# C++ 语 言 程 序 设 计

# 实
# 验
# 报
# 告

## 实 验 二

姓名：方尧

学号：190410102

班级：19 自动化 1 班

## 一 实验项目

1、 熟悉 C++程序设计
2、 掌握 C++基本输入输出方法
3、 掌握 C++中 string 类型的使用方法
4、 实现模板栈功能
5、 实现表达式中求值

## 二 实验原理

- 类模板

定义类模板 stack，oper 对象是操作符栈，为 int 型，初始化为 0，position_oper 为其栈顶位置，初始化为 0；number 对象为操作数栈，为 double 型，position_number 为其栈顶位置，初始化为 0；

- 优先级

order[11] = { 0,3,0,3,0,3,0,1,1,2,2 }下标为 1-10 对应 ' { } [ ] ( ) + - * / '

- 数组的预处理及作用说明：

command[length]用于存储接收进来的字符表达式，初始化全为'='，为 char 型
num[length]用于存储提取出的数值，初始化全为 0，为 double 型
brackets[length]用于存放提取出的操作符，初始化为 0，为 int 型，值与字符对应关系为：
1-'{'，2-'['，3-'('，4-'}'，5-']'，6-')'，7-'+'，8-'-'，9-'*'，10-'/'，11-'.'

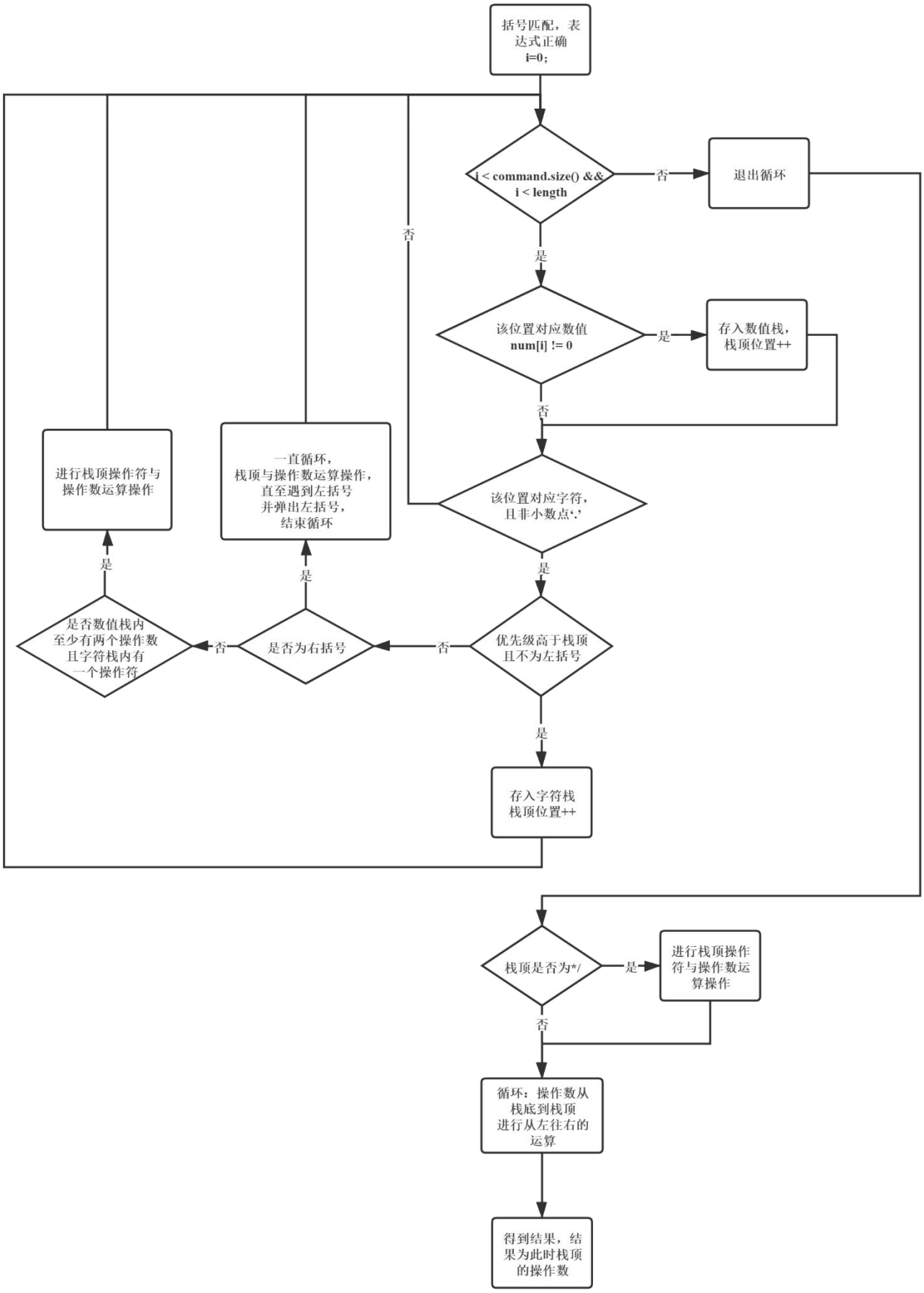
- 用到的 ASCII 码表：

figure 1 ASCII 码表

| 字符 | ASCII 码 | 字符 | ASCII 码 | 字符 | ASCII 码 | 字符 | ASCII 码 |
|---|---|---|---|---|---|---|---|
| { | 123 | } | 125 | 0 | 48 | 5 | 53 |
| [ | 91 | ] | 93 | 1 | 49 | 6 | 54 |
| ( | 40 | ) | 41 | 2 | 50 | 7 | 55 |
| + | 43 | - | 45 | 3 | 51 | 8 | 56 |
| * | 42 | / | 47 | 4 | 52 | 9 | 57 |
| . | 46 | | | | | | |

1、 说明表达式求值的方法（给出算法的流程图与简要说明）

主要利用优先级关系，配合栈类，建立数值栈和操作符栈，以及左括号优先级最高，循环计算表达式。循环完成，判断最后是否为乘除，计算。最后从左至右计算加减，得到结果。实现代码封装在 calcualte_result()函数内。


括号匹配，表达式正确
i=0;
i < command.size() && i < length
否
退出循环
是
该位置对应数值 num[i] != 0
是
存入数值栈，栈顶位置++
否
否
该位置对应字符，且非小数点‘.’
是
优先级高于栈顶且不为左括号
否
