南昌大学

实　验　报　告

实验课程：数据结构与算法实验

实验题目：栈与队列验证算术表达式的正确性

任课老师：吴之旭老师

学生姓名：殷骢睿

学　　号：5509121041

专业班级：2021 级人工智能实验班

2022 年 11 月 1 日

南昌大学实验报告

学生姓名：殷骢睿　学号：5509121041　专业班级：人工智能 211 班
实验类型：☐ 验证 ☐ 综合 ■ 设计 ☐ 创新　实验日期：2022.11.1　实验成绩：

一、 实验目的

(1) 掌握栈与队列的定义；
(2) 掌握栈与队列的基本操作；

二、 实验内容

1. 验证某算术表达式的正确性，若正确，则计算该算术表达式的值。
(1) 从键盘上输入表达式的值；
(2) 分析该表达式是否合法：
（a）是数字，则判断该数字的合法性。若合法，则压入数据存放堆栈
（b）是规定的运算符，则根据规则进行处理。在处理过程中，将计算该表达式的值。
（c）若是其它字符，则返回错误信息。
(3) 若上述处理过程中没有发现错误，则认为该表达式合法，并打印处理结果；
2. 表达式的计算过程：
1、变量直接输出；
2、遇到“（”，无条件入栈；
3、输入运算优先数是 2，3，4，5，时，如栈空则入栈，否则，将其与栈顶元素进行比较，如果优先级大，就进栈，否则就退栈输出；
4、如遇“）”，同时栈顶元为“（”，则退栈，抹去括号。

三、实验要求

(1) 程序要添加适当的注释，程序的书写要采用缩进格式。
(2) 程序要具在一定的**健壮性**，即当输入数据非法时，程序也能适当地做出反应。
(3) 程序要做到界面友好，在程序运行时用户可以根据相应的提示信息进行操作，比如如何结束一个多项式的输入。
(4) 根据实验报告模板详细书写实验报告,在实验报告中给出相应的流程图。
(5) 源程序和实验报告打包。实验报告命名为：实验报告 3.doc。源程序和实验报告压缩为一个文件（如果定义了头文件则一起压缩），按以下方式命名：学号姓名.rar，如 0814101 王五.rar。

四、主要实验步骤及程序分析

4.1 功能实现操作界面搭建

如果是数字，首先判断数字的合法性，如果数字是合法的，可以点击“1”进行入栈操作，如果是能够匹配的（如“（）”），则退栈并且进行判断，同时抹去括号，该 UI 的优点是人机交互友好，可以提示“栈已满”、“栈已空”等信息，以便于用户进行表达式的输入。
（1）功能实现函数

```
1. //*功能实现函数
```

```
void menu()
{
    printf("********1.入栈       2.出栈*********\n");
    printf("********3.取栈顶     4.退出*********\n");
}
//入栈功能函数 调用 Push 函数
void PushToStack(SqStack &s)
{
    int n;SElemType e;int flag;
    printf("请输入入栈元素个数(>=1): \n");
    scanf("%d",&n);
    for(int i=0;i<n;i++)
    {
     printf("请输入第%d 个元素的值:",i+1);
     scanf("%d",&e);
     flag=Push(s,e);
     if(flag)printf("%d 已入栈\n",e);
     else {printf("栈已满！！！\n");break;}
    }
}
//出栈功能函数 调用 Pop 函数
void PopFromStack(SqStack &s)
{
    int n;SElemType e;int flag;
    printf("请输入出栈元素个数(>=1): \n");
    scanf("%d",&n);
    for(int i=0;i<n;i++)
    {
     flag=Pop(s,e);
     if(flag)printf("%d 已出栈\n",e);
     else {printf("栈已空！！！\n");break;}
    }
}
//取栈顶功能函数 调用 GetTop
void GetTopOfStack(SqStack &s)
{
    SElemType e;bool flag;
    flag=GetTop(s,e);
    if(flag)printf("栈顶元素为:%d\n",e);
    else printf("栈已空！！！\n");
}
```

