武汉纺织大学计算机与人工智能学院

专业综合训练 I 课程报告

训练选题：　表达式及其应用

专业班级：　计算机类 12101 班

学　　号：　2107250127

学生姓名：　刘家麒

指导教师：　专业综合训练 I 课程组

2022 年　7　月　4　日

目录

1、项目介绍

1.1 训练内容

本专题为表达式求值及其应用，主要有三个任务，分别为——交互式求表达式值：支持 C 语言中大多数一元或二元运算符，符合 C 语言运算符的语法与语义，支持变量和常量作操作数，存储会话期变量的取值，输入表达式即求值；真值表及其应用：编写真值表程序，根据实际逻辑问题设计命题公式，通过输入命题公式到真值表程序的输出回答逻辑问题；二元树与表达式：通过中缀表达式建立对应二元树，对二元树操作求表达式的值，输出波兰和逆波兰表达式，仅带必要括号的中缀表达式。

1.2 训练要求

本次训练要求全体组员在培训过程认真听取老师所讲的内容并理解老师分发的关于简单的中缀表达式的求值的示例代码，理解计算机计算表达式的几种算法，并熟悉数据结构中栈的基本操作和实际应用。另外，还需要了解表达式可以以一颗二叉树无歧义表示，通过对树的不同顺序遍历可以获得前中和后缀表达式，通过对树的递归操作可以求得表达式的值其次，还要理解离散数学课程中学习的真值表技术、判断命题的真假、通过构造真值表确定命题公式的真值、了解连结词的优先级、结合律。在理解上述理论知识后，还需要通过查阅资料、询问他人等方式学习 C++部分 STL 的基本使用方式以及 C++语言的一些基本语法，掌握一些数据结构的代码实现，加之要具有一定规模程序的编写和调试解决问题的能力、与组内其他成员进行沟通协商的合作能力。在该报告的编写阶段，还要能撰写软件开发过程中的关键环节，阐述并分析方案的合理性与选择性，以较为规范的方式写入到报告中。

2、项目分析

2.1 功能列表

实现了专题一选题的拓展任务一：较复杂表达式的交互式求值。输入指定的命令和表达式可以进行对应的操作。

此模块是一个交互式的求值计算器，支持 C 语言以下 + - * / % == != < <= > >= 的双目运算符，以及“！”这一单目运算符和表示优先级的（）运算符。所有运算符均和 C 语言中的对应的意义相同，优先级也与 C 语言一致，其次还支持了标识符（变量）。

其中各个分模块如下：

计算表达式模块，可以输入 calc (表达式)计算表达式的值并显示出来，表达式可以含有所有上述的运算符，也可以是含有嵌套的括号的较为复杂的表达式，对于操作数，可以是小数，也可以是负数，但是复数必须要用“（）”来表示其为一个负数。在表达式中还可以具有已经定义过的变量，程序运行时会将变量当前的值取出参与计算。

变量的定义、赋值、释放、显示模块，此模块参考 C++语言的内存动态管理的语法模

仿编写。可以输入 new（变量名）= （表达式）来新创建一个新的变量，并已输入的表达式的计算结果为初始值。变量名必须为纯字母、且不能重复定义的变量。对于已经被定义的变量，输入 let（变量名）（赋值符号）（表达式）对此变量赋一个新的值，其中赋值符号可以是普通赋值即“=”，也可以是“+= -= *= /= ”这类的复合赋值符号。其运算方式和 C 语言一致。此外，输入 show（变量名）显示已经定义的变量的值，输入 showvar 显示当前所有变量，输入 delete（变量名）即可删除一个曾经定义的变量，删除后不可对其进行任何操作。对于创建成功但未被删除的变量，可以在上述计算表达式模块进行使用，计算时会取出变量当时所保存的值作为标识符的值参与计算。

此外，在命令行太多内容的情况下，输入 cls 命令会清空此前的输出。想要退出程序时，只需要输入 EXIT 即可。

2.2 实施方案

对于最基本的表达式的求值模块，通过浏览 B 站、CSDN、GitHub 等网站的相关内容以及复习了数据结构、离散数学的教材，我大概明白了要实现中缀表达式的求值，主要以下几种方式：第一种便是通过一个运算符栈将中缀表达式转为后缀表达式，再利用一个操作数栈实现后缀表达式的求值；第二种就是一次同时利用运算符栈和操作数栈来实现中缀表达式的直接求值；第三种便是该专题的第三个拓展任务中的方法，即先从中缀表达式建立该表达式的二元树，再通过对树的递归操作求得表达式的值。

对于变量模块，在初始阶段通过资料的查阅，初步确定了 std::map、散列表、链表、二叉树等方案，上述的方案都是可以选择的，都可以实现变量模块增删变量、查找到指定变量进行取值和赋值的基本操作。

