****

项目说明文档

**数据结构课程设计**

**——算数表达式求解**

**培养单位：软件学院**

**本 科 生：蓝 笙 聆**

**学 号：1951096**

**指导老师：张 颖**

二○二○年十二月

目录

[第1章 功能分析 1](#_Toc58797751)

[1.1 功能分析 1](#_Toc58797752)

[1.2 算法分析 1](#_Toc58797753)

[1.2.1 中缀表达式转后缀表达式 1](#_Toc58797754)

[1.2.2 后缀表达式求值 2](#_Toc58797755)

[第2章 设计 3](#_Toc58797756)

[2.1 数据结构设计 3](#_Toc58797757)

[2.2 类结构设计 3](#_Toc58797758)

[2.3 成员与操作设计 3](#_Toc58797759)

[2.4 系统设计 4](#_Toc58797760)

[第3章 实现 5](#_Toc58797761)

[3.1 中缀表达式转后缀表达式功能的实现 5](#_Toc58797762)

[3.1.1 中缀表达式转后缀表达式功能流程图 5](#_Toc58797763)

[3.1.2 中缀表达式转后缀表达式功能核心代码 6](#_Toc58797764)

[3.1.3 中缀表达式转后缀表达式功能截屏示例 6](#_Toc58797765)

[3.2 后缀表达式求值功能的实现 7](#_Toc58797766)

[3.1.1 后缀表达式求值功能流程图 7](#_Toc58797767)

[3.1.2 后缀表达式求值功能核心代码 8](#_Toc58797768)

[3.1.3 后缀表达式求值功能截屏示例 9](#_Toc58797769)

[3.2 总体系统的实现 10](#_Toc58797770)

[3.2.1 总体系统流程图 10](#_Toc58797771)

[3.2.2 总体系统代码实现 10](#_Toc58797772)

[3.2.3 总体功能截屏示例 11](#_Toc58797773)

[第4章 测试 13](#_Toc58797774)

[4.1 通常功能测试 13](#_Toc58797775)

[4.2 其他功能测试 13](#_Toc58797776)

[4.2.1 多单目运算符测试 13](#_Toc58797777)

[4.2.2 大数运算测试 14](#_Toc58797778)

[4.2.3 小数运算测试 15](#_Toc58797779)

[4.2.3 多括号运算测试 15](#_Toc58797780)

[4.3 出错测试 16](#_Toc58797781)

[4.3.1 输入括号数不匹配 16](#_Toc58797782)

[4.3.2 输入非法表达式 16](#_Toc58797783)

[4.3.3 输入含有其他字符 16](#_Toc58797784)

# 第1章 功能分析

## 1.1 功能分析

从键盘上输入中缀算数表达式，包括括号，计算出表达式的值。

项目基本要求：

程序对所有输入的表达式作简单的判断，如表达式有错，能给出适当的提示。支持包括加减，乘除取余，乘方和括号等操作符，其中优先级是等于<括号<加减<乘除取余<乘方。

能处理单目运算符：+或-。

拓展提升：能处理10以上的数字。能处理小数。能处理多个单目运算符

## 1.2 算法分析

### 1.2.1 中缀表达式转后缀表达式

为了让中缀表达式更方便求值，我们考虑先将中缀表达式转换成后缀表达式。转换过程需要用到栈，具体过程如下：

1）如果遇到操作数，我们就直接将其输出。

2）如果遇到操作符，则我们将其放入到栈中，遇到左括号时我们也将其放入栈中。

3）如果遇到一个右括号，则将栈元素弹出，将弹出的操作符输出直到遇到左括号为止。注意，左括号只弹出并不输出。

4）如果遇到任何其他的操作符，如（“+”， “\*”，“（”）等，从栈中弹出元素直到遇到发现更低优先级的元素(或者栈为空)为止。弹出完这些元素后，才将遇到的操作符压入到栈中。有一点需要注意，只有在遇到" ) "的情况下我们才弹出" ( "，其他情况我们都不会弹出" ( "。

