp:
#include <iostream>
using namespace std;

class Complex { public: // class
    double re, im; // Data members
};

int main() {
   // Object n1 declared, initialized
   Complex n1 = {4.2, 5.3};
   cout << n1.re << " " << n1.im; // Use
   return 0;
}
-----
4.2 5.3
```

- `class` is a new keyword in C+ for data aggregation
- The `class Complex` is defined as composite data type containing two double (re, im) data members
- `class Complex` is **User-defined Data Type (UDT)** used to create Complex type object n1
- Data members are accessed using '.' operator.
- `class` aggregates and helps to do more for building a UDT

Program 11.03/04: Points and Rectangles: Attributes

C Program

```
// File Name:Rectangle_object.c:
#include <stdio.h>


typedef struct { // struct Point
    int x; int y;
} Point;

typedef struct { // Rect uses Point
    Point TL; // Top-Left
    Point BR; // Bottom-Right
} Rect;

int main() {
    Rect r = {{0,2}, {5,7}};
    // r.TL <-- {0,2}; r.BR <-- {5,7}
    // r.TL.x <-- 0; r.TL.y <-- 2

    // Members of structure r accessed
    printf("[(%d %d) (%d %d)]",
        r.TL.x, r.TL.y, r.BR.x, r.BR.y);
    return 0;
}
-----
[(0 2) (5 7)]
```

C++ Program

```
// File Name:Rectangle_object_c++.cpp:
#include <iostream>
using namespace std;

class Point { public: // class Point
    int x; int y; // Data members
};

class Rect { public: // Rect uses Point
    Point TL; // Top-Left
    Point BR; // Bottom-Right
};

int main() {
    Rect r = {{0,2}, {5,7}};
    // r.TL <-- {0,2}; r.BR <-- {5,7}
    // r.TL.x <-- 0; r.TL.y <-- 2

    // Rectangle Object r accessed
    cout << "[(" << r.TL.x << " " << r.TL.y <<
        ") (" << r.BR.x << " " << r.BR.y << ")]";
    return 0;
}
-----
[(0 2) (5 7)]
```

- Data members of user-defined data types

Program 11.05/06: Stacks: Attributes

C Program

```
// File Name:Stack_object.c:
#include <stdio.h>


typedef struct Stack { // struct Stack
    char data [100];
    int top;
} Stack;

// Codes for push, pop, top, empty

int main() {
    // Variable s declared
    Stack s;
    s.top = -1;

    // Using stack for solving problems

    return 0;
}
```

C++ Program

```
// File Name:Stack_object_c++.cpp:
#include <iostream>
using namespace std;

class Stack { public: // class Stack
    char data [100];
    int top;
};

// Codes for push, pop, top, empty

int main() {
   // Object s declared
    Stack s;
    s.top = -1;

    // Using stack for solving problems

    return 0;
}
```

- Data members of mixed data types

Classes

- A class is an implementation of a *type*. It is the only way to implement **User-defined Data Type (UDT)**
- A class contains **data members** / **attributes**.
- A class defines a **namespace**
- Thus, classes offer **data abstraction** / **encapsulation** of **Object Oriented Programming**
- Classes are similar to structures that aggregate data logically
- A class is a **blue print** for its instances (objects)

Objects

- An **object** of a class is an **instance** created according to its **blue print**. Objects can be automatically, statically, or dynamically created
- A object comprises **data members** that specify its **state**
- Data members of an object can be accesses by "." (dot) operator on the object

Program 11.07/08: Complex Numbers: Methods

C Program

```
// File Name:Complex_func.c:
#include <stdio.h>
#include <math.h>


typedef struct Complex {
    double re, im;
} Complex;

// Norm of Complex Number - global fn.
double norm(Complex c) {
    return sqrt(c.re*c.re + c.im*c.im);
}
// Print number with Norm - global fn.
void print(Complex c) {
    printf("|%lf+j%lf| = ", c.re, c.im);
    printf("%lf", norm(c)); // Call global
}

int main() { Complex c = { 4.2, 5.3 };