在最后方案的选择上，为降低对用户输入的中缀形式的格式的要求，即是能够允许冗余的括号以及冗余的空格作为分隔提升可读性，故最后没有选择直接对中缀表达式进行操作，而是选用了先把输入的表达式的变量替换为其实际的值得到只含操作符和运算数中缀表达式，随后再转为标准格式的后缀表达式（即为上述的第二种方式）。其标准格式为 每个操作数或是运算符后都以一个空格作为分隔，并且如同含有多个字符的运算符，在转为后缀表达式的过程中会替换为单个字符便于对后缀表达式的处理。其中“!=”变为“\”；“>=”变为“]”；“<=”变为“[”;而“|| && ==”这样的运算符则直接去掉第二个字符即可。选用此种表达式的求值方法还能提高部分代码的复用性，对于中缀表达式转标准后缀表达式的代码就可以给拓展任务三通过中缀表达式建立二叉表达式树的代码编写过程中就可以复用中缀表达式转后缀表达式的代码。而对于变量模块，链表显然是不太合适的，虽然双向循环链表插入新的元素、删除指定元素的速度都较快，但查找指定元素的速度显然是不够快的。对于散列表，由于变量的个数事先不能确定，故实现能增长的散列表较为复杂，冲突问题发生概率较大，故该方案也被放弃。树形结构显然是合适的，具有较快的查找速度，也便于在不确定数据规模的情况下增删数据，为了简化代码以及加强对 STL 的使用能力，故最后选择用 std::map 的方案，使用 std::map<string,LDouble>来存放变量，以变量名为键值，以变量值为实值。

3、项目设计与实现

3.1 数据模型（或数据结构设计）

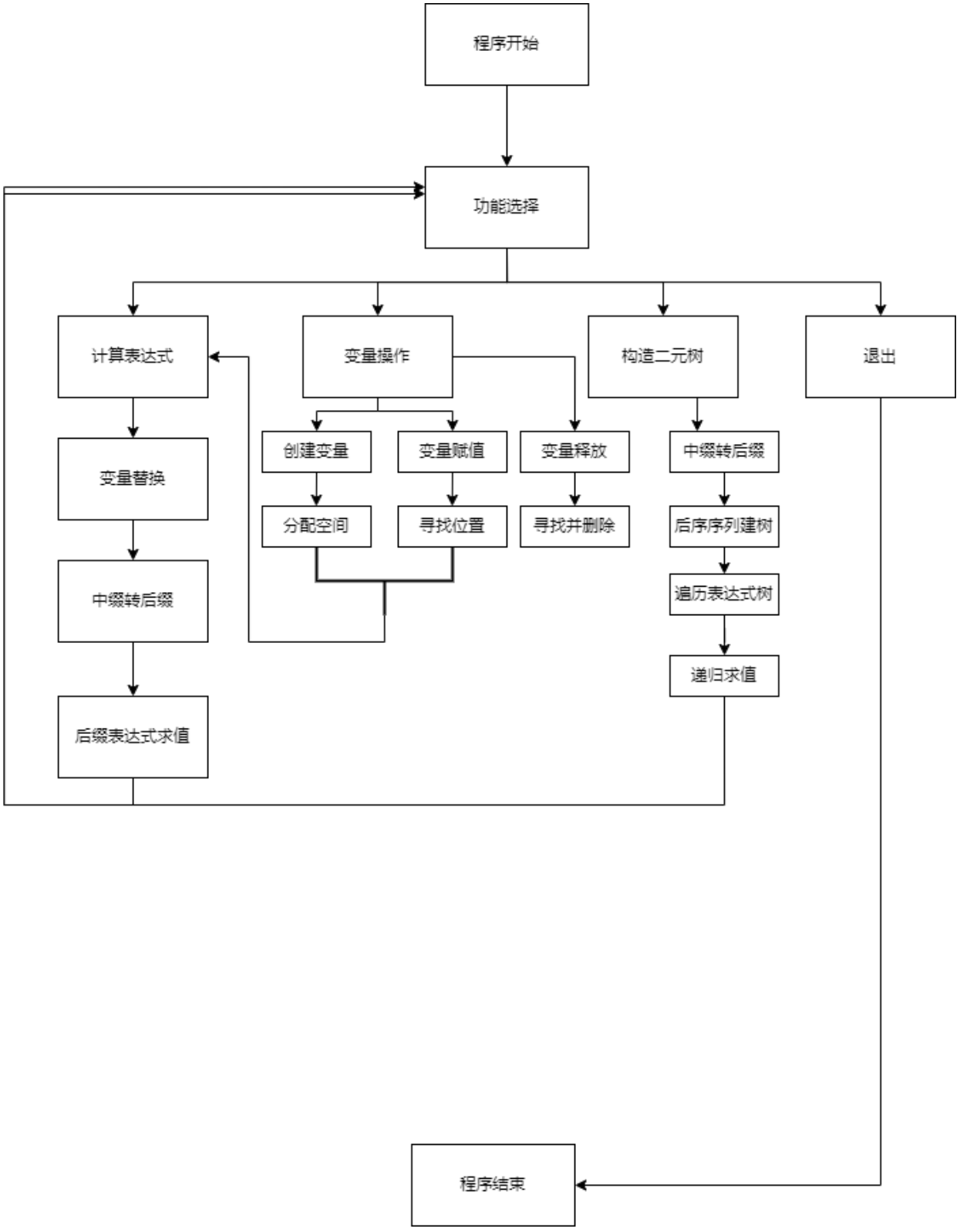
主要涉及对变量类的数据结构的设计和实现。由于变量名（标识符）与变量的值具有键值对的一一对应的关系，而且变量名一定是唯一的，需要随机访问且频繁插入删除，故应该使用 C++ STL 中的 map 进行实现，其底层是红黑树，具有较快的插入删除以及查找速度，基本符合变量类的使用场景。该类的定义如图（3.3-1），其封装着一个键为 string 类型的变量名，值为 LDouble 类型的实际值的 map。内部有许多便于中缀表达式中变量名到变量值的替换、变量新增、变量赋值和变量释放的方法。

