第2次上机作业：

1.将编号为0和I的两个栈存放于一个数组空间 V[m]中，栈底分别处于数组的两端。当第0号栈的栈顶指针top[0]等于-1时该栈为空，当第1号栈的栈顶指针top[1]等于m时该我为空。两个栈均从两端向中间增长。当向第0号栈插入一个新元素时，使 top[0]增1得到新的栈顶位置，当向第1号栈插入一个新元素时，使top[1]减1得到新的栈顶位置。当 top[0]+1==top[1]时或top[0]==top[1]-1时，栈空间满，此时不能再向任一栈加入新的元素。试定义这种双栈(Double Stack)结构的类定义，并实现判栈空、判栈满、插入、删除算法。

2.假设以数组 sequ[m]存放循环队列的元素，同时设变量 rear 和quelen 分别指示循环队列中队尾元素的位置和内含元素的个数。试给出判别此循环队列的队满条件，并写出相应的入队列和出队列的算法(在出队的算法中要返回队头元素)。

3.假设以带头结点的循环链表表示队列，并且只设一个指针指向队尾元素结点(注意不设头指针)，试编写相应的置空队、入队列和出队列的算法。

1. 双栈结构体定义，
    - 属性有：
        1. 栈顶指针 * 2
        2. 数组长度
    - 要实现的方法有：
        1. 栈空栈满判断
        2. 插入
        3. 删除

```
#include <iostream>

class DoubleStack {
private:
    int* arr;     // 存储栈元素的数组
    int top[2];   // 两个栈顶指针
    int size;     // 数组的长度

public:
    DoubleStack(int size) : size(size) {
        arr = new int[size];
        top[0] = -1;
        top[1] = size;
    }

    // 析构函数 释放内存 防止内存泄漏
    ~DoubleStack() {
        delete[] arr;
    }
```

```
    bool isFull() {
        return top[0] + 1 == top[1];
    }

    bool isEmpty(int stackId) {
        if (stackId == 0)
            return top[0] == -1;
        else
            return top[1] == size;
    }

    void push(int stackId, int value) {
        if (isFull())
            throw std::overflow_error("Stack is full");
        if (stackId == 0)
            arr[++top[0]] = value;
        else
            arr[--top[1]] = value;
    }

    int pop(int stackId) {
        if (isEmpty(stackId))
            throw std::underflow_error("Stack is empty");
        if (stackId == 0)
            return arr[top[0]--];
        else
            return arr[top[1]++];
    }
};

int main() {
    DoubleStack ds(5); // 创建一个大小为5的双栈

    // 对第一个栈进行操作
    // 检查栈空栈满状态
    std::cout << "Is stack 0 empty? " << (ds.isEmpty(0) ? "Yes" : "No")
<< std::endl; // 应该输出 Yes
    ds.push(0, 1);
    ds.push(0, 2);
    ds.push(0, 3);
    std::cout << "Pop from stack 0: " << ds.pop(0) << std::endl; // 应该
输出 3

    // 对第二个栈进行操作
    std::cout << "Is stack 1 empty? " << (ds.isEmpty(1) ? "Yes" : "No")
<< std::endl; // 应该输出 Yes
    ds.push(1, 10);
    ds.push(1, 20);
```

```
    ds.push(1, 30);

    std::cout << "Is stack full? " << (ds.isFull() ? "Yes" : "No") <<
std::endl; // 应该输出 Yes
    std::cout << "Pop from stack 1: " << ds.pop(1) << std::endl; // 应该
输出 30
    std::cout << "Is stack full? " << (ds.isFull() ? "Yes" : "No") <<
std::endl; // 应该输出 No

    return 0;
}
```

```
➜ output git:(main) X ./"double_stack"
Is stack 0 empty? Yes
Pop from stack 0: 3
Is stack 1 empty? Yes
Is stack full? Yes
Pop from stack 1: 30
Is stack full? No
```

