**姓名: 张景耀 学号: 3200602123 班级: 计算机2005**

### 实验内容1. 设计两个顺序栈共享一个存储空间：

完成共享栈的入栈、出栈和判断栈空等三个函数

#### 源代码

#include<bits/stdc++.h>

using namespace std;

const int DEFAULT\_SIZE = 10;

template<class T>

class DoubleSeqStack

{

protected:

    //fields:

    int t1, t2;

    int size;

    T\* data;

public:

    //methods:

    DoubleSeqStack(int size = DEFAULT\_SIZE)

    {

        this->data = new T[size];

        this->size = size;

        this->t1 = -1, this->t2 = size;

    }

    void push1(T val)

    {

        if(this->t1 + 1 >= t2)

            return;

        this->data[++t1] = val;

    }

    void push2(T val)

    {

        if(this->t2 - 1 <= t1)

            return;

        this->data[--t2] = val;

    }

    T pop1()

    {

        if(isEmpty1())

            return INT\_MIN;

        return this->data[t1--];

    }

    T pop2()

    {

        if(isEmpty2())

            return INT\_MIN;

        return this->data[t2++];

    }

    bool isEmpty1()

    {

        return this->t1 == -1;

    }

    bool isEmpty2()

    {

        return this->t2 == size;

    }

    void traverse(void (\*visit)(T val))

    {

        printf("The First Stack:\n");

        for(int i = t1; i >= 0; )

            visit(this->data[i--]);

        printf("\nThe Second Stack:\n");

        for(int i = t2; i < size; )

            visit(this->data[i++]);

        puts("");

    }

    ~DoubleSeqStack()

    {

        delete[] this->data;

    }

};

#### 测试函数

template<typename T>

void visit(T val)

{

    cout << val << " ";

}

int main(int argc, char const \*argv[])

{

    auto doubleStk = DoubleSeqStack<int>(DEFAULT\_SIZE);

    printf("Push 1-5 in the First Stack && 6-10 in the Second Stack:\n\n");

    for(int i = 0; i < 5; i++)

        doubleStk.push1(i + 1), doubleStk.push2(6 + i);

    doubleStk.traverse(visit);

    puts("");

    printf("Push 999 into the First Stack which is already Full:\n\n");

    doubleStk.push1(9999);

    doubleStk.traverse(visit);

    printf("Nothing happend\n\n");

    printf("Pop everything in the Second Stack:\n\n");

    while(!doubleStk.isEmpty2())

        cout << doubleStk.pop2() << " has been poped\n";

    puts("");

    doubleStk.traverse(visit);

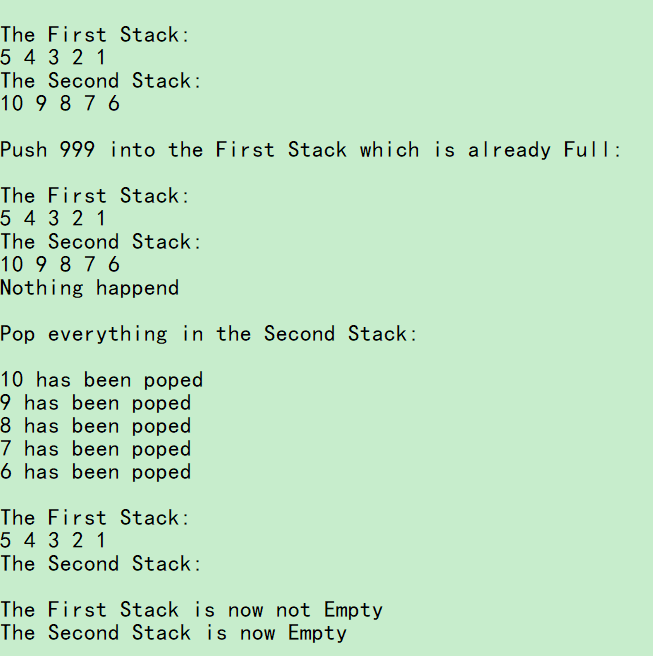
    cout << "The First Stack is now " << (doubleStk.isEmpty1() ? "Empty" : "not Empty") << endl;

    cout << "The Second Stack is now " << (doubleStk.isEmpty2() ? "Empty" : "not Empty") << endl;

    return 0;

