****

**数据结构**

**实 验 报 告**

实验名称：实验二 栈和队列

实验学时：2

实验环境：dev511

学生学院：信息科学与技术学院/网络空间安全学院

专业班级：计算机2101

学 号：20211003153

学生姓名：赖永超

指导教师：周咏梅

学 年： 2021-2022第二学期

2022 年 4月 5日

目录

[一、实验目的 3](#_Toc100395250)

[二、实验内容 3](#_Toc100395251)

[1. 银行业务队列简单模拟 3](#_Toc100395252)

[2. 后缀表达式求值 10](#_Toc100395253)

[3.表达式求值 16](#_Toc100395254)

[三、实验总结 24](#_Toc100395255)

**实验二 栈和队列**

姓名：赖永超 班级：计算机2101 学号：20211003153

## 一、实验目的

1.熟练掌握栈和队列的基本操作，理解栈和队列的应用。

## 二、实验内容

### 1. 银行业务队列简单模拟

1）实验题目3-7-6 银行业务队列简单模拟

2）问题描述

设某银行有A、B两个业务窗口，且处理业务的速度不一样，其中A窗口处理速度是B窗口的2倍 —— 即当A窗口每处理完2个顾客时，B窗口处理完1个顾客。给定到达银行的顾客序列，请按业务完成的顺序输出顾客序列。假定不考虑顾客先后到达的时间间隔，并且当不同窗口同时处理完2个顾客时，A窗口顾客优先输出。

输入格式:

输入为一行正整数，其中第1个数字N(≤1000)为顾客总数，后面跟着N位顾客的编号。编号为奇数的顾客需要到A窗口办理业务，为偶数的顾客则去B窗口。数字间以空格分隔。

输出格式:

按业务处理完成的顺序输出顾客的编号。数字间以空格分隔，但最后一个编号后不能有多余的空格。

输入样例:

8 2 1 3 9 4 11 13 15

输出样例:

1 3 2 9 11 4 13 15

3）算法分析

①数据结构

使用两个队列a和b，存储在A、B窗口排队的顾客。我使用了C++中的STL标准模板库。定义代码见图1-1。

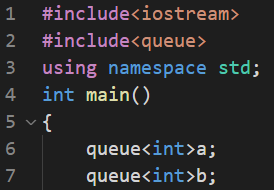


图1-1 a、b队列的定义

②功能描述

输入进入银行的顾客顺序后，根据题目设定，按业务完成顺序输出顾客顺序。

③算法描述

A.输入

根据题意，先输入总人数n，然后输入n个顾客进入银行的顺序。在输入顾客的同时，根据判断顾客是奇数还是偶数，分配到对应的窗口队伍中。代码见图1-2。

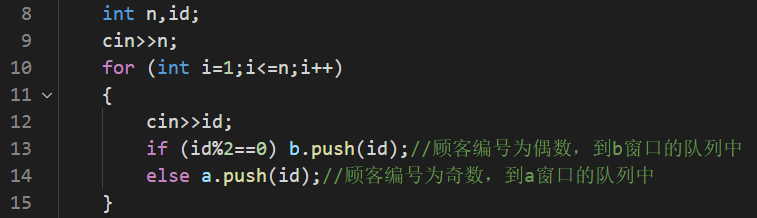


图1-2 输入信息，分配队列

B.模拟业务处理

定义一整型变量t，记录时间。当两个队列都不为空时，循环模拟出队。每循环一次，t=t+1。对任何的t，只要a队列不为空，就出队一个。当t为偶数且b队列不为空，b队列出队一个。当a和b队列都为空时，循环结束，程序结束。代码见图1-3。

