Object Oriented Programming with C++

2024 Spring Semester

21 CST H3Art

Chapter 6 Constructors and Destructors

- **Constructors（构造函数）**
  - Enable an object to initialize itself when it is created
  - A special member function to initialize the objects of its class
    - **No return value（无返回值）**
    - The function **name is the same as the class（函数名与类名相同）**
    - Should be declared in the **public section（在公有段定义）**：
      - Defined in the **private section**, you can implement **static classes** that cannot create instances.
    - Invoked automatically when the objects are created
    - Allocate memory space for the non-static data members of an object
    - Initialize part or all data members of an object
  - **Multiple constructors（多个构造函数）** in a class
    - When we do **not implement the constructor ourselves**, the compiler will automatically implement a **parameterless constructor** for us
      - If we implement any constructor, this parameterless constructor will **not automatically generated（自动生成的无参构造函数只会在开发者没有自己实现构造函数时生成）**
    - **Overloaded constructors（重载构造函数）**: argument list must be different

```
class X{
  public:
    X();
    X(int);
    X(int, char);
    X(float, char);
  private:
    int m;
    char c;
};

void f(){
  X a;  // invoke construtor X()
  X b(1); // invoke construtor X(int)
  X c(1, 'c');  // invoke construtor X(int, char)
  X d(2.3, 'd');  // invoke construtor X(float, char)
}
```

    - **Default argument constructor** and **default constructor**. If a class has a constructor with all arguments have default values, it is **illegal** to overload a default constructor for the class

```
class X{
  public:
    X();
    X(int i = 0);
};

int main(){
  X x; // ambiguity, call X::X() or X::X(int i = 0) ?
}
```

- **Initializer List（初始化列表）**
  - Some special data members (e.g.**constant members**, **reference members** etc.) **cannot** be initialized with assignment statements in the constructor function body:

    ```
    class X{
        const int a;
        const int &r;
      public:
        X(){
          a = 9; // Err
          r = a; // Err
        }
    };
    ```

  - To achieve the above requirement, we need initializer list:

    `constructor_name(arglist): member_name(expression), ...{ function_body }`

    ```
    class X{
        const int a;
        const int &r;
      public:
        X(): a(9), r(a){}
    };
    ```

  - Usage of initializer list:
    - Initialize normal data members
    - Initialize **object data members（初始化对象数据成员）**

```
class studentID {
  public:
    studentID(int d) {
      value = d;
      cout << "Assigning student id " << value << endl;
    }

  protected:
    int value;
};

class student {
  public:
    student(char *pname = "no name", int ssID = 0){
      // Constructor for 'student' must explicitly initialize
      // the member 'id' which does not have a default constructor
      id = studentID(ssID); // Err
      cout << "Constructing student" << pname << endl;
      name = pname;
    }

  protected:
    char *name;
    studentID id;
};
```

        - Initialize **constant data members（常量初始化）** and **reference data members（引用初始化）**
    - **Default constructor（默认构造函数）**
        - If **no constructor is defined**, the complier supplies a default constructor
        - Default constructor:
            - **Object member（对象成员）**: initialize with the default constructor of its own class（仅使用这些对象成员的默认构造函数初始化）
            - **Data members of build-in or compound data type（内建数据类型或复合数据类型/结构体）**: only initialize global object
        - Classes with **constant member** and **reference member** cannot use the default constructor provided by the complier（具有常量或引用数据成员的类，不能使用默认构造函数）
    - **Arrays of Objects（对象数组）**
        - Arrays of variables that are of the type **class**
        - Arrays of objects **cannot be initialized** unless the class have:
            - no constructor
            - a default constructor
            - a constructor with all parameters have default value
    - **Dynamic Initialization of Objects（对象动态初始化）**
        - The initial value of an object may be provided during run time

```
class teacher{
  private:
    int *id;
  public:
    teacher(int input_id){
      *id = input_id; // Segmentation Fault, the variable *id wasn't allocated memory space.
      id = new int(input_id); // OK
    }
};
```

- **Destructors（析构函数）**

- Destroy the objects when they are **out of scope（离开作用域）** to clean up storage
- The destructor is a member function whose name is `~class_name`, it take no argument nor does it return any value
- When objects are out of scope, compiler **invokes destructor automatically**
- If no destructor is defined, the **complier** supplies a **default destructor**
- Constructors and destructors are automatically called in **reverse order**:

```
class X{
  ...
};

void func(int a){
  X aa; // Call constructor for aa
  X bb; // Call constructor for bb
  if (a > 0){
    X cc; // Call constructor for cc
    ...
  } // Destroy cc
  X dd; // Call constructor for dd

  // Destroy dd
  // Destroy bb
  // Destroy aa
}
```

- Whenever `new` is used to allocate memory in the constructors, we should use `delete` to free that memory:

```
class teacher{
  private:
    int *id;
  public:
    teacher(int input_id){
      id = new int(input_id);
    }

    ~teacher(){
      delete id;
    }
    void show();
};
```

- **Copy constructor（拷贝构造函数）**
  - Copy constructor is a constructor
  - The argument of the copy constructor is a **reference to its own class（引用作为参数，直接读取被拷贝对象的内容，一般还加`const`修饰，使值仅可读）**

```
class point{
  private:
    int x, y;
  public:
    point(int intx = 0, int inty = 0){
      x = intx;
      y = inty;
    }
    point(const point& pt){
      x = pt.x;
      y = pt.y;
    }
};
```

- If not copy constructor is defined, **compiler** supplies a **default copy constructor**
- **Default copy constructor** assigns the values of one object to another object **member by member**
- When the copy constructor will be used: **（3种拷贝构造函数被调用的情况）**
  - Declare and initialize an object from another object **（初始化对象）**
  - Pass value to an object argument while calling a function **（函数传参——值传递）**
  - When the return value of a function is an object, calling copy constructor to copy the returned object to a temporary object **（函数返回类对象时会创建一个对象拷贝）**

```
class location{
  private:
    int X, Y;
  public:
    location(int xx = 0, int yy = 0){
      X = xx;
      Y = yy;
      cout << "Obj constructed" << endl;
    }
    location(const location& p){
      X = p.X;
      Y = p.Y;
      cout << "Copy constructor called" << endl;
    }
    ~location(){
      cout << X << "," << Y << "Obj destroyed" << endl;
    }
};

location g(){
  location A(1, 2);
  return A;
}

int main(){
  location B;
  B = g();
  return 0;
}
```

The output of this program is as follows:

```
Obj constructed           // Create object B
Obj constructed           // Create local object A
Copy constructor called   // Initialize temporary object for return value
1 , 2 Obj destroyed       // Delete temporary object
1 , 2 Obj destroyed       // Delete local object A
1 , 2 Obj destroyed       // Delete object B
```

- For some object, we **cannot pass values by simply assigning values member by member** (i.e. cannot use the default copy constructor provided by the complier).

```
class c{
  private:
    int *p;
  public:
    c(){
      p = new int(0);
    }
    c(const c& obj){
      *p = obj.*p;
    }
};
```