是否为右括号
否
是
一直循环，栈顶与操作数运算操作，直至遇到左括号并弹出左括号，结束循环
是
是否数值栈内至少有两个操作数且字符栈内有一个操作符
否
是
进行栈顶操作符与操作数运算操作
是
存入字符栈栈顶位置++
栈顶是否为*/
是
进行栈顶操作符与操作数运算操作
否
循环：操作数从栈底到栈顶进行从左往右的运算
得到结果，结果为此时栈顶的操作数

2、 给出程序的源代码，说明关键代码的操作含义，给出运行结果

A. 源代码见附录。

B. 操作及代码说明：
reset()函数，每次输入命令前重置 command，num，brackets
former()函数，数值整数部分操作函数
later()，数值小数部分操作函数
spl()函数，识别操作符，并将操作符信息存入 brackets 中；调用 former()和 later()函数，识别提取数值，并存入 num 中
cal()函数，判断括号匹配，若错误，输出错误信息
主要利用字符与 ASCII 码间的关系，识别字符串中的运算符以及数值。
outputformat()函数，格式化输出错误信息
calcualte_result()函数，括号匹配正确后的表达式计算函数
calcualte_simple()函数，实现二操作数加减乘除

- 定义类模板栈

```
template<class type>
class stack
{
public:
    stack(type a)
    {
        for (int i = 0; i < length; i++)
        {
            link[i] = a;
        }
    }
    type link[length];
};
```

- 建立操作符栈和操作数栈

```
//操作符栈
    stack <int> oper(0);
    int position_oper = 0;
    //操作数栈
    stack <double> number(0);
    int position_number = 0;
```

- 栈顶操作符与操作数计算

```
number.link[position_number - 1] = calcualte_simple(oper.link[position_oper], \
        number.link[position_number - 1], number.link[position_number]);
position_number--; position_oper--;
```

- 从左至右计算

```
for (int i = 1; i <= position_oper; i++)
{
      number.link[i + 1] = calcualte_simple(oper.link[i], \
            number.link[i], number.link[i + 1]);
}
```

