# HW7

*181220076 周韧哲*

## 一. 概念题

1. 虚函数是C++中用于实现多态的机制，核心理念就是通过基类访问派生类定义的函数，作用是允许在派生类中重新定义与基类同名的函数，并且可以通过基类指针或引用来访问基类和派生类中的同名函数。如果以一个基类指针指向其派生类，删除这个基类指针只能删除基类对象部分，而不能删除整个派生类对象，原因是通过基类指针无法访问派生类的析构函数。
   但是，当基类的析构函数也是虚的，那么派生类的析构函数也必然是虚的，删除基类指针时，它就会通过虚函数表找到正确的派生类析构函数并调用它，从而正确析构整个派生类对象。
2. 静态绑定：绑定的是对象的静态类型，某特性（比如函数）依赖于对象的静态类型，发生在编译期。动态绑定：绑定的是对象的动态类型，某特性（比如函数）依赖于对象的动态类型，发生在运行期。对于一个类如果有虚函数，则编译程序会创建一个虚函数表，记录了该类所有的虚函数入口地址，当创建一个包含虚函数的类的对象时，在所创建对象的内存空间中有一个隐藏的指针指向该对象所属类的虚函数表，从而当通过基类的引用或指针来访问基类的虚成员函数时，就会利用实际引用或指向的对象的虚函数表来动态绑定调用的函数，从而处于动态绑定的情况。

## 二. 编程题

1. 运行结果如下：

```
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.

```
class Queue{
protected:
    struct Node{
        int data;
```

```
        Node *next;
        Node *prev;
    }*top;
public:
    Queue():top(NULL){}
    virtual bool enqueue(int num) = 0;
    virtual bool dequeue(int &num) = 0;
    void print(){
        for(Node *p=top;p!=NULL;p=p->next){
            cout<<p->data<<" ";
        }
        cout<<endl;
    }
};

class Queue1:public Queue{
    Node *tail;
public:
    Queue1():tail(NULL){}
    bool enqueue(int num){
        Node *p = new Node;
        p->data = num;
        p->next = NULL;
        if(tail==NULL){
            top = tail = p;
            p->prev = NULL;
        }else{
            assert(tail->next==NULL);
            tail->next = p;
            p->prev = tail;
            tail = p;
        }
        return true;
    }
    bool dequeue(int &num){
        if(top==NULL){
            cout<<"The Queue is empty!"<<endl;
            return false;
        }
        if(top==tail){
            num = top->data;
            delete top;
            top = tail = NULL;
        }else{
            assert(top->next!=NULL);
            Node *p = top;
            top = top->next;
            top->prev = NULL;
            num = p->data;
            delete p;
        }
        return true;
    }
};

class Queue2:public Queue{
    Node *min;
    void find_min(){
```

```
        min = top;
        for(Node *p=top->next;p!=NULL;p=p->next){
            if(min->data > p->data)
                min = p;
        }
    }
public:
    Queue2():min(NULL){}
    bool enqueue(int num){
        Node *p = new Node;
        p->data = num;
        p->prev = NULL;
        if(top==NULL){
            p->next = NULL;
            top = min = p;
        }else{
            p->next = top;
            top->prev = p;
            top = p;
            if(min->data > top->data)
                min = top;
        }
        return true;
    }
    bool dequeue(int &num){
        if(top==NULL){
            cout<<"The Queue is empty!"<<endl;
            return false;
        }
        if(top->next==NULL){
            num = top->data;
            delete top;
            top = min = NULL;
        }else{
            if(min==top){
                Node *p = top;
                top = top->next;
                top->prev = NULL;
                num = p->data;
                delete p;
                min = NULL;
            }else{
                assert(min->prev!=NULL);
                Node *prev = min->prev;
                Node *next = min->next;
                prev->next = next;
                if(next)
                    next->prev = prev;
                num = min->data;
                delete min;
                min = NULL;
            }
            find_min();
        }
        return true;
    }
};

```

```
class Queue3:public Queue{
    Node *max;
    void find_max(){
        max = top;
        for(Node *p=top->next;p!=NULL;p=p->next){
            if(max->data < p->data)
                max = p;
        }
    }
public:
    Queue3():max(NULL){}
    bool enqueue(int num){
        Node *p = new Node;
        p->data = num;
        p->prev = NULL;
        if(top==NULL){
            p->next = NULL;
            top = max = p;
        }else{
            p->next = top;
            top->prev = p;
            top = p;
            if(max->data < top->data)
                max = top;
        }
        return true;
    }
    bool dequeue(int &num){
        if(top==NULL){
            cout<<"The Queue is empty!"<<endl;
            return false;
        }
        if(top->next==NULL){
            num = top->data;
            delete top;
            top = max = NULL;
        }else{
            if(max==top){
                Node *p = top;
                top = top->next;
                top->prev = NULL;
                num = p->data;
                delete p;
                max = NULL;
            }else{
                assert(max->prev!=NULL);
                Node *prev = max->prev;
                Node *next = max->next;
                prev->next = next;
                if(next)
                    next->prev = prev;
                num = max->data;
                delete max;
                max = NULL;
            }
            find_max();
        }
        return true;
```

```
179     }
180 };
```