5）如果我们读到了输入的末尾，则将栈中所有元素依次弹出。

### 1.2.2 后缀表达式求值

将中缀表达式转换成等价的后缀表达式后，求值时，不需要再考虑运算符的优先级，只需从左到右扫描一遍后缀表达式即可。具体求值步骤为：从左到右扫描后缀表达式，遇到运算符就把表达式中该运算符前面两个操作数取出并运算，然后把结果带回后缀表达式；继续扫描直到后缀表达式最后一个表达式。

设置一个栈，开始时，栈为空，然后从左到右扫描后缀表达式，若遇操作数，则进栈；若遇运算符，则从栈中退出两个元素，先退出的放到运算符的右边，后退出的 放到运算符左边，运算后的结果再进栈，直到后缀表达式扫描完毕。此时，栈中仅有一个元素，即为运算的结果。

# 第2章 设计

## 2.1 数据结构设计

如上所述，为了完成中缀表达式转后缀表达式并求值，考虑使用栈。

## 2.2 类结构设计

注意到链表类可以直接实现栈的大部分功能，直接通过继承链表类完成栈类。

## 2.3 成员与操作设计

**模板链表节点结构体（Node）**

template <class T>

struct Node {

    T \_data;      //数据

    Node \*\_next;  //指针域

    Node();

};

**模板链表类（Node）**

template <class T>

class List {

   private:

    Node<T> \*\_head;  //表头结点

    int \_len;        //链表长度

   public:

    void ListIns(int n, T data);  //将数据插入第n个节点中

    void ListDel(int n);          //删除第n个节点

    T ListPop(T data);    //查找链表中的数据返回并删除节点

    int ListFind(T id);   //查找数据在链表中的位置

    int ListLen() const;  //返回链表长度

    bool Empty() const;   //返回是否为空

    void ListChange(int n, T data);  //修改链表第n个节点的值

    Node<T> \*GetHead() const { return this->\_head; }  //返回头结点

    std::string ListPrint(int n) const;               //将链表输出

    List();

    ~List();

};

**模板栈类（Stack）**

template <class T>

class Stack : public List<T> {

   public:

    void Push(T val);

    void Pop();

    T Top();

};

## 2.4 系统设计

系统进入循环调用loop函数，输入表达式，先转换为后缀表达式然后求值，并输出求值之后的结果。

# 第3章 实现

## 3.1 中缀表达式转后缀表达式功能的实现

### 3.1.1 中缀表达式转后缀表达式功能流程图

### 3.1.2 中缀表达式转后缀表达式功能核心代码

string toSuffix(const string &s) {

    string ret = "";                      // 用于返回的后缀表达式

    Stack<char> st;                       // 用于实现转换的栈

    for (int i = 0; i < s.size(); i++) {  // 枚举每一位

        if (s[i] == '.' ||

            isdigit(s[i])) {  // 如果为数字或小数点，直接输出到结果中

            ret += s[i];

        } else if (s[i] == '(') {  // 如果为左括号，直接加入栈中

            st.Push(s[i]);

        } else if (s[i] == ')') {  // 如果为右括号，

            while (st.Top() != '(')  // 一直弹出并输出直到遇到左括号，

                ret += ' ', ret += st.Top(), st.Pop();  // 并弹出左括号

            st.Pop();

        } else {  // 否则即为普通的运算符，

            ret += ' ';

            while (!st.Empty() && priority(st.Top()) >= priority(s[i]))

                ret += st.Top(), ret += ' ',

                    st.Pop();  // 一直弹出并输出直到遇到比当前优先级低的运算符，

            st.Push(s[i]);  // 最后把当前运算符压入栈中

        }

    }

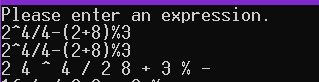
    while (!st.Empty())  // 最后如果栈中还有剩余的字符，直接弹出并输出

        ret += ' ', ret += st.Top(), st.Pop();