}

#### 测试结果



### 实验内容2. 设计利用两个栈sl，s2模拟一个队列：

请写出实现入队、出队和判队列空的函数的实现

#### 源代码

#include<bits/stdc++.h>

using namespace std;

const int DEFAULT\_SIZE = 10;

template<class T>

class Stack

{

public:

    //fields:

    T\* data;

    int size;

    int top;

public:

    //methods:

    Stack(int size = DEFAULT\_SIZE) : size(size), top(-1)

    {

        this->data = new T[size];

    }

    Stack(const Stack& s)

    {

        \*this = s;

        this->data = new T[size];

        for(int i = 0; i <= top; i++)

            data[i] = s.data[i];

    }

    bool isEmpty()

    {

        return this->top == -1;

    }

    void push(T val)

    {

        if(this->top == size)

            return;

        this->data[++top] = val;

    }

    T pop()

    {

        if(isEmpty())

            return INT\_MIN;

        return data[top--];

    }

};

template<class T>

class Queue

{

protected:

    //fields:

    Stack<T> \*stk1, \*stk2;

    int size, len;

public:

    //methods:

    Queue(int size = DEFAULT\_SIZE) : size(size), len(0)

    {

        this->stk1 = new Stack<T>(size);

        this->stk2 = new Stack<T>(size);

    }

    Queue(const Queue& q)

    {

        \*this = q;

        this->stk1 = new Stack<T>(\*q.stk1);

        this->stk2 = new Stack<T>(\*q.stk2);

    }

    bool isEmpty()

    {

        return stk1->isEmpty() and stk2->isEmpty();

    }

    void push(T val)

    {

        if(this->len == this->size)

            return;

        this->stk1->push(val);

        this->len++;

    }

    T pop()

    {

        if(stk2->isEmpty())

            while(!stk1->isEmpty())

                stk2->push(stk1->pop());

        if(stk2->isEmpty())

            return INT\_MIN;

        else

        {

            this->len--;

            return stk2->pop();

        }

    }

    void traverse(void (\*visit) (T val))

    {

        printf("The Queue: ");

        Queue<T> q(\*this);

        while (!q.isEmpty())

            visit(q.pop());

        puts("");

    }

    ~Queue()

    {

        delete this->stk1;

        delete this->stk2;

    }

};

#### 测试函数

template<typename T>

void visit(T val)

{

    cout << val << " ";

}

int main(int argc, char const \*argv[])

{

    auto q = Queue<int>(DEFAULT\_SIZE);

    cout << "The Queue is now " << (q.isEmpty() ? "Empty\n" : "not Empty\n") << endl;

    printf("After Push 1-15 into the Queue:\n\n");

    for(int i = 1; i <= 15; i++)

        q.push(i);

    q.traverse(visit);

    cout << "\nThe Queue is now " << (q.isEmpty() ? "Empty\n" : "not Empty\n") << endl;

    printf("After Pop 5 Elements from the Queue\n\n");

    for(int i = 0; i < 5; i++)

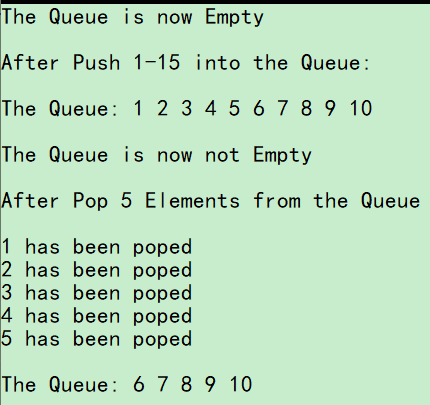
        cout << q.pop() << " has been poped" << endl;

    puts("");

    q.traverse(visit);

}

#### 测试结果



### 实验内容3. 设计循环队列：

以front和length分别表示循环队列中的队头位置和队列中所含元素的个数.