C. 运行结果

运行结果均正确

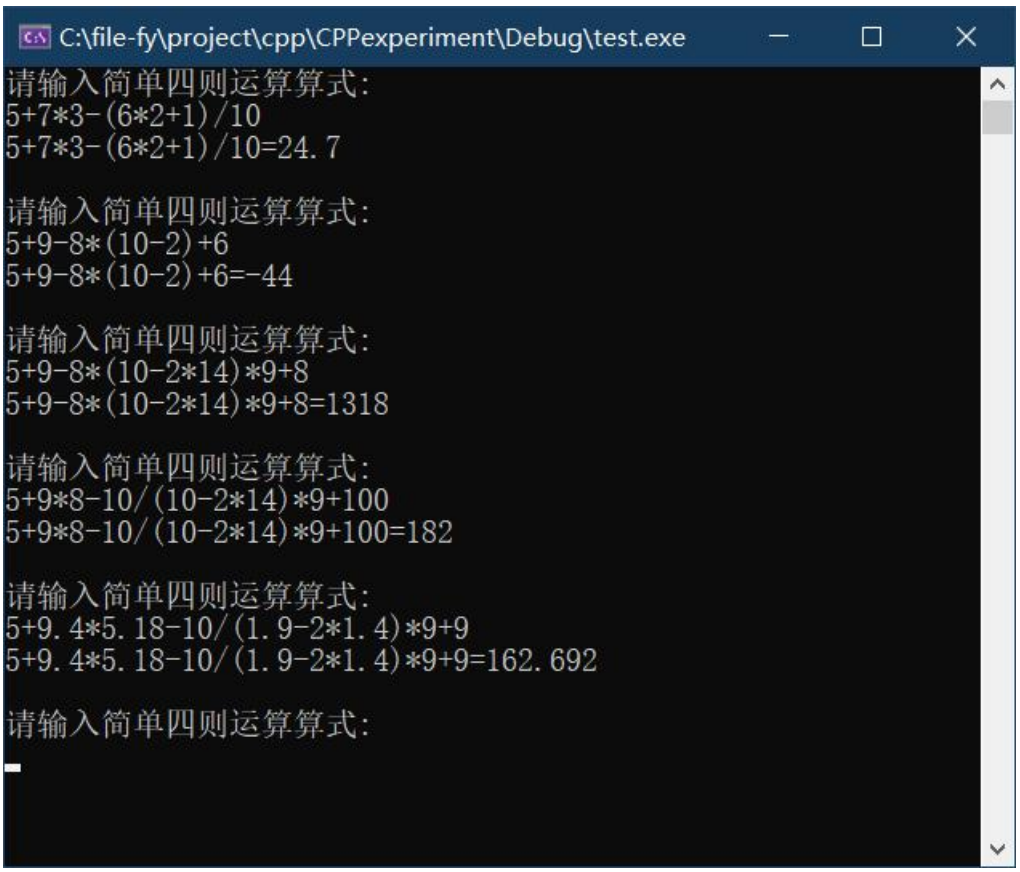



figure 1 运行结果截图

## 三 实验总结与建议

实验实施过程：先构思；画框架图；编写伪代码；用实际代码实现；运行调试；debug 直至无错误并得到预期结果。
实验解决方案：定义栈类，建立操作数（double 型）和操作符栈（int 型），实现运算符计算。

