Assignment 7

Fuyun Wang

191300051

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

```
default construct A
default construct A
default construct B
copy construct A
```

```
A::f
A::g
destruct A
A::f
A::g
copy construct A
A::f
A::g
destruct A
A::f
B::g
copy construct A
A::f
A::g
destruct A
A::f
A::g
destruct A
destruct B
destruct A
```

2.2

```cpp
#include <iostream>
#include <queue>
#include <set>
using namespace std;

//使用STL实现。
class Queue
{
public:
    virtual bool enqueue(int num) = 0;
    virtual bool dequeue(int &num) = 0;
};
//先进先出
class Queue1 : public Queue
{
private:
    std::queue<int> lst;

public:
    bool enqueue(int num)
    {
        lst.push(num);
        return true;
    }
    bool dequeue(int &num)
    {
```

```
        if (lst.empty())
        {
            return false;
        }
        num = lst.front();
        return true;
    }
};
//最小队列
class Queue2 : public Queue
{
private:
    //std::less<int>() is a constructor call. It creates a new
std::less<int> object which, yes, has overloaded operator().
    std::multiset<int> lst;

public:
    bool enqueue(int num)
    {
        lst.insert(num);
        return true;
    }
    bool dequeue(int &num)
    {
        if (lst.empty())
        {
            return false;
        }
        num = *lst.begin();
        lst.erase(lst.begin());
        return true;
    }
};
//最大队列


//通过构造一个新的类作为比较函数。
//或者重载操作符。
class CTestCmp {
public:
        bool operator() (int x, int y) {
            return x>y;
        }
};
class Queue3 : public Queue
{
private:
    std::multiset<int,CTestCmp> lst;

```

```
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 }
```