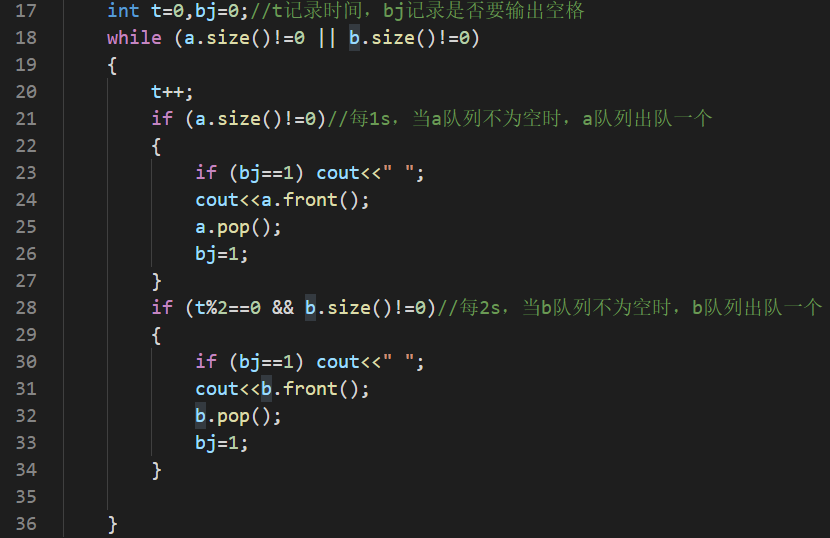


图1-3 模拟循环出队

4）具体代码

GitHub：[点击此处](https://github.com/Wilson-LYC/Data-Structure/blob/main/DS-2%20%E6%A0%88%E5%92%8C%E9%98%9F%E5%88%97/sy2_1_1_2101_20211003153_%E8%B5%96%E6%B0%B8%E8%B6%85.cpp" \o "点击打开GitHub网页)

