项目说明文档

《离散数学》课程实验报告

——逻辑命题联结词、真值表、主范式

作 者 姓 名： 杨滕超

学 号： 2151298

指 导 教 师： 唐剑锋

学院、 专业： 软件学院 软件工程

同济大学

Tongji Universit

目 录

[1 题目简介](#_Toc495668153) 4

[1.1 题目原理背景](#_Toc495668154) 4

[1.1.1 命题](#_Toc495668163) 4

[1.1.2 命题逻辑联接词](#_Toc495668163) 4

[1.1.3 真值表](#_Toc495668163) 4

[1.1.4 主范式](#_Toc495668163) 4

[1.2 题目概述](#_Toc495668155) 5

[1.3 程序功能](#_Toc495668154) 5

[1.3.1 A题目](#_Toc495668163) 5

[1.3.2 B题目](#_Toc495668163) 5

[1.3.3 C题目](#_Toc495668163) 5

[2 解题思路](#_Toc495668156) 5

[2.1 A题目](#_Toc495668154) 5

[2.2 B题目](#_Toc495668154) 6

[2.3 C题目](#_Toc495668154) 6

[3 数据结构设计](#_Toc495668161) 6

[3.1 解决A题目的类](#_Toc495668162) 6

[3.1.1 说明](#_Toc495668167) 6

[3.1.2 UML图](#_Toc495668167) 6

[3.2 解决BC题目的类](#_Toc495668166) 7

[3.2.1 说明](#_Toc495668167) 7

[3.2.2 UML图](#_Toc495668167) 7

[4 核心算法](#_Toc495668161) 8

[4.1 A题目输入](#_Toc495668174) 8

[4.1.1 流程图](#_Toc495668176) 8

[4.1.2 说明](#_Toc495668176) 8

[4.1.3 主要代码](#_Toc495668176) 9

[4.2 A题目运算与输出](#_Toc495668174) 9

[4.2.1 说明](#_Toc495668176) 9

[4.2.2 主要代码](#_Toc495668176) 10

[4.3 A题目是否继续下一轮](#_Toc495668174) 11

[4.3.1 流程图](#_Toc495668176) 11

[4.3.2 说明](#_Toc495668176) 11

[4.3.3 主要代码](#_Toc495668176) 12

[4.4 A题目解决方法整合](#_Toc495668174) 13

[4.4.1 流程图](#_Toc495668176) 13

[4.4.2 说明](#_Toc495668176) 13

[4.4.3 主要代码](#_Toc495668176) 14

[4.5 BC题目求2的n次方](#_Toc495668174) 14

[4.5.1 说明](#_Toc495668176) 14

[4.5.2 主要代码](#_Toc495668176) 14

[4.6 BC题目判断表达式是否合法](#_Toc495668174) 15

[4.6.1 流程图](#_Toc495668176) 15

[4.6.2 说明](#_Toc495668176) 16

[4.6.3 主要代码](#_Toc495668176) 16

[4.7 BC题目从表达式中获得命题](#_Toc495668174) 18

[4.7.1 说明](#_Toc495668176) 18

[4.7.2 主要代码](#_Toc495668176) 19

[4.8 BC题目二进制加一](#_Toc495668174) 20

[4.8.1 流程图](#_Toc495668176) 20

[4.8.2 说明](#_Toc495668176) 20

[4.8.3 主要代码](#_Toc495668176) 21

[4.9 BC题目计算命题公式的值](#_Toc495668174) 22

[4.9.1 流程图](#_Toc495668176) 22

[4.9.2 说明](#_Toc495668176) 22

[4.9.3 主要代码](#_Toc495668176) 23

[4.10 BC题目对外接口](#_Toc495668174) 26

[4.10.1 流程图](#_Toc495668176) 26

[4.10.2 说明](#_Toc495668176) 27

[4.10.3 主要代码](#_Toc495668176) 27

[5 实验结果](#_Toc495668161) 30

[5.1 A题目](#_Toc495668174) 30

[5.1.1 进入界面](#_Toc495668176) 30

[5.1.2 正确运算结果](#_Toc495668176) 30

[5.1.3 错误输入](#_Toc495668176) 31

[5.1.4 正确退出](#_Toc495668176) 31

[5.1.5 错误选择](#_Toc495668176) 32

[5.2 BC题目](#_Toc495668174) 32

[5.2.1 进入界面](#_Toc495668176) 32

[5.2.2 基本运算](#_Toc495668176) 33

[5.2.3 复杂的命题公式](#_Toc495668176) 35

[5.2.4 带括号的命题公式](#_Toc495668176) 36

[5.2.5 仅有一个命题](#_Toc495668176) 36

[5.2.6 输入带有空格](#_Toc495668176) 37

[5.2.7 括号不匹配](#_Toc495668176) 37

[5.2.8 缺少操作数](#_Toc495668176) 37

[5.2.9 缺少操作符](#_Toc495668176) 37

[5.2.10 没有输入](#_Toc495668176) 37

[5.2.11 非法字符](#_Toc495668176) 38

[6 心得体会](#_Toc495668161) 38

1 题目简介

* 1. 题目原理背景
     1. 命题

命题分为两类，一类是不能再分解为更简单命题的命题，这种命题称为简单命题；另一类是可以分解为更简单命题的命题，这种命题称为复合命题。

在命题演算中，用符号A、B、C…或A1，A2，B1，B2，…等等表示简单命题。当命题A取值“真”时，又称A具有值T(True)，当命题A取值“假”时，又称A具有值F(False)，T和F称为命题常数。为了方便也可将T记为1，将F记为0。现在引入逻辑连接词，并通过它们从简单命题 “构造”复合命题。