在该类内，有新建变量的接口，会调用 map 的 insert 方法插入新的变量，若变量已经存在，则会直接退出该方法，即可保证变量名的唯一性，用于释放变量的接口同理也会在确认该变量已经存在后调用 map 的 erase 方法进行删除。

```
class Variable //变量类
{
protected:
    map<string, LDouble> varMap;  //key为变量名 value为数值的map
public:
    void newVar(string name,LDouble firstVaule);  //新建变量
    void delVar(string name); //删除变量
    LDouble& at(string name); //获取变量的引用
    int size(void); //获取变量个数
    bool VarExist(string name);//是否存在该变量
    void showAll(void); //查看所有变量
    void show(string name);  //查看单个变量
    string getValueStr(string name); //得到value的数字字符串
};
```

（图 3.1-1）

3.2 模块设计


程序开始
功能选择
计算表达式
变量操作
构造二元树
退出
变量替换
中缀转后缀
后缀表达式求值
创建变量
变量赋值
变量释放
分配空间
寻找位置
寻找并删除
中缀转后缀
后序序列建树
遍历表达式树
递归求值
程序结束

3.3 算法设计

关键算法主要包括：表达式中变量名替换为变量值的算法、中缀表达式的简要验证算法、中缀转后缀表达式算法的设计、后缀表达式计算表达式结果算法。

表达式中变量名替换为变量值算法：遍历用户输入的带有变量的准中缀表达式字符串 preInfixExp，并创建一个中缀表达式字符串 infixExp，遍历时，若当前的字符不是字母则直接输出到新字符串中。反之若是字母，则继续遍历直到遇到非字母时结束。此时检查遍历得到的单词字符串是否是当前已经保存的变量，若不是则返回失败，若是则将变量的值输出到新字符串中，考虑到负数和负号的存在，变量的值输出时外层会输出一个括号。当变量结束时为字符串加上休止符号‘\0’，并返回成功。

中缀表达式的简要验证算法：首先是括号的匹配问题，遍历表达式字符串，遇到“（”压栈，遇到左括号则弹出一个“）”，当遍历结束时栈空则匹配成功，反之则匹配失败。其次检查“.”是否不在字符串头尾且两边均为数字。当且仅当两次检查都成功时验证才能通过验证。

中缀表达式转后缀表达式算法：遍历经过表达式替换后的中缀表达式串 infixEXp，并创建后缀表达式 suffixExp。遍历时，若该字符是空格则跳过；若是操作数则继续向后遍历得到整个数字随后直接输出到后缀表达式，取得数字后更新遍历指针到数字的最后一位，特别地，若识别到负号则也直接输出到表达式中；若是运算符，则当栈顶运算符的优先级小于当前运算符时可以继续压栈，否则则需要先弹栈输出直至栈顶运算符优先级小于当前运算符时再将当前运算符压栈。若遇到右括号则无条件压栈，遇到左括号则无条件弹栈输出并输出运算符直至右括号出栈（不输出到后缀表达式），且若栈顶为右括号，则可以无条件压栈。遍历结束时将栈内所有运算符全部输出，并为字符串加上休止符‘\0’。另外，为了保证后缀表达式的标准性，便于后续处理，会在输出一个数字或是一个运算符时输出一个空格表示分隔。

后缀表达式计算表达式结果算法：先初始化一个操作数栈，随后遍历后缀表达式 suffixExp，若是该字符是数字，则继续向后遍历获得整个操作数并压入栈中，若是单目元算符，则对取出栈顶元素，进行运算后重新压栈；若是双目运算符，则连续取出栈顶的两个运算数，运算完成后将结果压栈。遍历结束后，在表达式格式正确的前提下，栈的操作数个数唯一，且栈顶操作数为表达式的计算结果。

3.4 函数实现

中缀达式转后缀表达式函数(图 3.4-1)：

void infixToSuffixExp(char* infixExp,char*& suffixExp)