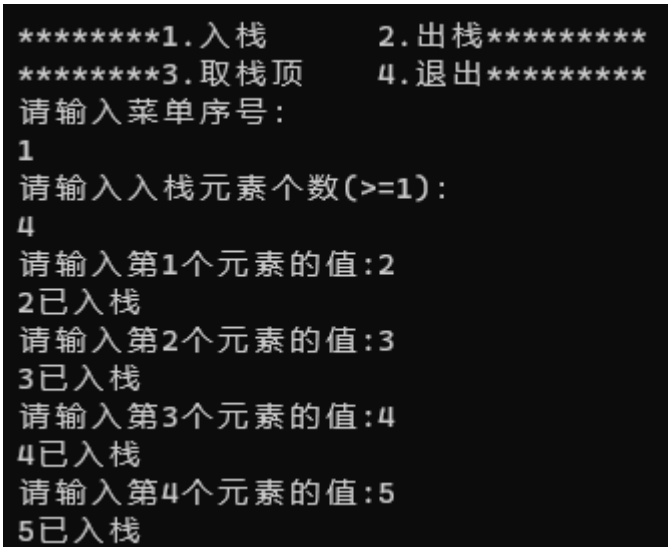


图 1 功能实现操作界面搭建示意图

（2）实现匹配

```
ool Match(char exp[],int n)//匹配
{
    int i=0;
    char e;
    bool match=true;
    LiStack *st;
    initStack(st);
    while(i<n&&match)//扫描所有字符
    {//小括号
        if(exp[i]=='('
            Push(st,exp[i]);//进栈
        else if(exp[i]==')')
        {
            if(GetTop(st,e)==true)
            {
                if(e!='('//栈顶不是（时不匹配
                    match=false;
                else
                    Pop(st,e);//匹配成功出栈
            }
            else match=false;//取不到栈顶元素时不匹配
```

图 2 匹配示意图

4.2 主要具体步骤（完整代码见附录）

(1) 所用数据结构的定义及其相关说明（相关结构体或类的定义及其含义）

```
1.  //定义一个栈类型的结构体
2.  typedef struct SqStack
3. {
4.     SElemType * base;         /* 在栈构造之前和销毁之后，base 的值为
   NULL */
5.     SElemType * top;          /* 栈顶指针*/
6.     int stacksize;            /* 当前已分配的存储空间，以元素为单位*/
7. }SqStack;
```

(2) 自定义函数的名称及其功能说明

```
1. Status InitStack(SqStack &S)   /* 构造一个空栈 S */
2. Status GetTop(SqStack &S, SElemType &e)  /* 若栈不空，则用 e 返回 S 的栈顶
   元素，并返回 e；否则返回 ERROR */
3. Status Push(SqStack &S, SElemType e)   /* 插入元素 e 为新的栈顶元素*/
4. Status Pop(SqStack &S, char &e) /* 若栈不空，则删除 S 的栈顶元素，用 e 返回
   其值，并返回 e；否则返回 ERROR */
5. char Precede(char t1,  char t2) /* 判断两符号的优先关系*/
6.     int In(char c) /* 判断 c 是否为运算符*/
7. char Operate(char a,  char theta,  char b) //进行运算的函数
8. char EvaluateExpression() //输入运算表达式并计算
```

(3) 主要功能算法的时间复杂度

```
1. Status InitStack(SqStack &S)                          O(1);
2. Status GetTop(SqStack &S, SElemType &e)             O(1);
3. Status Push(SqStack &S, SElemType e)                 O(1);
4. Status Pop(SqStack &S, char &e)                      O(1);
5. char Precede(char t1,  char t2)                       O(1);
6.     int In(char c)                                        O(1);
7. char Operate(char a,  char theta,  char b)            O(1);
8. char EvaluateExpression()                            O(n);
```

