

# Assignment 7

---

Fuyun Wang

191300051

---

## Assignment 7

### 一.概念题

1.1 C++中虚函数的作用是什么？为什么C++中析构函数往往是虚函数？

1.2 1.2 简述C++中静态绑定和动态绑定的概念，并说明动态绑定发生的情况

### 二.编程题

2.1

2.2

## 一.概念题

### 1.1 C++中虚函数的作用是什么？为什么C++中析构函数往往是虚函数？

(1) 实现多态性的动态绑定。对于以public继承的两个类，可以使用基类的指针或者引用访问派生类。但是C++默认的是使用静态的消息绑定，采用基类的消息处理。使用虚函数后，调用这样的同名函数时就会调用派生类的函数。

(2) 将可能会被继承的父类的析构函数设置为虚函数，可以保证当我们new一个子类，然后使用基类指针指向该子类对象，释放基类指针时可以释放掉子类的空间，防止内存泄漏。

### 1.2 1.2 简述C++中静态绑定和动态绑定的概念，并说明动态绑定发生的情况

- 静态绑定：绑定的是静态类型，所对应的函数或属性依赖于对象的静态类型，发生在编译期；
- 动态绑定：绑定的是动态类型，所对应的函数或属性依赖于对象的动态类型，发生在运行期；

非虚函数一般是静态绑定，虚函数一般是动态绑定。

## 二.编程题

### 2.1

```
1 default construct A
2 default construct A
3 default construct B
4 copy construct A
```

```
5  A::f
6  A::g
7  destruct A
8  A::f
9  A::g
10 copy construct A
11 A::f
12 A::g
13 destruct A
14 A::f
15 B::g
16 copy construct A
17 A::f
18 A::g
19 destruct A
20 A::f
21 A::g
22 destruct A
23 destruct B
24 destruct A
```

## 2.2

```
1  #include <iostream>
2  #include <queue>
3  #include <set>
4  using namespace std;
5
6  //使用STL实现。
7  class Queue
8  {
9  public:
10     virtual bool enqueue(int num) = 0;
11     virtual bool dequeue(int &num) = 0;
12 };
13 //先进先出
14 class Queue1 : public Queue
15 {
16 private:
17     std::queue<int> lst;
18
19 public:
20     bool enqueue(int num)
21     {
22         lst.push(num);
23         return true;
24     }
25     bool dequeue(int &num)
26     {
```

```
27         if (lst.empty())
28         {
29             return false;
30         }
31         num = lst.front();
32         return true;
33     }
34 };
35 //最小队列
36 class Queue2 : public Queue
37 {
38 private:
39     //std::less<int>() is a constructor call. It creates a new
    std::less<int> object which, yes, has overloaded operator().
40     std::multiset<int> lst;
41
42 public:
43     bool enqueue(int num)
44     {
45         lst.insert(num);
46         return true;
47     }
48     bool dequeue(int &num)
49     {
50         if (lst.empty())
51         {
52             return false;
53         }
54         num = *lst.begin();
55         lst.erase(lst.begin());
56         return true;
57     }
58 };
59 //最大队列
60
61
62 //通过构造一个新的类作为比较函数。
63 //或者重载操作符。
64 class CTestCmp {
65 public:
66     bool operator() (int x, int y) {
67         return x>y;
68     }
69 };
70 class Queue3 : public Queue
71 {
72 private:
73     std::multiset<int,CTestCmp> lst;
74
```

```

75 public:
76     bool enqueue(int num)
77     {
78         lst.insert(num);
79         return true;
80     }
81     bool dequeue(int &num)
82     {
83         if (lst.empty())
84         {
85             return false;
86         }
87
88         // set<int>::iterator i = lst.end();
89         // i--;
90         // num = *i;
91         // lst.erase(i);
92         num=*lst.begin();
93         lst.erase(lst.begin());
94         return true;
95     }
96 };
97
98 int main()
99 {
100     Queue3 a;
101     int num = 0;
102     std::cout << a.enqueue(1) << std::endl;
103     std::cout << a.enqueue(1) << std::endl;
104     std::cout << a.enqueue(2) << std::endl;
105     std::cout << a.dequeue(num) << std::endl;
106     std::cout << num << std::endl;
107     std::cout << a.dequeue(num) << std::endl;
108     std::cout << num << std::endl;
109     std::cout << a.dequeue(num) << std::endl;
110     std::cout << num << std::endl;
111 }

```