参数依次为中缀表达式；所生成的后缀表达式

```
void infixToSuffixExp(char* infixExp, char*& suffixExp)
{
    int len = strlen(infixExp);
    suffixExp = new char[100 * len+200];
    int writeLoc = 0;
    stack<char> op;
    for (int i = 0; i < len; ++i)
    {
        if (infixExp[i] == ' ') continue; //过滤输入的表达式中的无用空格
        if (isNegative(infixExp, i))
        {
            suffixExp[writeLoc++]='-'; //把负号输出
            continue;
        }
        if (isNumber(infixExp[i])) //是数字
        {
            int j = i;
            while (isNumber(infixExp[j])||infixExp[j]=='.')  //把从该位置开始后面的数字全部输出
            {
                suffixExp[writeLoc++] = infixExp[j];
                ++j;
            }
            i = j - 1;
            suffixExp[writeLoc++] = ' '; //输出一个空格
        }

        else  //是运算符
        {
            char newOp = readOp(infixExp[i], infixExp, i); //识别新的运算符
            if (canPush(newOp,op)) //可以继续压栈
            {
                op.push(newOp);
            }
            else  //要出栈
            {
                if (infixExp[i] == ')')  //遇到左扩回就一直出栈直到遇到右括号
                {
                    char top = op.top();
                    op.pop();
                    while (top != '(')
                    {
                        suffixExp[writeLoc++] = top; //把 ( ) 间的运算符抛出
                        suffixExp[writeLoc++] = ' '; //输出一个空格
                        top = op.top();
                        op.pop();
                    }
                }
                else //栈顶运算符优先级大于等于新运算符
                {
                    for (; !canPush(newOp, op); op.pop()) //抛出栈内运算符直到新的可以入栈
                    {
                        suffixExp[writeLoc++] = op.top();
                        suffixExp[writeLoc++] = ' '; //输出一个空格
                    }
                    op.push(infixExp[i]);
                }
            }
        }

    }
    for (; !op.empty(); op.pop())  //栈里还有运算符就抛出
    {
        suffixExp[writeLoc++] = op.top();
        suffixExp[writeLoc++] = ' ';
    }
    suffixExp[writeLoc] = '\0'; //结尾加上结束符
}
```

（图 3.4-1）

变量替换为值函数(图 3.4-2)：

bool varSubstitute(Variable& var,char* preInfixexp,char*& infixExp)　返回值为代表是否把所有的字母都替换为了变量的值，参数依次为程序目前所有变量的引用；用户输入的带有变量的表达式字符串；完成替换后的中缀表达式字符串

```
bool varSubstitute(Variable& var, char* preInfixExp,char*& infixExp) //变量替换
{
    int len = strlen(preInfixExp);
    infixExp = new char[3 * len+300];
    int writeLoc = 0;
    for (int i = 0; i < len; ++i)
    {
        if (!isAlphabet(preInfixExp[i])) //不是字母无需处理
        {
            infixExp[writeLoc++] = preInfixExp[i];
        }
        else //变量替换为数字
        {
            int j = i;
            string name;
            string valueStr;
            int negativeLogo = 0; //代表表示负数的"次数"
            while (isAlphabet(preInfixExp[j]))
            {
                if (j >= 2 && preInfixExp[j - 1] == '-' && preInfixExp[j - 2] == '(')
                {
                    ++negativeLogo; //引用变量时取反
                }
                name.push_back(preInfixExp[j]);
                ++j;
            }
            i = j - 1; //更新遍历指针到最后一个字母
            if (!var.VarExist(name))
            {
                return false;
            }
            infixExp[writeLoc++] = '(';
            valueStr = var.getValueStr(name); //得到数字的字符串
            for(auto ch: valueStr)
            {
                if (ch == '-')
                {
                    ++negativeLogo;
                    if (negativeLogo == 2) continue;  //负负得正 一个负号的情况把负号输出
                }
                infixExp[writeLoc++] = ch; //数字替换
            }
            infixExp[writeLoc++] = ')';
        }
    }
    infixExp[writeLoc++] = '\0';
    return true;
}
```

（图 3.4-2）

优先级的比较函数(图 3.4-3)：
int priority(char op) 参数为当前运算符，返回值为该运算符的优先级

```
int priority(char op)  //运算符优先级
{
    if (op == '|') return 1;
    if (op == '&') return 2;
    if (op == '=' || op == '\\') return 3;   // \为不等于
    if (op=='<'||op=='>'||op==']'||op=='[') return 4;
    if (op == '+' || op == '-')  return 5;
    if (op == '*' || op == '/' || op == '%') return 6;
    if (op == '!') return 7;
}
```

(图 3.4-3)

3.5 功能测试

不带变量的表达式的计算： 带有括号和小数的四则运算（图 3.5-1）、带负数的运算（图 3.5-2，图 3.5-3）、含有关系、逻辑运算符的运算（图 3.5-4，图 3.5-5）。

```
Input command >>>calc (2+3)*5+60.3
自动生成的后缀表达式为 : 2 3 + 5 * 60.3 +
结果为: 85.30
```

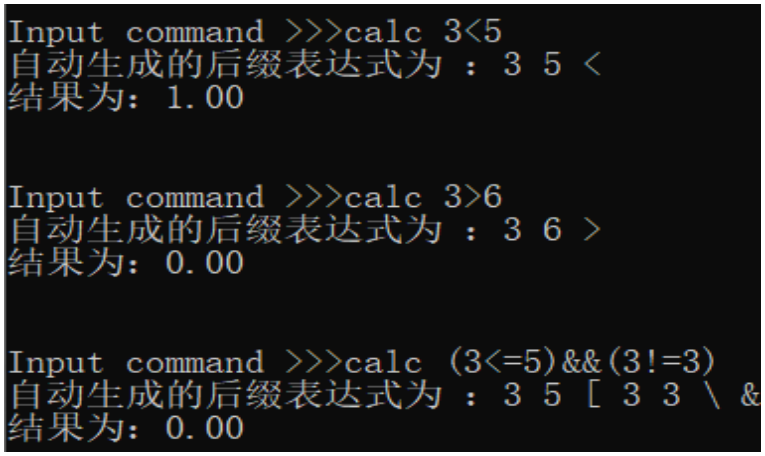
（图 3.5-1）

```
Input command >>>calc (2+3)%3+(-2)*6+(66/33.3)
自动生成的后缀表达式为 : 2 3 + 3 % -2 6 * + 66 33.3 / +
结果为: -8.02
```