4.3 流程图

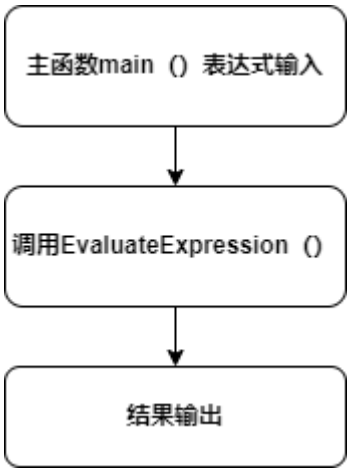


图 3 总体思路规划图

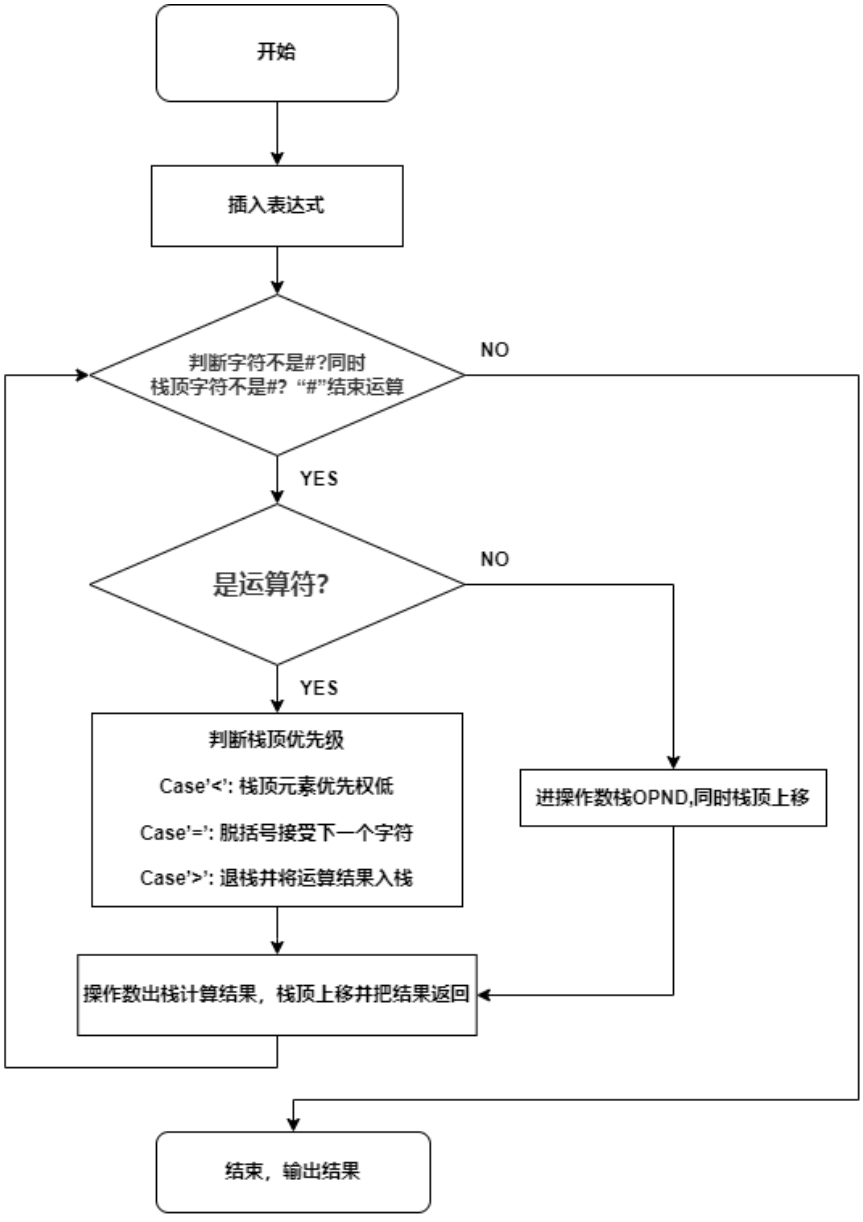
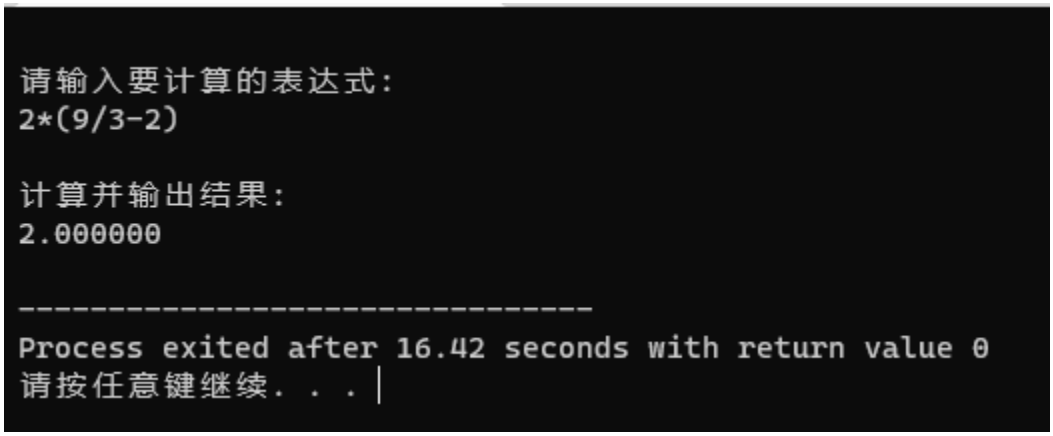


