Assignment 1

191220022 丁一凡

一、 概念题

1.

数据角度： 抽象指针对某个数据，描述出能对其实施的所有操作及这些操作之间的关系；而封装指把数据的表示及其操作作为一个整体描述，并把数据隐藏起来。

过程角度： 抽象指把程序的一些功能抽象为子程序，是有着只需要知道其接口而不需要知道其具体实现；而封装指对数据操作的封装，它通过子程序来实现。

2.

面向过程的程序设计基于过程抽象和封装，而面向对象的程序设计基于数据抽象和封装。

面向过程的程序设计对数据和操作的描述是分离的，这往往不利于程序的设计、理解与维护，而且操作所需要的数据是公开的，缺乏对数据的保护，同时不利于程序设计的模块化以及软件复用。但面向对象的程序设计能通过对数据的抽象与封装，很好的解决上述问题。

二、 编程题

1. 数组实现

```
class Queue
{
public:
    Queue() { capacity = 11; buffer = new int[capacity]; top = bottom = 0; }
    void enQueue(int i)
    {
        if ((top + 1) % capacity == bottom)
        {
            int n = 0;
            switch(capacity)
            {
                case 10: capacity = 20;break;
                case 20: capacity = 50;break;
                case 50: capacity = 100;break;
                default:
                    std::cout << "Queue is overflow.\n";
                    exit(-1);
            }
            int *temp = buffer;
            buffer = new int[capacity];
            while(top>bottom)
```

```
            {
                top--;
                buffer[n]=temp[top];
                n++;
            }
        }
        bottom = 0;
        top = n;
        buffer[top] = i;
        top = (top + 1) % capacity;
        delete []temp;
    }
    void deQueue(int& i)
    {
        if (top == bottom)
        {
            cout << "Queue is empty.\n";
            exit(-1);
        }
        i = buffer[bottom];
        bottom = (bottom+1)%capacity;
    }
    void printAll()
    {
        if (top == bottom)
        {
            cout << "Queue is empty.\n";
        }
        else
        {
            for (int i = top - 1; i >= bottom; i--)
            {
                cout << buffer[i] << "    ";
            }
            cout << endl;
        }
    }
private:
    int capacity;
    int top, bottom;
    int* buffer;
};
```

2. 链表实现

```
class Queue
{
    public:
    Queue() {capacity=10;num=0;bottom=buffer=NULL;}
    void enQueue(int i)
    {
        if(num==capacity)
        {
            switch(capacity)
            {
```

```
                case 10: capacity = 20;break;
                case 20: capacity = 50;break;
                case 50: capacity = 100;break;
                default:
                    cout << "Queue is overflow.\n";
                    exit(-1);
            }
            Node *p = new Node;
            p->data = i;
            p->next = NULL;
            if(num==0)
            {
                bottom = top = p;
            }
            else
            {
                top->next=p;
            }
            num++;
        }
    }
    void deQueue(int &i)
    {
        if(num==0)
        {
            cout << "Queue is empty.\n";
            exit(-1);
        }
        i = bottom->data;
        Node* p =bottom;
        bottom = bottom->next;
        p->next = NULL;
        delete p;
    }
    void printAll()
    {
        if(num==0)
            cout << "Queue is empty.\n";
        else
        {
            for(Node *p=bottom;p!=NULL;p=p->next)
                cout << p->data << "    ";
            cout << endl;
        }
    }
    private:
    struct Node
    {
        int data;
        Node* next;
    };
    Node* top,*bottom;
    int capacity;
    int num;
}
```