    // Call global fn. with 'c' as param
    print(c);

    return 0;
}
-----
|4.200000+j5.300000| = 6.762396
```

C++ Program

```
// File Name:Complex_func_c++.cpp:
#include <iostream>
#include <cmath>
using namespace std;

class Complex { public:
    double re, im;

    // MEMBER FUNCTIONS / METHODS
    // Norm of Complex Number - method
    double norm() {
        return sqrt(re*re + im*im);
    }
    // Print number with Norm - method
    void print() {
        cout << "|"<< re<< "+j"<< im<< "| = ";
        cout << norm(); // Call method
    }
}; // End of class Complex
int main() { Complex c = { 4.2, 5.3 };

    // Invoke method print of 'c'
    c.print();

    return 0;
}
-----
|4.2+j5.3| = 6.7624
```

Program 11.09/10: Rectangles: Methods

Using `struct`

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

typedef struct {
    int x; int y;
} Point;
typedef struct {
    Point TL; // Top-Left
    Point BR; // Bottom-Right
} Rect;

// Global function
void computeArea(Rect r) {
    cout << abs(r.TL.x - r.BR.x) *
            abs(r.BR.y - r.TL.y);
}

int main() {
    Rect r = { { 0, 2 }, { 5, 7 } };

    // Global fn. call
    computeArea(r);

    return 0;
}
-----
25
```

Using `class`

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

class Point { public:
    int x; int y;
};
class Rect { public:
    Point TL; // Top-Left
    Point BR; // Bottom-Right


    // Method
    void computeArea() {
        cout << abs(TL.x - BR.x) *
                abs(BR.y - TL.y);
    }
};
int main() {
    Rect r = { { 0, 2 }, { 5, 7 } };

    // Method invocation
    r.computeArea();

    return 0;
}
-----
25
```

Program 11.11/12: Stacks: Methods

Using `struct`

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

typedef struct Stack {
    char data_[100]; int top_;
} Stack;
bool empty(const Stack& s)
    { return (s.top_ == -1); }
char top(const Stack& s)
    { return  s.data_[s.top_]; }
void push(Stack& s, char x)
    { s.data_[++(s.top_)] = x; }
void pop(Stack& s) { --(s.top_); }

int main() {
    Stack s; s.top_ = -1;
    char str[10] = "ABCDE"; int i;

    for (i = 0; i < 5; ++i) push(s, str[i]);

    cout << "Reversed String: ";
    while (!empty(s)) {
        cout << top(s); pop(s);
    }
    return 0;
}
-----
Reversed String: EDCBA
```

Using `class`

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

class Stack { public:
    char data_[100]; int top_;
    // METHODS
    bool empty() { return (top_ == -1); }

    char top() { return data_[top_]; }

    void push(char x) { data_[++top_] = x; }

    void pop() { --top_; }
};
int main() {
    Stack s; s.top_ = -1;
    char str[10] = "ABCDE"; int i;

    for (i = 0; i < 5; ++i) s.push(str[i]);

    cout << "Reversed String: ";
    while (!s.empty()) {
        cout << s.top(); s.pop();
    }
    return 0;
}
-----
Reversed String: EDCBA
```

Classes

- A class has **operations** / **member functions** / **methods**
- A class defines a **namespace**
- Thus, classes offer **data abstraction** / **encapsulation** of **Object Oriented Programming**

Objects

- An **object** of a class is an **instance** created according to its **blue print**. Objects can be automatically, statically, or dynamically created
- A object supports **member functions** that specify its **behavior**
- Member functions are invoked by "." (dot) operator on the object

Program 11.13: this Pointer

- An implicit `this` pointer holds the address of an object
- `this` pointer serves as the **identity** of the object in C++
- Type of `this` pointer for a `class X` object: `X * const this;`
- `this` pointer is accessible *only in methods*

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

class X { public: int m1, m2;
    void f(int k1, int k2) {                // Sample Method
        m1 = k1;                            // Implicit access w/o 'this' pointer
        this->m2 = k2;                      // Explicit access w/  'this' pointer
        cout << "Id   = " << this << endl; // Identity (address) of the object
    }
};
int main() {
    X a;
    a.f(2, 3);
    cout << "Addr = " << &a << endl;        // Address (identity) of the object
    cout << "a.m1 = " << a.m1 << "  a.m2 = " << a.m2 << endl;
    return 0;
}
-----
Id   = 0024F918
Addr = 0024F918
a.m1 = 2  a.m2 = 3
```

this Pointer

- `this` pointer is implicitly passed to methods

In Source Code	In Binary Code
`class X { void f(int, int); ... }`	`void X::f(X * const this, int, int);`
`X a; a.f(2, 3);`	`X::f(&a, 2, 3); // &a = this`