    return ret;  // 最后返回结果

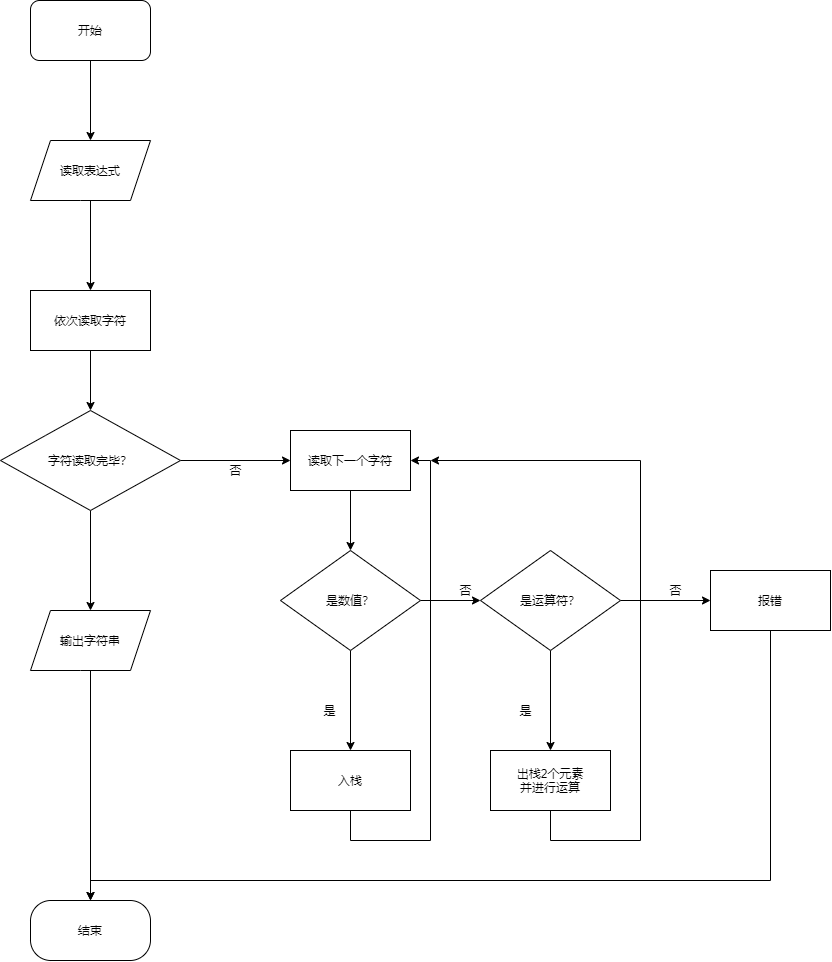
}

### 3.1.3 中缀表达式转后缀表达式功能截屏示例



## 3.2 后缀表达式求值功能的实现

### 3.1.1 后缀表达式求值功能流程图



### 3.1.2 后缀表达式求值功能核心代码

void calc(const string &s) {

    /\*\*

     \* 实现计算的数字栈，为了输出方便(不需要将每个元素弹出)

     \* 这里使用vector

     \*/

    vector<double> st;

    printSuffix(s);  // 先输出后缀表达式

    string str;

    for (int i = 0; i <= s.size(); i++) {

        if (i != s.size() && s[i] != ' ')

            str += s[i];

        else {

            double temp = -1;

            if (isdigit(str[0])) {  // 如果遇到数字直接压入栈

                temp = str2num(str);

                st.push\_back(temp);

            } else {          // 否则计算结果并压入栈

                double a, b;  //取出栈顶的两个元素

                a = st.back();

                st.pop\_back();

                b = st.back();

                st.pop\_back();

                st.push\_back(calcNum(b, a, str[0]));  // 注意计算顺序

                // 每计算一次输出一次计算过程

                for (auto &&i : st) cout << i << ' ';  // 输出栈中已计算过的数

                for (int j = i + 1; j < s.size(); j++)

                    cout << s[j];  // 再输出还未计算过的数

                cout << endl;

            }

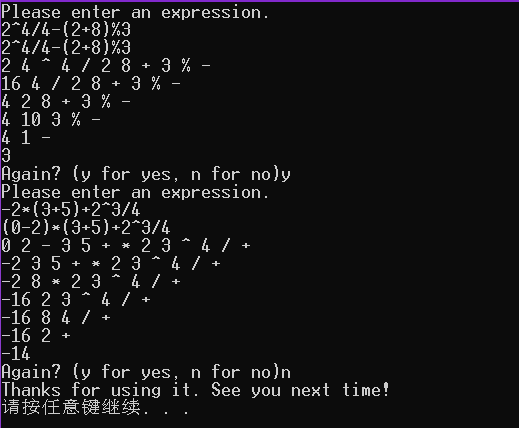
            str = "";