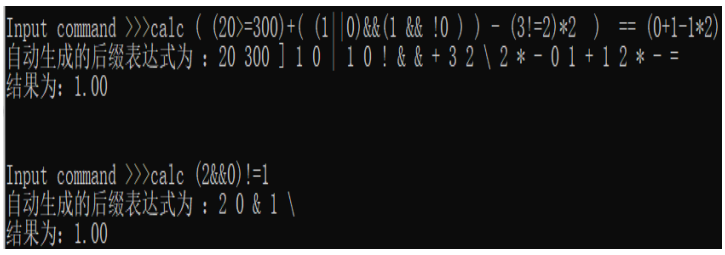
（图 3.5-2）

```
Input command >>>calc (-2)*(-3)
自动生成的后缀表达式为 : -2 -3 *
结果为: 6.00


Input command >>>calc 65/(-20.2)+50.62
自动生成的后缀表达式为 : 65 -20.2 / 50.62 +
结果为: 47.40
```

（图 3.5-3）

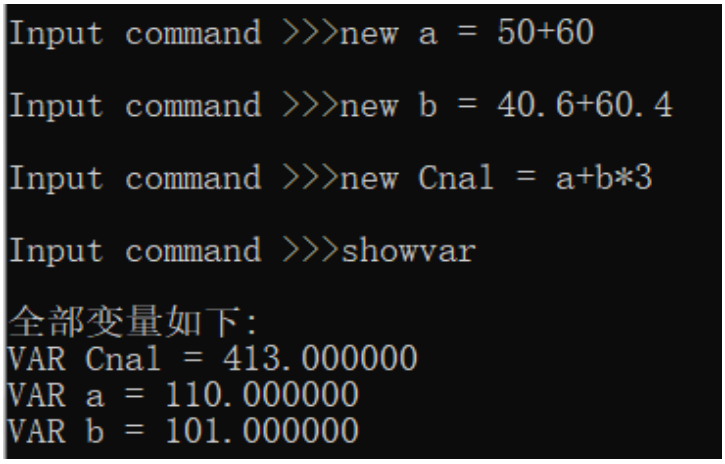


（图 3.5-4）

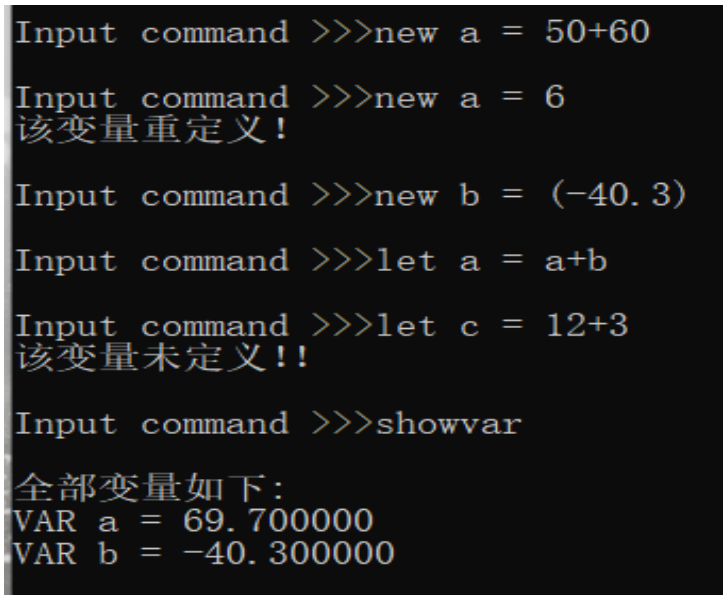


（图 3.5-5）

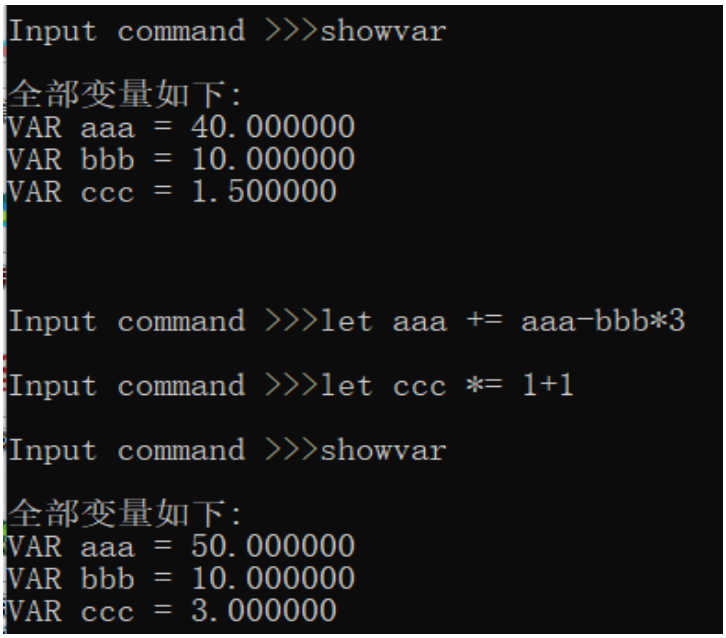
变量的定义、赋值、显示和释放：定义并显示变量（图 3.5-6，图 3.5-7）、变量赋值（图 3.5-7）、复合赋值（图 3.5-8，图 3.5-9）变量的释放（图 3.5-10）