图 4 利用栈验证算术表达式正确性完整逻辑流程图

4.4 程序结果

输入：需要计算的表达式

输出：相加多项式，各多项式的项式和最高次幂。



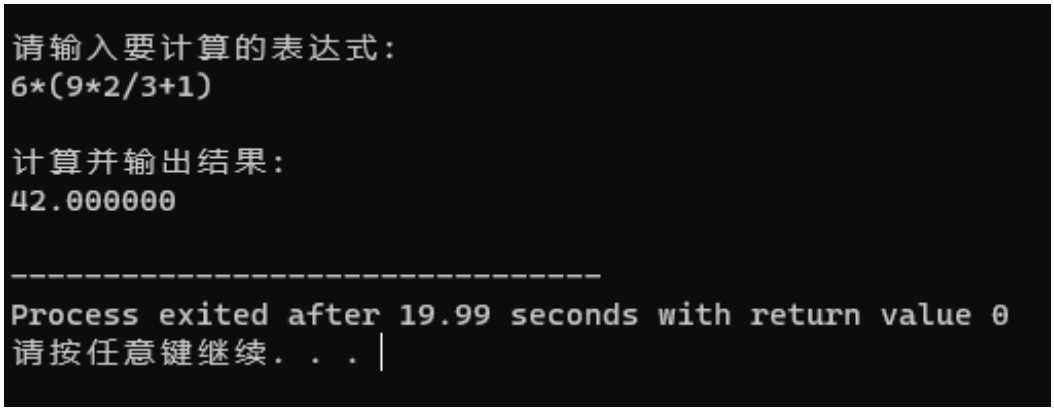


图 5 利用栈验证算术表达式正确性程序结果图

五、编写代码时遇到的问题及实验心得体会

遇到的问题：

(1) 所面对的问题直接想到运用栈和队列仍有难度，需要通过不断练习来进一步完善。
(2) 思考代码逻辑速度比较慢，需要继续花时间去提高。

实验心得体会：

(1) 通过编写算术表达式的程序实现了利用栈进行算术表达式求解。利用栈和队列分别存储运算符和字符然后比较优先级实现运算输出结果。老师的题目很难，也让我知道了自己理论能力和编程能力的不足，但是经过长时间的独立思考，觉得还是非常有成就感的。
(2) 这次编程坚决没有上网搜索现成的方案，锻炼了自己的意志力和克服困难的能力，很感谢老师的这次考验。
(3) 算术表达式求解运用了栈和队列相应知识，我从中深刻体会到了栈和队列的深刻意义，能够灵活运用栈和队列来运用到其他领域。

