沈阳航空航天大学

人工智能学院

**实验报告**

|  |  |
| --- | --- |
| 课程名称 | 算法导论 |
| 专 业 | 物联网工程 |
| 班 级 | 物联网2202 |
| 学 号 | 223428010210 |
| 学生姓名 | 陈梓欣 |
| 指导教师 | 孙恩岩 |
| 实验时间 | 2024年5月26日 3、4节 |
| 实验地点 | 机械馆410-3 |

# 一、实验名称

栈的基本操作和应用

# 二、实验目的

深入理解栈的存储结构，熟练掌握栈的基本操作。

# 三、实验内容和要求

将一组数据连续压入栈，所有元素都入栈后，再连续的将栈中的所有元素依此出栈，将会得到和原数据序列排列相反的一组数据，即利用栈实现逆序操作。

# 四、实验环境

Microsoft Visual Studio 2022

# 五、实验设计

本次实验主要涉及算法和数据结构设计，重点在于栈（Stack）这一数据结构的设计和实现。以下是实验中涉及的关键点：

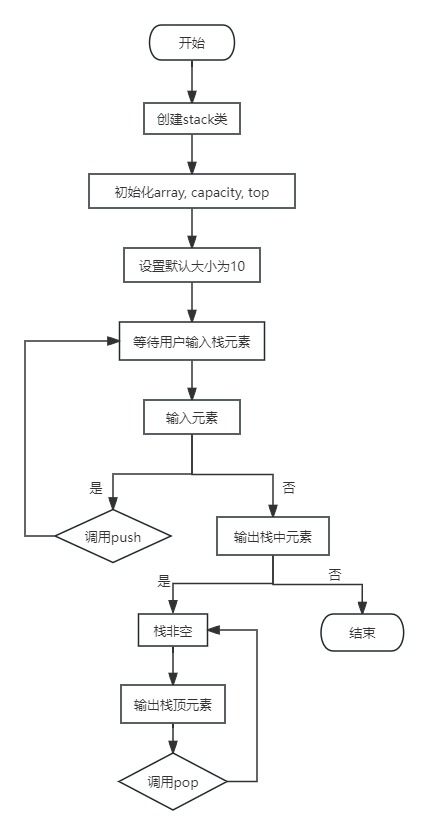
1. 算法和数据结构设计：

- 栈的基本操作：包括`push`（入栈）、`pop`（出栈）和`peek`（查看栈顶元素）。

- 栈的扩容和缩容：当栈满时自动扩容，当栈空闲空间过多时自动缩容，以优化内存使用。

- 动态内存管理：使用指针和动态内存分配来管理栈的内存。

- 栈的实现：使用数组作为栈的底层数据结构，通过指针来追踪栈顶位置。



**图1 主模块流程图**

程序代码要加入必要注释，程序示例如下：

#include<iostream>

using namespace std;

class Stack

{

private:

//储存栈元素的数组

int\* array;

//栈的容量

size\_t capacity;

//栈顶位置

size\_t top;

public:

//构造函数

Stack(size\_t size = 10) : capacity(size), top(0)

{

array = new int[capacity];

}

//析构函数

~Stack()

{

delete[] array;

}

//入栈

void push(int value)

{

if (top == capacity)

{

//栈满 需扩容

int\* newArray = new int[capacity \* 2];

for (size\_t i = 0; i < capacity; ++i)

{

newArray[i] = array[i];

}

delete[] array;

array = newArray;

capacity \*= 2;

//cout << "the new capacity of the stack: " << capacity << endl;

}

array[top++] = value;

}

//出栈

void pop()

{

if (isEmpty())

{

cerr << "栈为空，无法执行出栈操作" << endl;

return;

}

--top;

// 如果栈利用率低于25%，则缩容

if (top > 0 && top == capacity / 4)

{

int\* newArray = new int[capacity / 2];

for (std::size\_t i = 0; i < top; ++i)

{

newArray[i] = array[i];

}

delete[] array;

array = newArray;

capacity /= 2;

//cout << "the new capacity of the stack: " << capacity << endl;

}

}

// 获取栈顶元素

int& peek() {

if (isEmpty()) {

std::cerr << "栈为空" << std::endl;

throw std::runtime\_error("栈为空");

}

return array[top - 1];

}

// 检查栈是否为空

bool isEmpty() const {

return top == 0;

}

// 获取栈的大小

std::size\_t size() const {

return top;

}

};

int main()

{

//size\_t capacity;

//cout << "Enter the initial capacity of the stack: ";

//cin >> capacity;

//Stack myStack(capacity);

Stack myStack;

int value;

cout << "Enter the stack: ";

while (cin >> value)

{

myStack.push(value);

}

cout << "Elements in the stack (output in reverse order): ";

while (!myStack.isEmpty())

{

cout << myStack.peek() << " ";

myStack.pop();

