Object Oriented Programming with C++

2024 Spring Semester

21 CST H3Art

Chapter 5 Classes and Objects

- **C Structures（C结构体）**
    - It is a **user defined data** type with a **template**

      ```
      struct student{
          char name[20];
          int roll_number;
          float total_mark;
      }

      struct student stuA;

      strcpy(stuA.name, "John");
      stuA.roll_number = 999;
      ```

    - In C, a **struct** models what a thing has/is (i.e., the **data**, also called the **characteristics**), but not what it does (its **behavior**, represented by **functions**)
        - The functions are **outside** and separated from structs
- **How struct Becomes in C++（C++中的结构体）**
    - **First Step**: Put the functions **inside**

      ```
      struct stack{
          int data[100];
          int top;
          void push(int a); // implement outside
          int pop(void); // implement outside
          bool isEmpty(void); // implement outside
      }
      ```

        - In C++, the **characteristics** and **behavior** are **integrated into a single structure**, called a **class**
            - Indeed, C++ has a new reserved word, **class**
        - Any variable of the type defined by **struct** or **class** is called an **object** or **instance** of that class
        - The packaging of the data and the functions into a class type is called **data encapsulation（数据封装）**
        - In C++, the declared variables and functions inside structs/classes are called **members**:
            - member variables
            - member functions (also called **methods**)
    - **Second Step**: data hiding
        - C++, and other **object-oriented programming** languages, allow the programmer to designate certain members of a class as **private**, and other members as **public**.
            - **Private** members cannot be accessed from outside the class, while **public** members can
            - Private members are hidden (thus the term **data hiding**)
- **Specifying a Class（指定一个类）**
    - **Class declaration**:

```
    class class_name{
        private:
            member declarations (data + functions);
        public:
            member declarations (data + functions);
    }; // End with a semicolon
    ```

    - **struct**, **union**, **class** all can be used to define a class:
        - **struct**: by default, all members are **public**
        - **union**: all members are **public** and **can not change the visibility**
        - **class**: by default all memmbers are **private**
    - **Defining member functions**:
        - Outside the class definition:

          ```
          return_type class_name::function_name(parameters){
            function body
          }
          ```

        - Inside the class definition:

          ```
          class Item{
            int number;
            float cost;
          public:
            void getdata(int a, float b){
                number = a;
                cost = b;
            }
            void putdata(void){
                cout << "number=" << number << ' ' << "cost=" << cost << endl;
            }
          };
          ```

            - When a function is **defined inside a class**, it is treated as an **inline function**
- **Accessing class members（访问类成员）**
    - Inside the class, access directly
    - Outside of the class, only **public** members can be accessed
- **Characteristics of member functions（成员函数特性）**
    - Different classes can use the **same function name**
    - Member functions can access the **private** data of the class
    - A member function can call other member functions directly
        - A **private member function** can only be called by an other member function of the same class
- **Memory allocation for objects（对象内存分配）**
    - Memory of methods created when function defined
        - All objects share one
    - Memory of data created when objects defined
        - Every object has its own data
- `this` **Pointer**
    - For every non-static method in class:

```
class t{
  private:
    int x, y;
  public:
    void set(int a, int b){
        x = a;
        y = b;
    }
};
```

is equivalent to:

```
class t{
  private:
    int x, y;
  public:
    void set(int a, int b, t* const this){
        this->x = a;
        this->y = b;
    }
};
```

- `this` pointer points to the object by which a member function is called
- The pointer `this` acts as an **implicit** argument to all the non-static member functions
- When an object of a class is created this pointer is initialized to point to the object
- `this` pointer is a **const** pointer, the value of it cannot be altered:

  ```
  t * const this;
  ```

- `this` pointer can be used explicitly:

  ```
  void set(int a,int b) {
      this->x = a;
      this->y = b;
  }
  ```

- `this` pointer also can be return **（返回this指向的值的引用，实现成员函数的链式调用）**：