文件：sy2\_1\_1\_2101\_20211003153\_赖永超.cpp（文件编码：ANSI，推荐使用Dev C++打开）

5）运行结果

①PTA提交结果，图1-4。

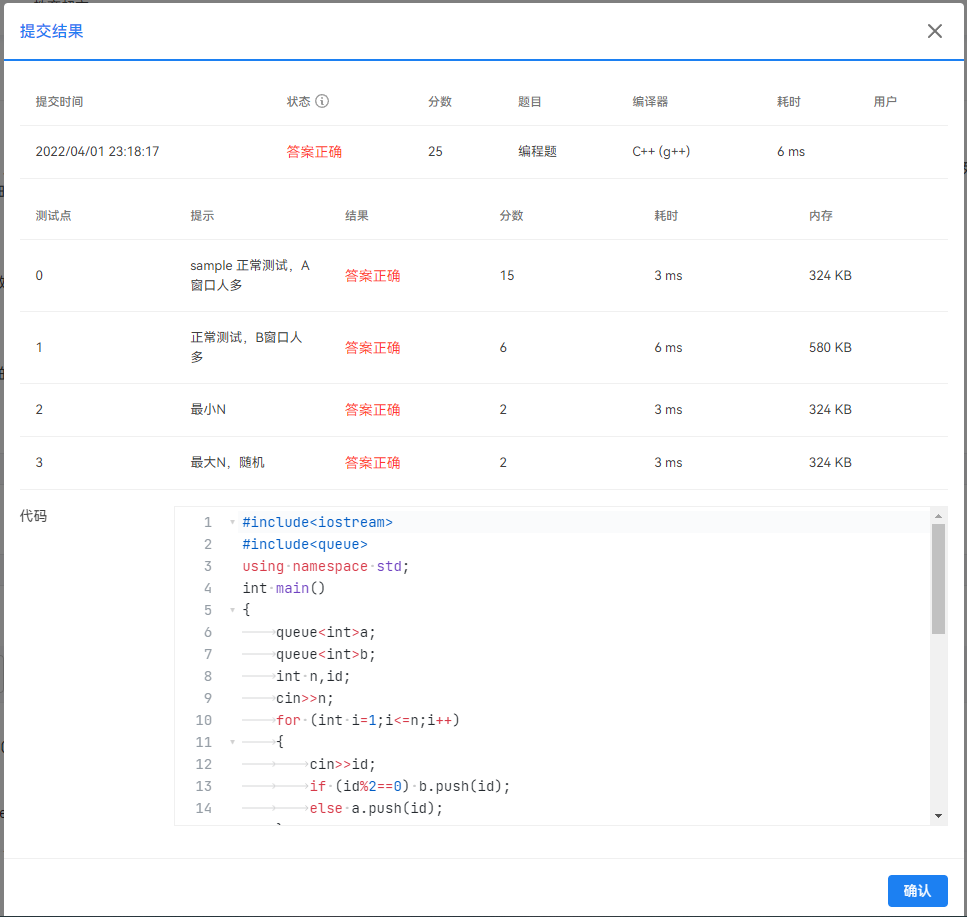


图1-4 PTA 银行业务队列简单模拟提交结果

②Dev C++运行截图

具体运行截图见图1-5，图1-6和图1-7。



图1-5 银行业务队列简单模拟 运行截图①



图1-6 银行业务队列简单模拟 运行截图②



图1-7 银行业务队列简单模拟 运行截图③

6）程序运行结果分析

程序运行正常，耗时正常，内存占用低。

7）功能扩展

①增加提示界面

A.功能描述

在输入数字前，输出中文提示

B.算法描述

在每个输入前，输出中文提示信息。

C.代码实现

核心代码见图1-8。

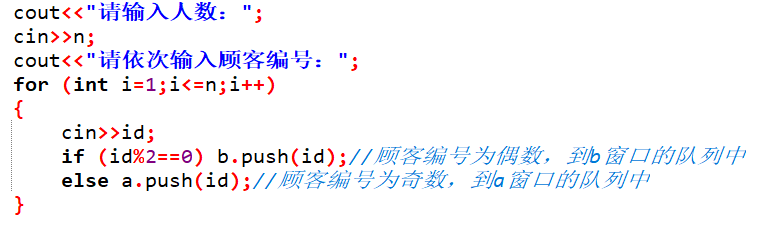


图1-8 中文提示信息

详细代码

GitHub：[点击此处](https://github.com/Wilson-LYC/Data-Structure/blob/main/DS-2%20%E6%A0%88%E5%92%8C%E9%98%9F%E5%88%97/sy2_1_2_2101_20211003153_%E8%B5%96%E6%B0%B8%E8%B6%85.cpp" \o "点击打开GitHub网页)

文件：sy2\_1\_2\_2101\_20211003153\_赖永超.cpp（文件编码：ANSI，推荐使用Dev C++打开）

D.运行结果，见图1-9，图1-10和图1-11。



图1-9 中文提示运行结果①



图1-10 中文提示运行结果②



图1-11 中文提示运行结果③

②文件输入

A.功能描述

采用文件输入的方式输入。

B.算法描述

检测根目录下是否又文件“in.txt”,如果没有则输出报错信息，否则读入信息进行计算。

C.代码实现

核心代码见图1-12。

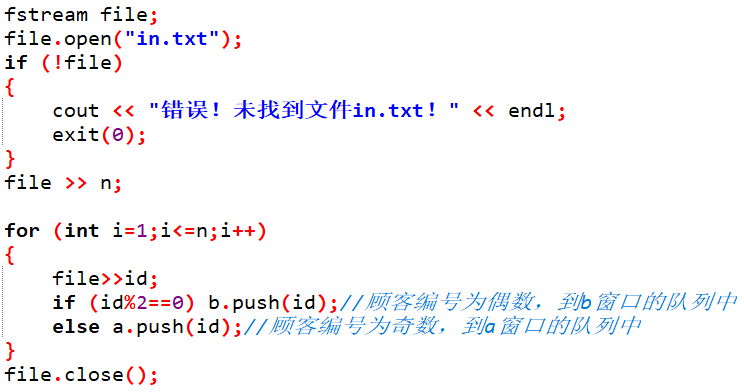


图1-12 银行业务队列简单模拟 文件输入

详细代码

GitHub：[点击此处](https://github.com/Wilson-LYC/Data-Structure/blob/main/DS-2%20%E6%A0%88%E5%92%8C%E9%98%9F%E5%88%97/sy2_1_3_2101_20211003153_%E8%B5%96%E6%B0%B8%E8%B6%85.cpp" \o "点击打开GitHub网页)