- Use of `this` pointer
  - Distinguish member from non-member

```
class X { public: int m1, m2;
    void f(int k1, int k2) {
        m1 = k1;       // this->m1 (member) is valid; this->k1 is invalid
        this->m2 = k2; // m2 (member) is valid; this->k2 is invalid
    }
};
```

  - Explicit Use

```
// Link the object
class DoublyLinkedNode { public: DoublyLinkedNode *prev, *next; int data;
    void append(DoublyLinkedNode *x) { next = x; x->prev = this; }
}
---
// Return the object
Complex& inc() { ++re; ++im; return *this; }
```

State of an Object: Complex

- The state of an object is determined by the combined value of all its data members. Consider `class Complex`:

```
class Complex { public:
    double re_, im_;  // ordered tuple of data members decide the state at any time

    double get_re { return re_; }
    void set_re(double re) { re_ = re; }
    double get_im { return im_; }
    void set_im(double im) { im_ = im; }
}

Complex c1 = {4.2, 5.3};
// STATE 1 of c1 = {4.2, 5.3} // Denotes a tuple / sequence
```

- A method may change the state:

```
Complex c = {4.2, 5.3};
// STATE 1 of c = {4.2, 5.3}

c.set_re(6.4);
// STATE 2 of c = {6.4, 5.3}

c.get_re();
// STATE 2 of c = {6.4, 5.3} // No change of state

c.set_im(7.8);
// STATE 3 of c = {6.4, 7.8}
```

State of an Object: Rectangle

- Consider `class Point` and `class Rect`:

```
Data members of Rect class: Point TL; Point BR; // Point class type object
Data members of Point class: int x; int y

Rectangle r = {{0, 5}, {5, 0}};  // Initialization
// STATE 1 of r = {{0, 5}, {5, 0}}
{ r.TL.x = 0; r.TL.y = 5; r.BR.x = 5; r.BR.y = 0 }

r.TL.y = 9;
// STATE 2 of r = {{0, 9}, {5, 0}}

r.computeArea();
// STATE 2 of r = {{0, 9}, {5, 0}} // No change in state

Point p = {3, 4};
r.BR = p;
// STATE 3 of r = {{0, 9}, {3, 4}}
```

State of an Object: Stack

- Consider `class Stack`:

```
Data members of Stack class: char data[5] and int top;

Stack s;
// STATE 1 of s = {{?, ?, ?, ?, ?}, ?} // No data member is initialized

s.top_ = -1;
// STATE 2 of s = {{?, ?, ?, ?, ?}, -1}

s.push('b');
// STATE 3 of s = {{'b', ?, ?, ?, ?}, 0}

s.push('a');
// STATE 4 of s = {{'b', 'a', ?, ?, ?}, 1}

s.empty();
// STATE 4 of s = {{'b', 'a', ?, ?, ?}, 1} // No change of state

s.push('t');
// STATE 5 of s = {{'b', 'a', 't', ?, ?}, 2}

s.top();
// STATE 5 of s = {{'b', 'a', 't', ?, ?}, 2} // No change of state

s.pop();
// STATE 6 of s = {{'b', 'a', 't', ?, ?}, 1}
```

Module Summary

- We have covered the following:

Class	`class Complex { public:`
	`  double re_, im_;`
	
	`  double norm() { // Norm of Complex Number`
	`  return sqrt(re_ * re_ + im_ * im_);`
	`  }`
	`};`

Attributes	`Complex::re_, Complex::re_im_`
Method	`double Complex::norm();`
Object	`Complex c = {2.6, 3.9};`

Access	`c.re_ = 4.6;`
	`cout << c.im_;`
	
	`cout << c.norm;`

this Pointer	`double Complex::norm() { cout << this; return ... }`