（图 3.5-6）


Input command >>>new a = 50+60
Input command >>>new a = 6
该变量重定义!
Input command >>>new b = (-40.3)
Input command >>>let a = a+b
Input command >>>let c = 12+3
该变量未定义!!
Input command >>>showvar
全部变量如下:
VAR a = 69.700000
VAR b = -40.300000

（图 3.5-7）


Input command >>>showvar
全部变量如下:
VAR aaa = 40.000000
VAR bbb = 10.000000
VAR ccc = 1.500000
Input command >>>let aaa += aaa-bbb*3
Input command >>>let ccc *= 1+1
Input command >>>showvar
全部变量如下:
VAR aaa = 50.000000
VAR bbb = 10.000000
VAR ccc = 3.000000

（图 3.5-8）

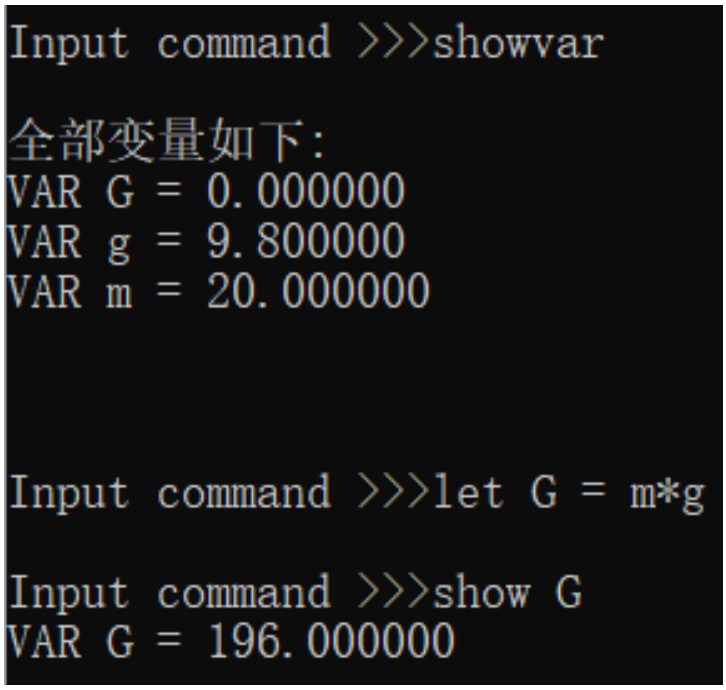
```
Input command >>>new a = 0

Input command >>>let a = (12+4)/4

Input command >>>show a
VAR a = 4.000000

Input command >>>let a -= 8

Input command >>>show a
VAR a = -4.000000

Input command >>>let a *= (-a)

Input command >>>show a
VAR a = 16.000000

Input command >>>let a /= 2.3

Input command >>>show a
VAR a = 6.956522
```

（图 3.5-9）

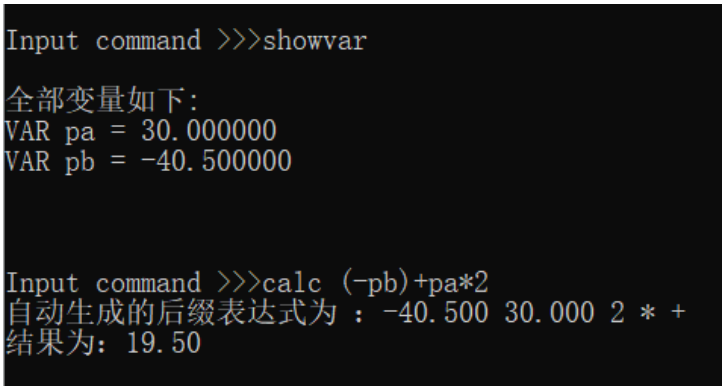
```
Input command >>>showvar

全部变量如下:
VAR PI = 3.140000
VAR e = 2.710000
VAR g = 9.800000



Input command >>>delete PI

Input command >>>let PI = e+g
该变量未定义!
```

（图 3.5-10）

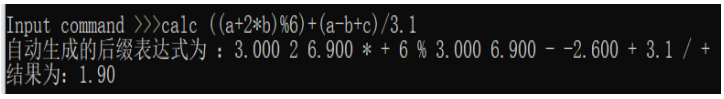
带有变量的表达式的计算：（图 3.5-11,图 3.5-12，图 3.5-13）



（图 3.5-11）



（图 3.5-12）



（图 3.5-13）

4、总结

4.1 团队合作

在本次实训中，我组的成员有刘家麒、杨成希与陈宇凡三人。三人均有热情的、积极的参与到到此次的实训中来，在实训过程中，我们有进行合理明确的分工，在分工后经常通过 QQ 聊天和语音通话交流自己的任务进度和所在过程中遇到的困难，交流沟通时三人均能较好地做到相互协作、相互信赖、相互分享相关的知识和技术、虚心听取其他组员的意见并做出合理反应。