文件：sy2\_1\_3\_2101\_20211003153\_赖永超.cpp（文件编码：ANSI，推荐使用Dev C++打开）

D.运行结果，见图1-13和图1-14



图1-13 银行业务队列简单模拟 文件输入运行结果①



图1-14 银行业务队列简单模拟 文件输入运行结果②

### 2. 后缀表达式求值

1)实验题目3-7-5 逆波兰表达式求值

2）问题描述

逆波兰表示法是一种将运算符（operator）写在操作数（operand）后面的描述程序（算式）的方法。举个例子，我们平常用中缀表示法描述的算式（1 + 2）\*（5 + 4），改为逆波兰表示法之后则是1 2 + 5 4 + \*。相较于中缀表示法，逆波兰表示法的优势在于不需要括号。

请输出以逆波兰表示法输入的算式的计算结果。

输入格式:

在一行中输入1个算式。相邻的符号（操作数或运算符）用1个空格隔开。

输出格式:

在一行中输出计算结果。

限制:2≤算式中操作数的总数≤100，1≤算式中运算符的总数≤99

运算符仅包括“+”、“-”、“\*”，操作数、计算过程中的值以及最终的计算结果均在int范围内。

输入样例1:

4 3 + 2 -

输出样例1:

5

输入样例2:

1 2 + 3 4 - \*

输出样例2:

-3

3）算法分析

①数据结构

用一个栈存储数字。见图2-1。

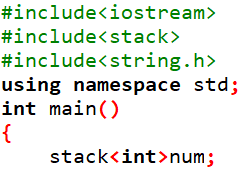


图2-1 数字栈的定义

②功能描述

计算后缀表达式（逆波兰表达式）

③算法描述

当遇到数字时，将数字进栈。遇到计算符号时，出栈两个数字，识别计算符号后进行计算，将结果入栈。

A.输入逆波兰表达式

使用字符串存储表达式。代码见图2-2。

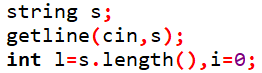


图2-2 逆波兰表达式的存储

B．遇到数字，识别数字并入栈

当遇到一个数字时，往后遍历，直到不为数字。代码见图2-3

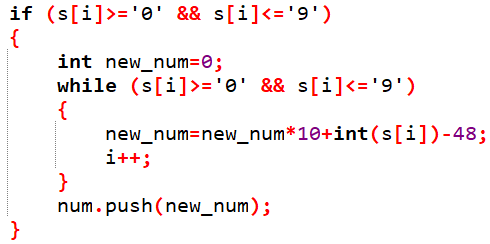


图2-3 数字识别与入栈

C．遇到计算符号，进行计算

遇到计算符号后，用swithc()语句判断进行什么运算。运算后将结果入栈，继续进行计算。具体代码见图2-4。

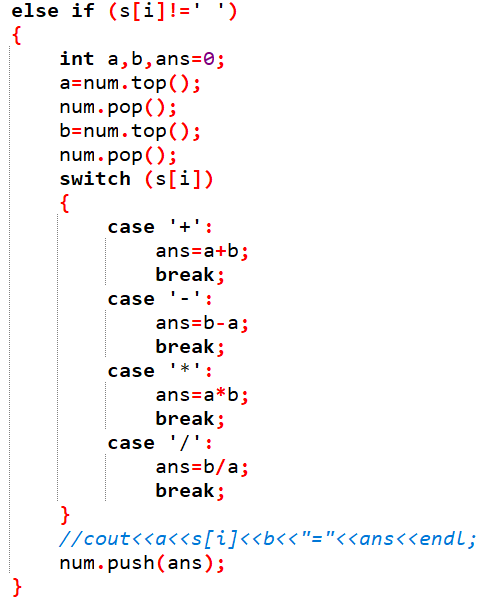


图2-4 计算符号的识别、计算与结果入栈

4）具体代码

GitHub：[点击此处](https://github.com/Wilson-LYC/Data-Structure/blob/main/DS-2%20%E6%A0%88%E5%92%8C%E9%98%9F%E5%88%97/sy2_2_1_2101_20211003153_%E8%B5%96%E6%B0%B8%E8%B6%85.cpp" \o "点击打开GitHub网页)