2. 循环队列结构体定义

> 借助数组实现

+ 属性

1. 队尾指针
2. 队列长度 quelen
3. 数组长度 m

+ 方法

1. 队满判读
2. 入队
3. 出队

```
#include <iostream>
using namespace std;

class CircularQueue {
private:
    int m;        // 数组长度
    int rear;     // 队尾指针
    int quelen;   // 队列当前长度
    int* arr;     // 存储队列元素的数组

public:
    CircularQueue(int size) : m(size), rear(0), quelen(0) {
        arr = new int[m];
    }

    ~CircularQueue() {
        delete[] arr;
    }
```

```
    bool isFull() {
        return quelen == m;
    }

    bool isEmpty() {
        return quelen == 0;
    }

    void enqueue(int value) {
        if (isFull())
            throw std::overflow_error("Queue is full");
        arr[rear] = value;
        rear = (rear + 1) % m;
        ++quelen;
    }

    int dequeue() {
        if (isEmpty())
            throw std::underflow_error("Queue is empty");
        int front = (rear - quelen + m) % m;  // 计算队头指针
        int value = arr[front];
        quelen--;
        return value;
    }
};

int main() {
    CircularQueue cq(5); // 创建一个大小为5的循环队列

    // 入队
    cq.enqueue(1);
    cq.enqueue(2);
    cq.enqueue(3);
    cq.enqueue(4);
    cq.enqueue(5);

    // 出队
    std::cout << "Dequeue: " << cq.dequeue() << std::endl; // 应该输出 1
    std::cout << "Dequeue: " << cq.dequeue() << std::endl; // 应该输出 2

    // 再次入队
    cq.enqueue(6);
    cq.enqueue(7);

    std::cout << "Is queue full? " << (cq.isFull() ? "Yes" : "No") <<
std::endl; // 应该输出 "Yes
    // 继续出队
    while (!cq.isEmpty()) {
        std::cout << "Dequeue: " << cq.dequeue() << std::endl;  // 应该输
```

```
出 3 4 5 6 7
    }

    return 0;
}
```

```
➜ output git:(main) X ./"circular_queue"
Dequeue: 1
Dequeue: 2
Is queue full? Yes
Dequeue: 3
Dequeue: 4
Dequeue: 5
Dequeue: 6
Dequeue: 7
```

1. 队列结构体定义

> 由带头节点的循环链表实现

- 属性
  1. 队尾指针
- 方法
  1. 置空
  2. 入队
  3. 出队

```
#include <iostream>
using namespace std;

class LinkedQueue {
private:
    struct Node {
        int data;
        Node* next;
        Node(int data) : data(data), next(nullptr) {}
    };

    Node* rear;  // 队尾指针

public:
    LinkedQueue() {
        // 初始化带头节点的循环链表
        rear = new Node(0);
        rear->next = rear;
    }

    ~LinkedQueue() {
        while (!isEmpty()) {
```

```
            dequeue();
        }
        delete rear;
    }

    bool isEmpty() {
        return rear->next == rear;
    }

    void enqueue(int value) {
        Node* newNode = new Node(value);
        newNode->next = rear->next;
        rear->next = newNode;
        rear = newNode;
    }

    int dequeue() {
        if (isEmpty())
            throw std::underflow_error("Queue is empty");
        Node* front = rear->next->next;
        int value = front->data;
        rear->next->next = front->next;
        if (front == rear) {  // 如果队列只有一个元素
            rear = rear->next;
        }
        delete front;
        return value;
    }
};

int main() {
    LinkedQueue lq; // 创建一个队列

    // 入队
    lq.enqueue(1);
    lq.enqueue(2);
    lq.enqueue(3);

    // 出队
    std::cout << "Dequeue: " << lq.dequeue() << std::endl; // 应该输出 1
    std::cout << "Dequeue: " << lq.dequeue() << std::endl; // 应该输出 2

    // 检查队列是否为空
    std::cout << "Is queue empty? " << (lq.isEmpty() ? "Yes" : "No") <<
std::endl; // 应该输出 No

    // 继续出队
    std::cout << "Dequeue: " << lq.dequeue() << std::endl; // 应该输出 3

    // 再次检查队列是否为空
```

```
    std::cout << "Is queue empty? " << (lq.isEmpty() ? "Yes" : "No") <<
std::endl; // 应该输出 Yes

    return 0;
}
```

```
➜ output git:(main) X ./"linked_queue"
Dequeue: 1
Dequeue: 2
Is queue empty? No
Dequeue: 3
Is queue empty? Yes
```