六、实验源码

附件一：Stack.cpp

```cpp
#include <stdio.h>
#include <stdlib.h>
#include <string.h>

#define MaxSize 9999

//用于存放运算符的栈的数据类型定义及基本操作：

//数据类型定义
#define ElemType1 char

typedef struct {
    ElemType1 data[MaxSize];
    int top;
} SqStack1;

//运算符栈的初始化操作：
void InitStack_1(SqStack1 &S) {
    memset(S.data, '\0', MaxSize);
    S.top = -1;
}

//取运算符栈的栈顶元素：
bool GetTop_1(SqStack1 &S, ElemType1 &x) {
    if (S.top == -1)
        return false;
    x = S.data[S.top];
    return true;
}

//运算符栈入栈操作：
bool Push_1(SqStack1 &S, ElemType1 x) {
    if (S.top == (MaxSize - 1))
        return false;
    S.data[++S.top] = x;
    return true;
}

//运算符栈出栈操作：
bool Pop_1(SqStack1 &S, ElemType1 &x) {
    if (S.top == -1)
```

```
42.         return false;
43.     x = S.data[S.top--];
44.     return true;
45. }
46.
47. //用于存放操作数的栈的数据类型定义及基本类型操作
48.
49. //数据类型定义
50. #define ElemType2 float
51.
52. typedef struct {
53.     ElemType2 data[MaxSize];
54.     int top;
55. } SqStack2;
56.
57. //操作数栈的初始化操作:
58. void InitStack_2(SqStack2 &S) {
59.     S.top = -1;
60. }
61.
62. //取操作数栈的栈顶元素:
63. bool GetTop_2(SqStack2 &S, ElemType2 &x) {
64.     if (S.top == -1)
65.         return false;
66.     x = S.data[S.top];
67.     return true;
68. }
69.
70. //操作数栈入栈操作:
71. bool Push_2(SqStack2 &S, ElemType2 x) {
72.     if (S.top == (MaxSize - 1))
73.         return false;
74.     S.data[++S.top] = x;
75.     return true;
76. }
77.
78. //操作数栈出栈操作:
79. bool Pop_2(SqStack2 &S, ElemType2 &x) {
80.     if (S.top == -1)
81.         return false;
82.     x = S.data[S.top--];
83.     return true;
84. }
85.
```

```
//用于判断运算符优先级的高低，优先级高于或等于就返回 true，否则为 false
bool Priority(char a, char b) {
    if ((a == '+' || a == '-') && (b == '+' || b == '-
' || b == '*' || b == '/')) {
        return true;
    } else if ((a == '*' || a == '/') && (b == '*' || b == '/')) {
        return true;
    }
    return false;
}

//用于两个操作数之间的计算
ElemType2 CalculateTwo(ElemType2 n1, ElemType2 n2, ElemType1 t) {
    //n1 是右操作数,n2 是左操作数
    if (t == '+')
            return n1 + n2;
      if (t == '-')
            return n2 - n1;
      if (t == '/')
            return n2 / n1;
      if (t == '*')
            return n2 * n1;
  }

  //计算中缀表达式的算法：
  void Calculate(ElemType1 str[], ElemType2 &e) {
      SqStack1 S1;//S1 是运算符栈
      SqStack2 S2;//S2 是操作数栈
      InitStack_1(S1);//初始化 S1
      InitStack_2(S2);//初始化 S2
      float n = 0;
      ElemType2 n1, n2;
      ElemType1 t;
      for (unsigned int i = 0; i < strlen(str); i++) {
          if (str[i] >= '0' && str[i] <= '9') {
              while (1) {
                  n *= 10;
                  n += (str[i] - '0');
                  if (str[i + 1] >= '0' && str[i + 1] <= '9')
                      i++;
                  else
                      break;
              }
              Push_2(S2, n);
```

```
129.                 n = 0;
130.             } else if (str[i] != ')') {
131.                 while (GetTop_1(S1, t) && t != '(' && Priority(str[i],
    t)) {
132.                     Pop_1(S1, t);
133.                     Pop_2(S2, n1); //n1 是右操作数
134.                     Pop_2(S2, n2); //n2 是左操作数
135.                     Push_2(S2, CalculateTwo(n1, n2, t));
136.                 }
137.                 Push_1(S1, str[i]);
138.             } else {
139.                 while (Pop_1(S1, t)) {
140.                     if (t == '(')
141.                         break;
142.                     Pop_2(S2, n1); //n1 是右操作数
143.                     Pop_2(S2, n2); //n2 是左操作数
144.                     Push_2(S2, CalculateTwo(n1, n2, t));
145.                 }
146.             }
147.         }
148.         while (Pop_1(S1, t) && Pop_2(S2, n1) && Pop_2(S2, n2)) {
149.             Push_2(S2, CalculateTwo(n1, n2, t));
150.         }
151.         Pop_2(S2, e);
152.     }
153.
154.     int main() {
155.         ElemType1 str[MaxSize];
156.         ElemType2 e;
157.         memset(str, '\0', MaxSize);
158.
159.         printf("\n 请输入要计算的表达式:\n");
160.         scanf("%s", str);
161.         printf("\n 计算并输出结果:\n");
162.         Calculate(str, e);
163.         printf("%f\n", e);
164.         return 0;
165.     }
```