HW1

181220076 周韧哲

1、从数据和过程的角度，简述抽象与封装的区别。

- 过程角度：
  - **过程抽象**是用一个名字来代表一段完成一定功能的程序代码，代码的使用者只需要知道代码的名字以及相应的功能，而不需要知道对应的程序代码是如何实现的。
  - **过程封装**是把命名代码的具体实现隐藏起来（对使用者不可见，或不可直接访问），使用者只能通过代码名字来使用相应的代码。命名代码所需要的数据是通过参数（或全局变量）来获得，计算结果通过返回机制（或全局变量）返回。
- 数据角度：
  - **数据抽象**只描述对数据能实施哪些操作以及这些操作之间的关系，数据的使用者不需要知道数据的具体表示形式。
  - **数据封装**把数据及其操作作为一个整体来进行实现，其中，数据的具体表示被隐藏起来（使用者不可见，或不可直接访问），对数据的访问（使用）只能通过提供的操作（对外接口）来完成。

2、简述面向过程与面向对象程序设计的区别；列举两个更适合面向对象的场景，并说明理由。

- 面向过程程序设计的重点是分析解决问题的步骤，以及完成步骤的流程，是一种结构化自上而下的程序设计方法，其函数可以实现代码复用。而面向对象程序设计的重点是把构成问题的事物分解成对象，从局部着手，通过迭代的方式逐步构建出整个程序，是一种以数据为核心，以类设计为主的自下而上的程序设计方法，类可以实现代码复用。
- 场景：游戏设计，学生管理系统等。因为游戏需要有大量的NPC，学生管理系统需要面向每个学生，由于面向对象的抽象、封装、继承、多态的特性，使用面向对象程序设计可以使得系统有更好的扩展性、维护性、复用性。

3、链表队列

```
class Queue{
private:
    struct Node{
        int data;
        Node *next;
    }*top, *tail;
    int count;
public:
    ListQueue(){
        top = tail = NULL;
        count = 0;
    }
    void enQueue(int i){
        if(count==100){
            cout<<"The Queue is overflow!"<<endl;
            exit(-1);
        }
```

```
        Node *p = new Node;
        p->data = i;
        p->next = NULL;
        if(tail==NULL){
            top = tail = p;
        }else{
            assert(tail->next==NULL);
            tail->next = p;
            tail = p;
        }
        count++;
    }
    void deQueue(int &i){
        if(top==NULL){
            cout<<"The Queue is empty!"<<endl;
            exit(-1);
        }
        if(top==tail){
            i = top->data;
            delete top;
            top = tail = NULL;
        }else{
            assert(top->next!=NULL);
            Node *p = top;
            top = top->next;
            i = p->data;
            delete p;
        }
        count--;
    }
};
```

4、动态数组队列

```
class Queue{
private:
    int *array, top, tail, size, count;
public:
    ArrayQueue(){
        size = 10;
        array = new int[size];
        top = tail = count = 0;
    }
    void enQueue(int i){
        if(count==size){
            if(size==100){
                cout<<"The Queue is overflow!"<<endl;
                exit(-1);
            }
            int *tmp = new int[size+10];
            for(int i=0;i<count;i++){
```

```
                int p = (top+i)%size;
                tmp[i] = array[p];
            }
            size += 10;
            top = 0;
            tail = count;
            delete array;
            array = tmp;
        }
        array[tail] = i;
        tail = (tail+1)%size;
        count++;
    }
    void deQueue(int &i){
        if(count==0){
            cout<<"The Queue is empty!"<<endl;
            exit(-1);
        }
        i = array[top];
        top = (top+1)%size;
        count--;
    }
};
```