试完成循环队列判断队空、判断队满入队和出队函数的实现。

#### 源代码

#include<bits/stdc++.h>

using namespace std;

const int DEFAULT\_SIZE = 10;

#define SUB(x) (x) % this->size

template<class T>

class CircularQueue

{

protected:

    //fields:

    int length;

    int front, rear;

    int size;

    T\* data;

public:

    //methods:

    CircularQueue(int size = DEFAULT\_SIZE) : size(size), length(0), front(0), rear(0)

    {

        this->data = new T[size];

    }

    bool isEmpty()

    {

        return this->rear == this->front;

    }

    bool isFull()

    {

        return SUB(rear + 1) == front;

    }

    void push(T val)

    {

        if(isFull())

            return;

        this->data[SUB(this->rear++)] = val;

        this->rear %= size;

        this->length++;

    }

    T pop()

    {

        if(this->isEmpty())

            return INT\_MIN;

        T ans = this->data[SUB(this->front++)];

        this->front %= size;

        this->length--;

        return ans;

    }

    void traverse(void (\*visit) (T val))

    {

        for(int i = 0; i < this->length; i++)

            visit(this->data[SUB(i + front)]);

        puts("");

    }

    ~CircularQueue()

    {

        delete[] this->data;

    }

};

#### 测试函数

template<typename T>

void visit(T val)

{

    cout << val << " ";

}

int main(int argc, char const \*argv[])

{

    auto q = CircularQueue<int>(DEFAULT\_SIZE);

    cout << "The Queue is now " << (q.isEmpty() ? "Empty\n" : "not Empty\n") << endl;

    printf("After Push 1-15 into the Queue:\n\n");

    for(int i = 1; i <= 15; i++)

        q.push(i);

    q.traverse(visit);

    cout << "\nThe Queue is now " << (q.isEmpty() ? "Empty\n" : "not Empty\n") << endl;

    printf("After Pop 5 Elements from the Queue\n\n");

    for(int i = 0; i < 5; i++)

        cout << q.pop() << " has been poped" << endl;

    puts("");

    q.traverse(visit);

    printf("\nAfter Push 10-20 into the Queue:\n\n");

    for(int i = 10; i <= 20; i++)

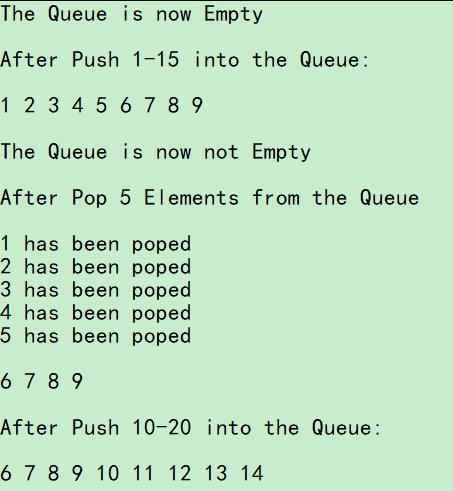
        q.push(i);

    q.traverse(visit);

    return 0;

}

#### 测试结果



### 实验内容4. 设二项式(a+b)n展开后，其系数构成杨辉三角形，写出利用队列实现打印杨辉三角形的前n行的算法：

#### 源代码

#include<bits/stdc++.h>

using namespace std;

void YHTriangle(int n)

{

    printf("YangHui Triangle with %d floor:\n", n);

    cout << R"(

1

1 1

)";

    queue<int> q({1, 1, 1});

    for(int i = 1; i < n - 1; i++, puts(""))

    {

        //Remove duplicated 1

        q.pop();

        printf("1 ");

        q.push(1);

        for(int j = 0; j < i; j++)

        {

            auto t = q.front();

            q.pop();

            t += q.front();

            cout << t << " ";

            q.push(t);

        }

        printf("1 ");

        q.push(1);

    }

}

#### 测试函数

int main(int argc, char const \*argv[])

{

    int n;

    scanf("%d", &n);

    YHTriangle(n);

    return 0;

}

#### 测试结果