        }

    }

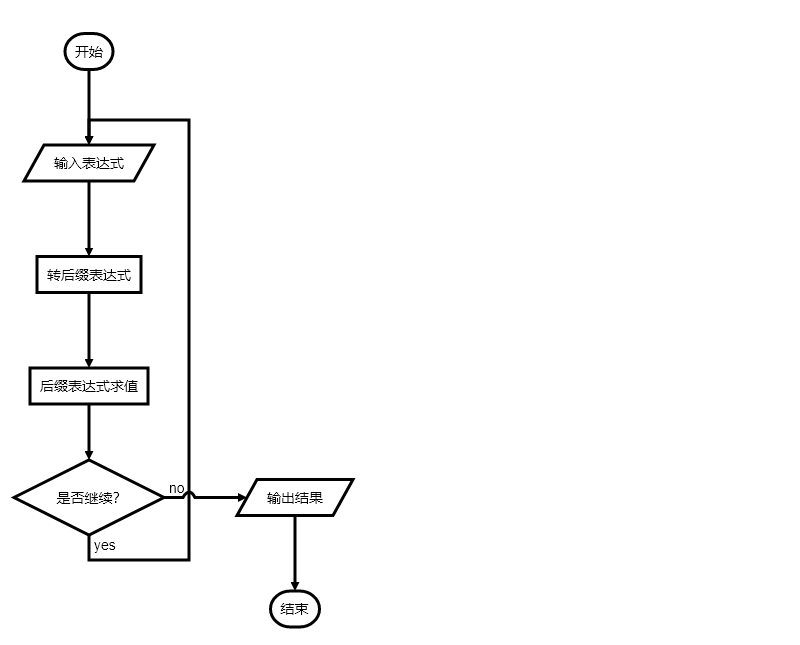
}

### 3.1.3 后缀表达式求值功能截屏示例



## 3.2 总体系统的实现

### 3.2.1 总体系统流程图



### 3.2.2 总体系统代码实现

bool loop() {

    string s;

    string ss = "";

    cout << "Please enter an expression. " << endl;

    cin >> s;

    while (!effective(s)) {

        cout << "An invalid expression was detected. Please try again." << endl;

        cin >> s;

    }

    int flag = 0;

    for (size\_t i = 0; i < s.size(); i++) {

        if ((s[i] == '+' || s[i] == '-') &&

            (i == 0 || (s[i - 1] != ')' && !isdigit(s[i - 1])))) {

            ss += '(';

            ss += '0';

            ss += s[i];

            flag++;

            continue;

        }

        ss += s[i];

        if (i != s.size() - 1 && (isdigit(s[i + 1]) || s[i + 1] == '.'))

            continue;

        for (size\_t i = 0; i < flag; i++) ss += ')';

        flag = 0;

    }

    cout << ss << endl;

    calc(toSuffix(ss));

    cout << "Again? (y for yes, n for no)";

    char c;

    cin >> c;

    if (c == 'y')

        return true;

    else

        return false;

}

int main() {

    while (loop())

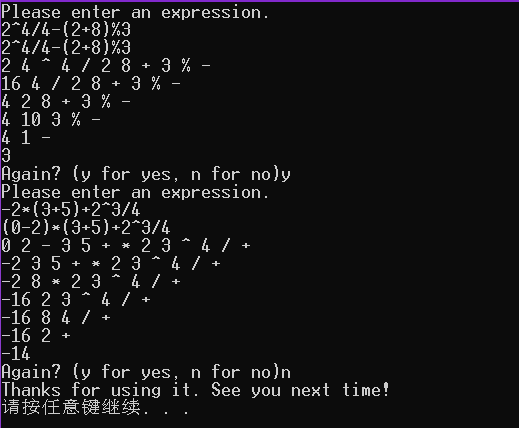
        ;

std::cout << "Thanks for using it. See you next time! " << std::endl;

    return 0;