```
# include <iostream>

using namespace std;

class t{
  public:
   t& set(int a, int b){
    x = a;
    y = b;
    return *this;
   }

   t& print(){
    cout << x << ', ' << y << endl;
    return *this;
   }

  private:
    int x, y;
};

void main(){
    t t1;
    t1.set.(10, 20).print().set(30, 40);
}
```

- **static data members（静态数据成员）**
  - Using keyword `static` to declare a data member as **static**
    - static data member is **shared by all the objects** of that class, no matter how many objects are created
    - A static data member should be **initialized outside the declaration of a class**
    - It is **visible only within the class**, but its **lifetime is the entire program**
  - Static data members belong to the **class** instead of **objects**
  - only public static data members can be accessed from outside of the class as:

    ```
    class_name::public static data memeber;
    ```

    - Notice: Use `class_name` to access static data member
  - From inside the class, all the static data members can be accessed directly
  - static data members should be initialized outside the class:

    ```
    type class_name::static_data_name = initial_value;
    ```

- **static member functions（静态成员函数）**
  - Static member functions are used to **access static member variables**
  - Static member functions have **NO** `this` pointer, it cannot access object's non-static variable directly
  - Static member functions can be called in following form:

    ```
    class_name::static_function_name(arguments_list);
    object_name::static_function_name(arguments_list);
    ```

- **friendly functions（友元函数）**
  - To make an outside function **"friendly"** to a class:

```
class X{
    int i;
    friend void func(X*, int); // friendly function
  public:
    void memeber_func(int);
};
```

- `func()` is **NOT** the member function of `class X`
- `func()` can be defined elsewhere in the program like a normal C++ function
- The definition of the `func()` does not use either the **keyword** `friend` or the **scope operator** `::`
- `func()` can access **private** members of the `class X`
- `func()` **cannot** access member names directly:

  ```
  void func(X* xptr, int a){
    xptr-> i = a;
  }
  ```

- Member functions of one class can be **friend functions of another class**:

  ```
  class X{
    ...
    public:
      void func();
    ...
  };

  class Y{
    ...
    public:
      friend void X::func();
    ...
  }
  ```

- Declare the `class Y` to be a **friend class（友元类）** of the `class X`, then **all the member functions** of class Y are friend functions of the `class X`
  - Friendly functions are **one-way（单向）**

```
class A{
    friend class B;
    int x;
  public:
    void display(){
      cout << x << endl;
    }
};

class B{
  public:
    void set(int i){
      a.x = i;
    }
    void display(){
      a.display();
    }
  private:
    A a;
}
```

- **Pointers to Members（成员指针）**
    - It is possible to take the address of a **non-static** member of a class and assign it to a pointer:

      ```
      class circle{
        public:
          int radius;
          void setradius(int);
      };

      int circle::*pint;
      pint = &circle::radius;

      circle c;
      c.radius = 10; // OK
      c.*pint = 10; // OK

      circle *pc = &c;
      pc->radius = 20; // OK
      pc->*pint = 20; // OK

      pint = &c.radius; // Err
      int* ip = &c.radius; // OK
      ```

    - **Pointers to Member Functions（指向成员函数的指针）**：
        - Syntex: `data_type (class_name::*variable_name)(arglist);`
        - eg(using above circle class):

          ```
          void (circle::*pmf)(int) = &circle::setradius;

          c.setradius(10);
          (c.*pmf)(10);

          pc->setradius(10);
          (pc->*pmf)(10);
          ```

        - Why use pointers to member functions: **Polymorphism（多态）**

```
class screen{
  public:
    screen& home();
    screen& forward();
    screen& back();
    screen& up();
    screen& down();
};

screen& move(screen &obj, screen &(screen::*pmf)()){
  (obj.*pmf)();
}

screen obj;
screen &(screen::*pmf)();

pmf = &screen::home;
move(obj, pmf);
pmf = &screen::forward;
move(obj, pmf);
```