具体的组员分工为：刘家麒负责拓展任务一（较复杂表达式的交互式求值）；杨成希负责拓展任务二（求真值表及应用）；陈宇凡负责拓展任务三（二元树与表达式）。

由于拓展任务一和三的联系较为紧密，而二与前两者虽有共通点，但差异性更大，故我组将拓展任务一和三合并到一个项目中 ，通过刘家麒和陈宇凡定期同步代码文件的方式完成代码的编写，使用这样的方式能提升组员的沟通能力、协同开发能力，还能提高部分代码的复用性、测试起来也更加方便。拓展任务二则单独作为另一个项目，在群组成员沟通了实现的大致思路和实现逻辑，共同查阅资料，并敲定了实现的解决方案后，由杨成希负责最终代码的编写。

对于本组的团队协作效果，我和其他组内成员都感到基本满意，任务的分配较为平均，组员的工作量较为平均，且所有成员均能积极按质按量的完成任务，在自己遇到困难时能够及时提出、在他人遇到困难时能够热心的帮助他人。在这样的模式下，我们的效率较高，大体上完成了任务一内的所有任务。但是，本次的团队协作由于经验不足以及技术水平有限，故还是有所欠缺。例如在代码的同步的方式上，由于对 git 命令的使用还不太了解，故选用了最原始的文件传输的方式来同步组内的代码，这样显然是不方便且不够规范的，在日后的团队协作中，应该学会使用 git 提交自己所写的代码，力争做到规范、高效。

具体地说，本人（刘家麒）在此次综合训练中的具体任务主要是：完成拓展任务一的全部任务，包括实现变量的存储以及表达式中变量替换为数值的、变量的定义、赋值和释放的功能的实现。以及中缀表达式转后缀表达式、后缀表达式求值的功能的实现。此外，由于我组选择了任务一、三合并到一为一个项目的方案，故主函数和一些需要共用的辅助性函数是有本人和陈宇凡共同完成。在完成自己的任务的基础上，本人还有参与到组内讨论中，尽力帮助组内其他成员解决他们所遇到的问题。

4.2 项目成果

本项目的完成情况较高。就本人所负责的任务而言，实现了 C 语言中的大多数的运算符，且处理方式与 C 语言一致。在我代码的编写过程中，我遇到的第一个较大的问题就是如何增加除了四则运算外的其他运算符。由于一开始时只支持了四个运算符号且没有使用真值表技术，中缀转后缀时的能否将运算符继续压栈的操作通过分类讨论的方式实现，这样显然时不支持运算符的进一步拓展的，因为每新加一个运算符都要重新写判断能否压栈的函数，这样的问题困扰了我很久。在组员的帮助和我查阅了部分资料后，才将上述问题解决，具体思路为构造一个真值表，实现一个函数返回参数运算符的优先级，栈空、当新运算符优先级高于栈顶或栈顶为右括号时才能继续压栈，否则要先弹出栈顶运算符。正式由于方法的改进，我才能实现在工作量增长不大的情况下比原本多了近乎一倍的运算符。经历上述的问题，我更深刻的明白了方案的选择对项目开发的重要性、以及代码的可维护性、可拓展性的重要性。

本项目的变量名、函数名的命名较为规范。一开始时，我组成员在编写代码时均未在意命名的规范性问题，导致后续讨论、协同开发的时候彼此都不理解对方的代码，这给协作带来了巨大的困扰。为提高代码质量和提升开发效率，经过大家的交流下我组决定要尽量规范代码中的命名和写法,积极使用翻译软件查询恰当的命名,拒绝使用拼音、大片的单字母变量，以提升可读性。在全组人员的努力下，我组代码的可读性有了一定程度的提升，沟通协作的效率也提高了。可见，在日后的学习甚至是将来的工作中，一定要提高自己编写的代码的质量：采用更合适易懂的变量名、保证缩进的规范性、增加必要的注释等。

4.3 项目展望

此次项目还有许多不完善的地方，程序有不少理论上应该改进但由于自身水平不足和时间不够充分等原因未能完成的地方。

首先就是由于代码编写的时间跨度较长，所以代码有些部分较为啰嗦，且由于编写部分代码时没有考虑到后续的功能实现的要求，所以一开始时存在选用的实现方式不合理导致后续编写其他功能的代码的难度提升的问题。例如一开始为了没有周到考虑选用了 char*方式的字符串来存储表达式字符串，在一开始并没有遇到任何问题，但在编写变量的模块时，由于 std::map 的键值的类型不能是 char*的字符串，那么就意味着变量只能是单个字母或是无法使用 std::map 的两难的问题。为了尽力解决该问题，最后使用了 string 作为键值存储变量的字符串，在遍历用户输出的表达式时通过 string 的构造函数将 char*字符串作为参数构造 string 类型字符串，需要取得变量名的字符串时使用 string 的 c_str()方法。这种方法虽然解决了问题，但是却使得代码变得割裂和臃肿，只能算一个不太优雅的解决问题的方式。若是开始时就能预计到后续编码的需求，一开始就使用 string 存储表达式串，就能提高代码的统一度，也能优化部分模块的逻辑。

其次是对非法表达式的检测不完善，大部分的非法表达式都不能正确的检测出来。若输入的表达式有误，在进行后缀表达式求值时会因为因为存在非法栈操作而崩溃，即所以的非法表达式导致的程序崩溃都发生在后缀表达式计算时的对栈的过度弹栈、