}

### 3.2.3 总体功能截屏示例



# 第4章 测试

## 4.1 通常功能测试

**测试用例**：

2^4/4-(2+8)%3

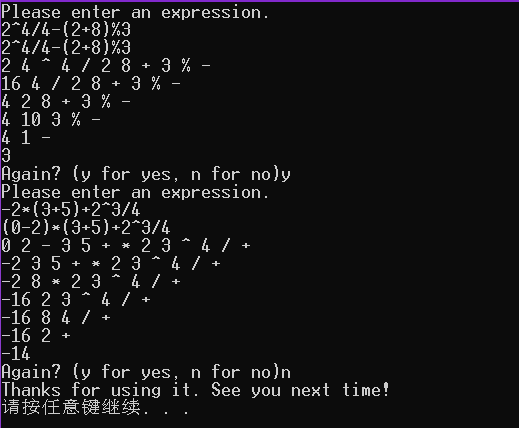
-2\*(3+5)+2^3/4

**预期结果**：

3

-14

**实验结果**



## 4.2 其他功能测试

### 4.2.1 多单目运算符测试

**测试用例：**

5------5

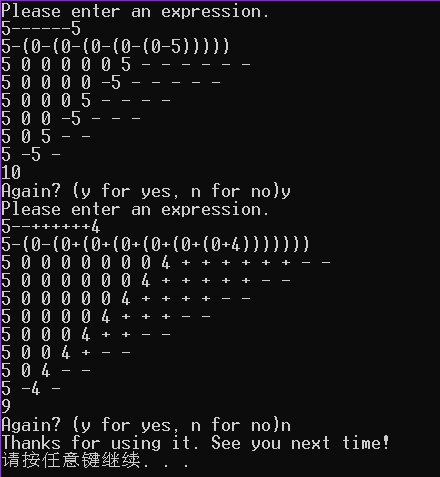
5--++++++4

**预期结果：**

10

9

**实验结果：**



### 4.2.2 大数运算测试

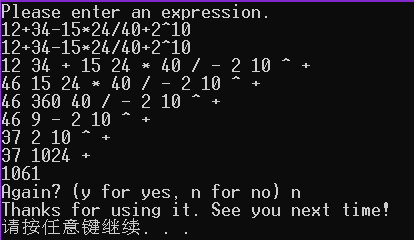
**测试用例：**

12+34-15\*24/40+2^10

**预期结果：**

1061

**实验结果：**



### 4.2.3 小数运算测试

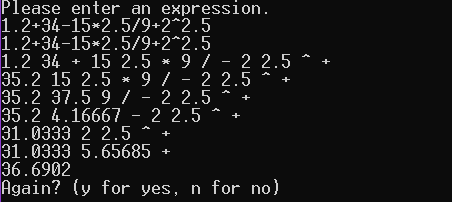
**测试用例：**

1.2+34-15\*2.5/9+2^2.5

**预期结果：**

约36.69

**实验结果：**



### 4.2.3 多括号运算测试

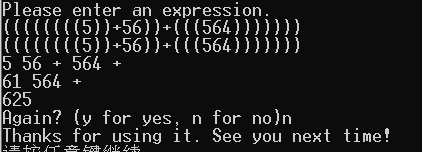
**测试用例：**

((((((((5))+56))+(((564)))))))

**预期结果：**

625

**实验结果：**



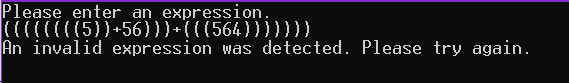
## 4.3 出错测试

### 4.3.1 输入括号数不匹配

**测试用例：**((((((((5))+56)))+(((564)))))))（多一个右括号

**预期结果：**抛出异常，程序要求重新输入，不崩溃。

**实验结果：**



### 4.3.2 输入非法表达式

**测试用例：**5\*\*\*5

**预期结果：**抛出异常，程序要求重新输入，不崩溃。

**实验结果：**



### 4.3.3 输入含有其他字符

**测试用例：**12+a-15\*24/40+2^10

**预期结果：**抛出异常，程序要求重新输入，不崩溃。

**实验结果：**