文件：sy2\_2\_1\_2101\_20211003153\_赖永超.cpp（文件编码：ANSI，推荐使用Dev C++打开）

5）运行结果

①PTA提交结果，图2-5。

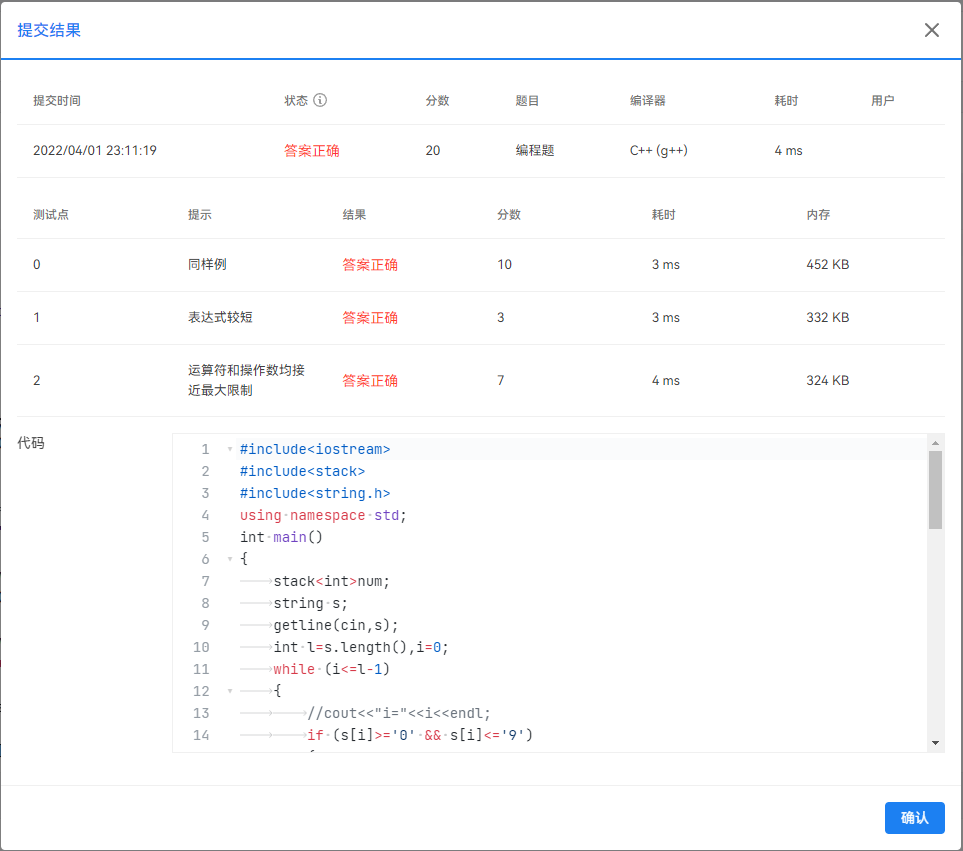


图2-5 逆波兰表达式PTA提交结果

②Dev C++运行结果

具体运行结果见图2-6，图2-7和图2-8。



图2-6 逆波兰表达式 运行结果①



图2-7 逆波兰表达式 运行结果②



图2-8 逆波兰表达式 运行结果③

6）程序运行结果分析

程序运行正常，计算结果正确，耗时正常，内存占用低，未发现错误。

### 3.表达式求值

1)实验题目pta- 2020DS-ch3练习题-zym-编程题2-7-4 表达式求值\_1

2）问题描述

在一个表达式中，只有“(”，“)”，“0-9”，“+”，“-”，“\*”，“/”，“^”，请求出表达式的值。(“/”用整数除法)。

输入格式:

共1 行，为一个算式。 （算式长度<=30 其中所有数据在 0~2^31-1的范围内）。

输出格式:

共一行，为表达式的值。

输入样例:

在这里给出一组输入。例如：

1+(3+2)\*(7^2+6\*9)/(2)

输出样例:

在这里给出相应的输出。例如：

258

3）算法分析

①数据结构

用一个栈存储数字，一个栈存储运算符号。见图3-1。



图3-1 数字栈于运算符栈的定义

②功能描述

计算只有“(”，“)”，“0-9”，“+”，“-”，“\*”，“/”，“^”的表达式

③算法描述

首先读入整个表达式，然后从字符串头遍历到字符串尾。遇到数字，则将数字入数字栈。遇到符号，判断当前的运算符与栈顶的运算符谁的计算级别高。如果当前运算符级别高，则将运算符入运算符栈，继续往后遍历。若当前运算符级别低于栈顶运算符级别，则开始往回计算栈内的数字。每计算一次，运算符栈出栈一个运算符，数字栈出栈两个数字，计算完成后结果入栈数字栈。直到当前的运算符界别比栈顶运算符高时，结束计算。当字符串遍历完成后，遍历数字栈和运算符栈，结算最终的结果。

A.输入表达式

使用字符串存储表达式。代码见图3-2。

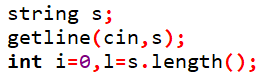


图3-2表达式的存储

B．遇到数字，识别数字并入栈

当遇到一个数字时，往后遍历，直到不为数字。代码见图3-3

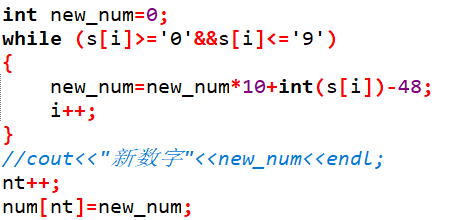


图3-3 数字入数字栈

C．遇到计算符号，进行识别

遇到运算符时，要做一下判断。

a．是否是”(“，如果是则直接入符号栈。代码见图3-4。

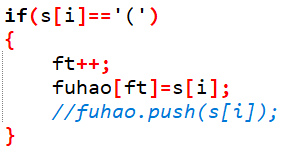


图3-4 遇到“（”，直接入符号栈

b．是否是“）”，如果是则可以开始往回计算，将括号内的内容先计算出来。代码见图3-5。

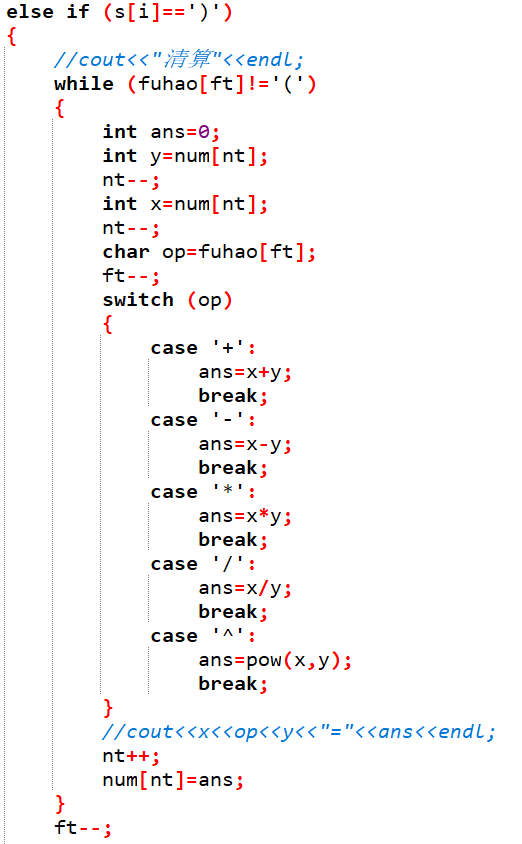


图3-5 遇到“）”，计算括号内内容

c．遇到其它的运算符

c.1若当前运算符栈为空，直接入栈。代码见图3-6。

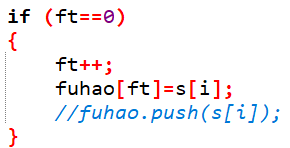


图3-6 运算符栈为空，直接入栈

c.2 若当前运算符级别比运算符栈栈顶的级别高，直接入栈，代码见图3-7。

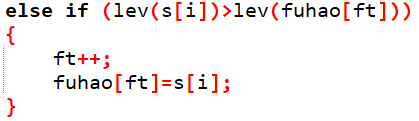


图3-6 运算符级别高，直接入栈