# 附录：源代码

```
#include<iostream>
#include<math.h>
#include<string>
#define length 100
using namespace std;

template<class type>
class stack
{
public:
    stack(type a)
    {
        for (int i = 0; i < length; i++)
        {
            link[i] = a;
        }
    }
    type link[length];
};

void outputformat(string command, double num[], int brackets[], int n = length);
void spl(string command, double num[], int brackets[]);
void calcualte(string command, double num[], int brackets[]);
double calcualte_result(string command, double num[], int brackets[]);
double calcualte_simple(int n, double a, double b);
void reset(string command, double num[], int brackets[]);
int former(int i, string command, double num[]);
int later(int i, int j, string command, double num[]);

int main()
{
    string command;
    double num[length] = { 0 };
    int brackets[length] = { 0 }; //1-{,2-[,3-(,4-},5-],6-)
    for (;;)
    {
        reset(command, num, brackets);
        cout << "请输入简单四则运算算式:" << endl;
        cin >> command;
        spl(command, num, brackets);
        calcualte(command, num, brackets);
        cout << endl;
    }
    return 0;
}
void reset(string command, double num[], int brackets[])
{
    int i, j;
    for (int i = 0; i < length; i++)
    {
        command = "";
        num[i] = 0;//num，brackets 默认初值为 0,'0'
        brackets[i] = 0;
    }
}
void spl(string command, double num[], int brackets[])
{
    for (int i = 0; i < command.size() && i < length; i++)
    {
        switch (command[i])
        {
        case '{':
            brackets[i] = 1;
            break;
        case '}':
```

```
            brackets[i] = 2;
            break;
        case '[':
            brackets[i] = 3;
            break;
        case ']':
            brackets[i] = 4;
            break;
        case '(':
            brackets[i] = 5;
            break;
        case ')':
            brackets[i] = 6;
            break;
        case '+':
            brackets[i] = 7;
            break;
        case '-':
            brackets[i] = 8;
            break;
        case '*':
            brackets[i] = 9;
            break;
        case '/':
            brackets[i] = 10;
            break;
        case '.':
            brackets[i] = 11;
            break;
        }
    }
    for (int i = 0; i < command.size() && i < length; i++)
    {
        int j, k;
        if (int(command[i]) >= 48 && int(command[i]) <= 57)
        {
            num[i] = command[i] - '0';
            j = former(i, command, num);
            k = later(i, j, command, num);
            if (k != 0)
                i = k;
            else
            {
                i = j;
                cout << "wrong\n";
                /*错误输出*/
            }
        }
    }
}
//小数点前数字处理
int former(int i, string command, double num[])
{
    int j;
    for (j = i + 1; j < command.size() && j < length; j++)
    {
        if (int(command[j]) >= 48 && int(command[j]) <= 57)
        {
            num[i] = num[i] * 10 + command[j] - '0';
        }
        else
            break;
    }
    return j;
}

//小数点后的数字处理
int later(int i, int j, string command, double num[])
{
```

```
    int k = 0;
    if (int(command[j]) == 46)
    {
        if (!(command[j + 1] >= 48 && command[j + 1] <= 57))
            return 0;
        else
        {
            for (int k = j + 1; k < command.size() && k < length; k++)
            {
                if (int(command[k]) >= 48 && int(command[k]) <= 57)
                {
                    //cout<<num[i]<<"\t"<<command[k]-'0'<<"\t"<<double(pow(10,k-j))<<endl;

                    num[i] = num[i] + (command[k] - '0') / double(pow(10, k - j));
                    //最后为数字结尾
                    if (k == command.size() - 1) return command.size() - 1;
                }
                else
                    return k;
            }
        }
    }
    else
        return j;
    return j;
}

void calcualte(string command, double num[], int brackets[])
{
    int wrong = 0;
    //left 从第二个存储位置开始存
    stack <int> left('0');
    int position = -1;
    //判断括号是否匹配
    for (int i = 0; i < command.size() && i < length; i++)
    {
        if (brackets[i] == 1 || brackets[i] == 3 || brackets[i] == 5)
        {
            left.link[position + 1] = brackets[i];
            position++;
        }
        if (brackets[i] == 2 || brackets[i] == 4 || brackets[i] == 6)
        {
            if (position == -1) { wrong = i; break; }
            if (brackets[i] == 2)
            {
                if (left.link[position] != 1)
                {
                    wrong = i;
                    break;
                }
                else
                {
                    left.link[position] = 0;
                    position--;
                }
            }
            if (brackets[i] == 4)
            {
                if (left.link[position] != 3)
                {
                    wrong = i;
                    break;
                }
                else
                {
                    left.link[position] = 0;
                    position--;
                }
```

```
            }
            if (brackets[i] == 6)
            {
                if (left.link[position] != 5)
                {