取空栈的栈顶。原本的思路是通过捕获 std::stack 所抛出的异常随后在主函数进行异常处理并提示表达式非法，随后等待输入新的命令。但由于对 C++语言的使用还不够熟悉，在 std::stack 抛出了异常后始终不能正确捕获，导致此功能不能实现，从而对非法表达式的过滤能力有限。

还有就是由于一开始编写时确立的“命令令牌”+“操作”的交互方式，使得交互时对格式的要求较多，如赋值时必须为 let(空格)(变量名)（空格)=(空格)(表达式)。这使得如“++”、“=”以及“，”等运算符难以实现，但由于这样的交互方式在一开始编写的时候就已经确立，在后期发现时已经不好修改，所以就使得交互逻辑较为臃肿、扩展性较差。若一开始选用 get 函数读取一行，再对字符串进行语义分分析，获取命令令牌和操作，这样对格式的要求就会大大降低，也使得上述未能实现的运算符能更加容易地实现。出现这样的问题，主要是由于没能即使发现劣质的代码逻辑，导致发现时由于后续代码已经对前面的劣质逻辑有了依赖，不好修正，只能沿用，这就增加了后续编码的难度。

4.4 收获和感悟

我在本次的专业综合训练过程中受益匪浅，既学习到了很多拓展性的知识，又积累到了很多经验，还提升了自己的代码编写能力。

在本次的实训过程中，我学到了不少 C++语言的知识，例如 STL 的部分容器的使用、C++类和对象的基本知识和语法、通过 new 关键字调用类的有参构造函数创建一个对象，在类内提供一些方法对对象进行操作。可以说，C++语言的这些特性虽然有难度，但熟悉之后却比 C 语言更加方便、快捷，能简化代码的编写流程。其次，我的算法与数据结构水平也有了一定程度的提高，从原本的畏惧算法慢慢变得不畏再到愿意开始积极地动手尝试，可以说是克服了惧怕算法的心理恐惧。

其次，我也积累了许多的经验。最重要的就是编程相关的经验。虽然我一直使用的开发环境都是 Visual Studio 2022,但是限于以往的代码体谅过小，故 ide 的大部分功能都还不能的使用。在这次训练过程中，我逐渐能够较为熟练的使用断点进行调试，通过使用条件断点、合理使用逐语句、逐过程、跳出等执行方式，能够更快的定位问题的位置并解决问题。故虽然我的代码在编写过程中出现了许多错误，但是由于调试能力的提升，我最终还是较为顺利的完成了任务。另外，我也逐步的掌握了 ide 的重命名、多文件之间的跳转的方法，提高了编写代码时的便利性与降低了错误率。

最后，经历了这次实训，我的编程能力也有了较大的跨越。从原来的单兵作战到如今的团队合作，显然要更注重代码的规范性、正式性。以往写程序是总是所有代码都放在一个文件内、甚至直接写在 main 函数中，这样的写法很难完成本次实训较大体量的代码，也不利于团队的合作交流。故我也开始将代码分成多个文件，一个模块的代码放在一个文件中、一个原子功能写成一个函数。这样既能提升可读性，也能提升代码的复用性。

当然，纵使有了能力有提升，但我的水平还是有限的，在未来还需要更加刻苦认真的学习，努力使自己成为一名技术过硬、协作能力强的计算机专业的学生。

5 参考文献

[1]严蔚敏 / 吴伟民.数据结构（C 语言版）[M]. :清华大学出版社, 2012. -.

[2][美] 史蒂芬·普拉达（Stephen Prata）.C++ Primer Plus 第 6 版中文版[M]. :人民邮电出版社, 2020. -.

[3]周晓聪/乔海燕.离散数学基础[M]. :清华大学出版社, 2021. -.

6.致谢

时光飞逝，不知不觉大一一年的时光转瞬即逝，再开学即使大二。对于那些帮助我的同学、老师们，我的心中充满了无限的敬意，感谢他们在我大一两学期及综合训练过程中对我的帮助，鼓励与鞭策。

首先，我要感谢我大一学期的各位老师，正是因为他们的诲人不倦以及悉心指导，才是我从对计算机毫无认知的高考毕业生变为如今已经对计算机和编程有了一定认知的计算机专业的学生。没有他们的帮助，不可能学到这么多计算机专业领域的知识。

其次，我要感谢我团队的全体成员（杨成希、陈宇凡），正是因为我们的齐心协力、精诚合作，本次综合训练才得以圆满完成，每一个人对我们的团队的付出都值得我去诚恳的感谢。没有我们的共同努力，就没有我们最后的成果。

最后，我要感谢这个互联网高度发达、信息高度透明的时代，大大地方便了我学习新的知识。正是由于互联网上能够查阅到许多知识、浏览到许多教程，我才能按时地完成综合训练的任务。如果没有如此便利的学习方式，那么这次任务将会完成的更加费力。感谢我生在了这样一个先进发的的时代。

成绩记录

一、评语（根据软件成果质量，报告质量，答辩情况综合写出）。

二、评分

评分项目及分值	**软件成果**			**课程报告**		**陈述与答辩**		**合计（100 分）**
	任务达成（20 分）	系统与算法设计（20 分）	代码质量（10 分）	文献学习方案分析（20 分）	规范表达（10 分）	团队协作（10 分）	表述与回答（10 分）	
得　分								