c.3 若当前运算符界别低于运算符栈栈顶的界别，则可以进行计算。直到当前运算符界别高于运算符栈栈顶的级别，计算完成后当前运算符入运算符栈。代码见图3-7。

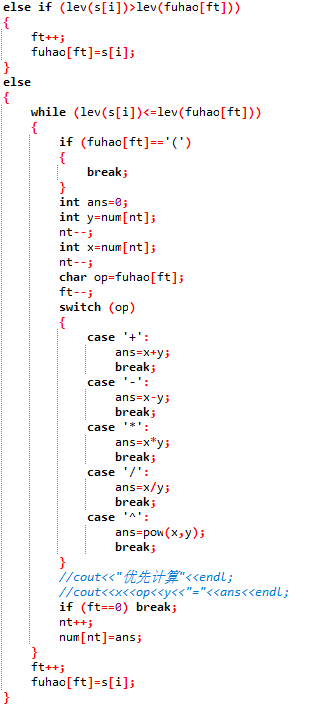


图3-7 运算符界别判断和计算

c.3 遍历完成字符串后，遍历栈，计算最后的结果。代码见图3-8。

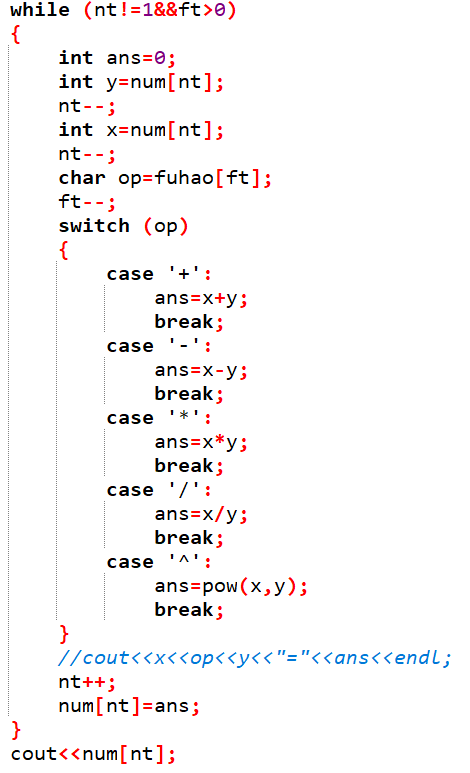


图3-8 计算最后结果

4）具体代码

GitHub：[点击此处](https://github.com/Wilson-LYC/Data-Structure/blob/DS/DS-2%20%E6%A0%88%E5%92%8C%E9%98%9F%E5%88%97/sy2_3_1_2101_20211003153_%E8%B5%96%E6%B0%B8%E8%B6%85.cpp" \o "点击打开GitHub网页)

文件：sy2\_3\_1\_2101\_20211003153\_赖永超.cpp（文件编码：ANSI，推荐使用Dev C++打开）

5）运行结果

①PTA提交结果，图3-9。



图3-9 表达式求值\_1 PTA提交结果

②Dev C++运行结果

具体运行结果见图3-10，图3-11和图3-12。



图3-10 表达式求值\_1 运行结果①



图3-11 表达式求值\_1 运行结果②



图3-12 表达式求值\_1 运行结果③

6）程序运行结果分析

程序运行正常，计算结果正确，耗时正常，内存占用低，未发现错误。

## 三、实验总结

在本次实验中，主要使用了栈和队列两种数据结果解决问题。在栈和队列的是实现上，采用了C++ STL标准库和顺序表两种形式。使用STL解决表达式求值时间，会出现第二个数据点“段错误”的问题。使用顺序表则可以正常通过该点。使用STL调用queue和stack非常的方便，不需要考虑容量大小和编写相关的操作函数，直接使用即可，它的函数也是比较好理解的。在使用顺序表实现队列和栈时，需要定义相关的“下标指针”，用于记录栈顶或者是对头和队尾。使用顺序表来存储队列，会出现a[1]-a[对头]这以一部分的空间被浪费。当遇到数据量比较大的问题时，需要定义非常长的队列，这也带来了超内存的风险。此时，可以使用STL或者是使用链表来实现队列。