                    wrong = i;
                    break;
                }
                else
                {
                    left.link[position] = 0;
                    position--;
                }
            }
        }
    }
    if (wrong != 0)
    {
        outputformat(command, num, brackets, wrong);
        cout << "匹配错误" << endl;
    }
    else
    {
        if (position == -1)//正确，输出计算结果
        {
            double result = calcualte_result(command, num, brackets);
            if (result == 999.99)
                cout << "wrong" << endl;
            else
            {
                cout << command;
                cout << "=" << result << endl;
            }
        }
        else
        {
            outputformat(command, num, brackets);
            cout << "无法匹配" << endl;
        }
    }
}
double calcualte_result(string command, double num[], int brackets[])
{
    int order[11] = { 0,3,0,3,0,3,0,1,1,2,2 };//加减乘除 7,8,9,10
    double result = 999.99;

    //操作符栈
    stack <int> oper(0);
    int position_oper = 0;
    //操作数栈
    stack <double> number(0);
    int position_number = 0;
    for (int i = 0; i < command.size() && i < length; i++)
    {
        if (num[i] != 0)//数值数组非 0 存入
        {
            number.link[position_number + 1] = num[i];
            position_number++;
        }
        if (brackets[i] != 0 && brackets[i] != 11)//操作符数组非 0 且非.存入
        {
            if (order[oper.link[position_oper]] < order[brackets[i]] || order[oper.link[p
osition_oper]] == 3)//优先级高于现栈顶优先级，压入
            {
                oper.link[position_oper + 1] = brackets[i];
                position_oper++;
            }
            else
            {
```

```
                if (brackets[i] == 2 || brackets[i] == 4 || brackets[i] == 6)
                {
                    for (;;)
                    {
                        if (oper.link[position_oper] == brackets[i] - 1)
                        {
                            position_oper--;
                            break;
                        }
                        else
                        {
                            number.link[position_number - 1] = calcualte_simple(oper.link[position_oper], \
                                number.link[position_number - 1], number.link[position_number]);
                            position_number--; position_oper--;
                        }
                    }
                }
                else
                {
                    if (position_number > 1)
                    {
                        if (position_oper > 0)
                        {
                            number.link[position_number - 1] = calcualte_simple(oper.link[position_oper], \
                                number.link[position_number - 1], number.link[position_number]);
                            position_number--; position_oper--;
                        }
                    }
                    oper.link[position_oper + 1] = brackets[i];
                    position_oper++;
                }
            }
        }
    }
    if (oper.link[position_oper] == 9 || oper.link[position_oper] == 10)
    {
        number.link[position_number - 1] = calcualte_simple(oper.link[position_oper], \
            number.link[position_number - 1], number.link[position_number]);
        position_number--; position_oper--;
    }
    for (int i = 1; i <= position_oper; i++)
    {
        number.link[i + 1] = calcualte_simple(oper.link[i], \
            number.link[i], number.link[i + 1]);
    }
    result = number.link[position_number];
    return result;
}
double calcualte_simple(int n, double a, double b)
{
    if (n == 7)return a + b;
    if (n == 8)return a - b;
    if (n == 9)return a * b;
    if (n == 10 && b != 0)return a / b;
    return 0;
}
void outputformat(string command, double num[], int brackets[], int n)
{
    for (int i = 0; i < command.size() && i < length && i <= n; i++)
    {
        if (num[i] != 0)
        {
            cout << "操作数:\t" << num[i] << endl;
        }
        if (brackets[i] != 0 && brackets[i] != 11)
```

```
363.         {
364.             switch (brackets[i])
365.             {
366.             case 1:
367.                 cout << "操作符:\t 左大括号\t";
368.                 break;
369.             case 2:
370.                 cout << "操作符:\t 右大括号\t";
371.                 break;
372.             case 3:
373.                 cout << "操作符:\t 左中括号\t";
374.                 break;
375.             case 4:
376.                 cout << "操作符:\t 右中括号\t";
377.                 break;
378.             case 5:
379.                 cout << "操作符:\t 左小括号\t";
380.                 break;
381.             case 6:
382.                 cout << "操作符:\t 右小括号\t";
383.                 break;
384.             case 7:
385.                 cout << "操作符:\t 加号\t";
386.                 break;
387.             case 8:
388.                 cout << "操作符:\t 减号\t";
389.                 break;
390.             case 9:
391.                 cout << "操作符:\t 乘号\t";
392.                 break;
393.             case 10:
394.                 cout << "操作符:\t 除号\t";
395.                 break;
396.             }
397.             if (i != n)cout << endl;
398.         }
399.     }
400. }
```