}

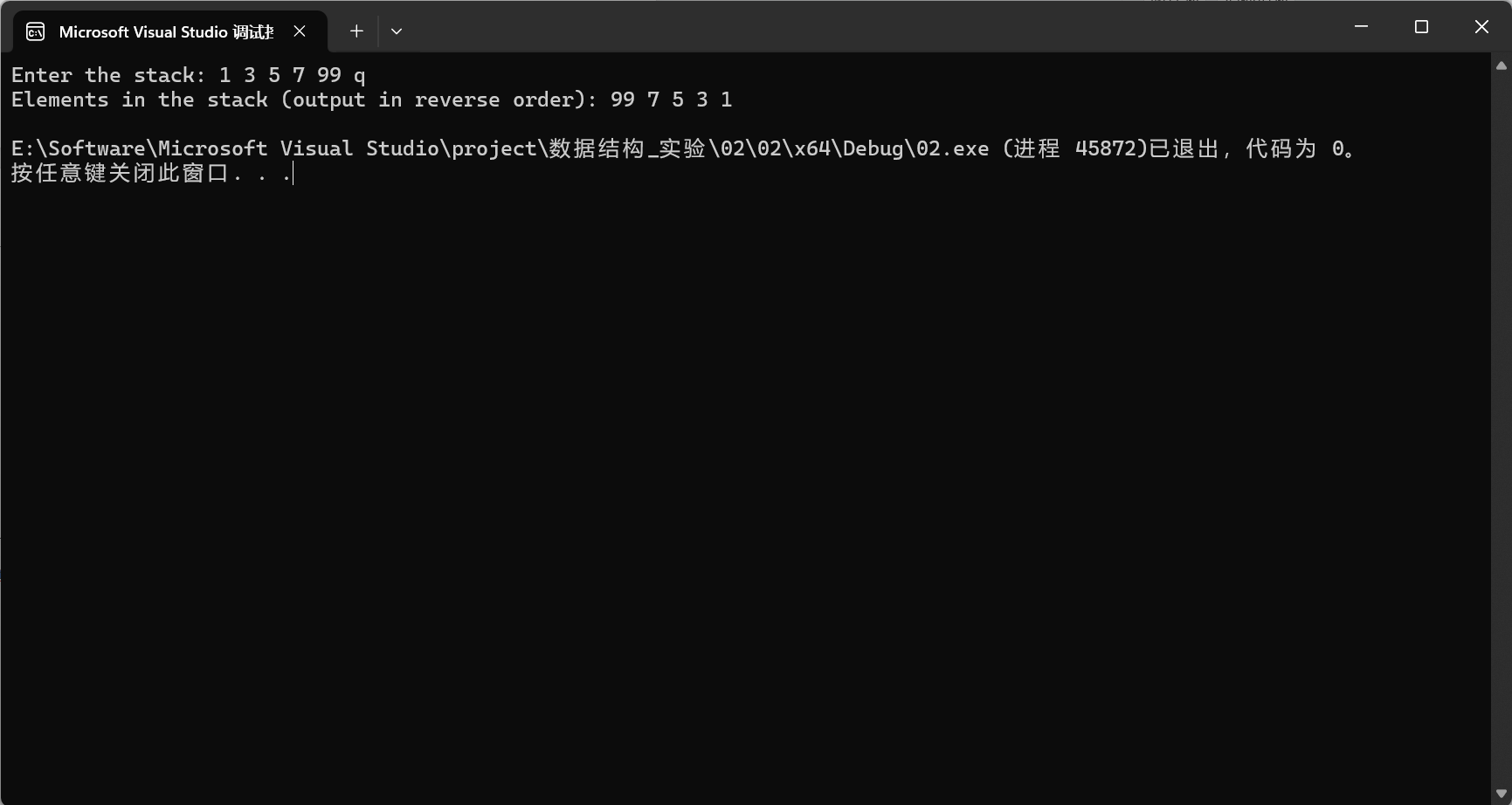
cout << endl;

return 0;

}

# 六、实验步骤及实验结果

本节用文字和屏幕截图详实记录实验的设计结果、完成过程（步骤）和每一步的测试结果



**图2 打印信息**

# 七、结论

本次实验通过手动实现栈数据结构，我深入理解了栈的存储结构和基本操作。实验中，我们完全脱离了标准库的容器，自己管理内存和容量，这使我更直观地理解了栈的工作原理。

实验结果验证了栈的先进后出（LIFO）原则、动态内存管理的重要性、扩容和缩容的必要性以及异常处理在编程中的重要性。

我从中归纳出的一般性、概括性的判断是：栈是一种非常实用的数据结构，它的先进后出特性在很多应用场景中非常有用。动态内存管理是实现高效栈的关键，能够自动扩容和缩容的栈可以更好地适应数据的变化。在实现栈时，考虑异常情况并妥善处理是非常重要的，这有助于提高程序的健壮性。