* + 1. 命题逻辑联接词

下面介绍几种常用的命题逻辑联接词。

合取：将两个命题P、Q联结起来，构成一个新的命题P∧Q, 读作P、Q的合取, 也可读作P与Q。这个新命题的真值与构成它的命题P、Q的真值间的关系为只有当两个命题变项P = T, Q = T时方可P∧Q =T, 而P、Q只要有一方为F则P∧Q = F。P∧Q可用来表示日常用语P与Q, 或P并且Q。

析取：将两个命题P、Q联结起来，构成一个新的命题P∨Q, 读作P、Q的析取, 也可读作P或Q。这个新命题的真值与构成它的命题P、Q的真值间的关系为只有当两个命题变项P = F, Q = F时方可P∨Q =F, 而P、Q只要有一为T则P∨Q = T。P∨Q可用来表示日常用语P或者Q。

条件：将两个命题P、Q联结起来，构成一个新的命题P→Q, 读作P条件Q, 也可读作如果P，那么Q。这个新命题的真值与构成它的命题P、Q的真值间的关系为只有当两个命题变项P = T, Q = F时方可P→Q =F, 其余均为T。

双向条件：将两个命题P、Q联结起来，构成一个新的命题P←→Q, 读作P双条件于Q。这个新命题的真值与构成它的命题P、Q的真值间的关系为当两个命题变项P = T, Q =T时方可P←→Q =T, 其余均为F。

* + 1. 真值表

真值表：表征逻辑事件输入和输出之间全部可能状态的表格。列出命题公式真假值的表。通常以1表示真，0表示假。命题公式的取值由组成命题公式的命题变元的取值和命题联接词决定，命题联接词的真值表给出了真假值的算法。真值表实在逻辑中使用的一类数学表，用来确定一个表达式时候为真或有效。

* + 1. 主范式

主范式包括主析取范式和主合取范式，其中主析取范式为：在含有n个命题变元的简单合取式中，若每个命题变元与其否定不同时存在，而两者之一出现一次且仅出现一次，则称该简单合取式为极小项。由若干个不同的极小项组成的析取式称为主析取式；与A等价的主析取范式称为A的主析取范式。任意含n个命题变元的非永假命题公式A都存在与其等价的主析取范式，并且是唯一的。

主合取范式为：在含有n个命题变元的简单析取式中，若每个命题变元与其否顶不同时存在，而两者之一出现一次且仅出现一次，称该简单析取式为极大项。由若干个不同的极大项组成的合取式称为主合取式；与A等价的主合取范式称为A的主合取范式。任意含n个命题变元的非永真命题公式A都有与其等价的主合取范式，并且是唯一的。

* 1. 题目概述

本实验课程训练学生掌握命题逻辑中的联接词、真值表、主范式等，进一步能用它们来解决实际问题。通过实验提高学生编写实验报告、总结实验结果的能力；使学生具备程序设计的思想，能够独立完成简单的算法设计和分析。利用C/C＋＋语言，实现二元合取、析取、条件和双向条件表达式的计算。充分利用联接词和逻辑运算符之间的相似性来实现程序功能。

* 1. 程序功能
     1. A题目

从键盘输入两个命题变元P和Q的真值，求它们的合取、析取、条件和双向条件的真值。

* + 1. B题目

通过键盘输入，并对输入的合法性进行判断，得到一个命题公式，求出该命题公式的真值表（B）。利用C/C＋＋语言，实现任意输入公式的真值表计算。一般将公式中的命题变元放在真值表的左边，将公式的结果放在真值表的右边。命题变元可用数值变量表示，合式公式的表示及求真值表转化为逻辑运算结果。

* + 1. C题目

通过B题目中所求的真值表，求出该命题公式的主析取范式和主合取范式。

2 解题思路

2.1 A题目

从利用C++语言中自带的逻辑运算&&、||、！表示离散数学中的命题逻辑联接词：与、或、非。而对于不是这三种命题联接词中的一种的可以表示的例如蕴含和双向蕴含，可以利用它们的组合间接表示。其中，P->Q可以等价地表示为(!P)||Q；P<->Q可以等价地表示为((!P)||Q)&&((!Q)||P)。

2.2 B题目

首先我们需要对输入的命题公式是否合法进行判断，然后从式子查找出命题变量的个数。根据该个数我们可以开辟一个一位数组存储二进制数，用于生成真值表，每循环一次就进行二进制数的加一操作。然后利用函数进行命题公式的求值运算，并输出结果。

2.3 C题目

根据B所求得得真值表，我们可以判断对应命题序列所导出得命题公式值为T的为主析取范式中的极小项，反之为主合取范式的极大项，由此将其收集起来，从而得到主析取范式和主合取范式。

3 数据结构设计

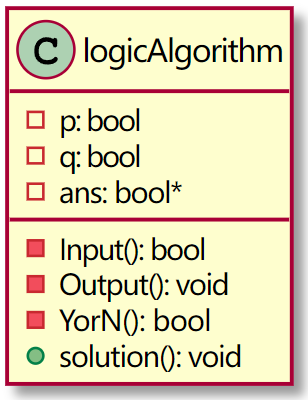
3.1 解决A题目的类

3.1.1 说明

该类的数据成员包括两个布尔型变量，用于存储用户输入的命题变元的值，并利用一个布尔型指针用于申请数组空间存储命题逻辑运算的结果。

成员函数包括用户输入函数，输出函数以及是否决定是否继续下一轮输入的函数。

3.1.2 UML图



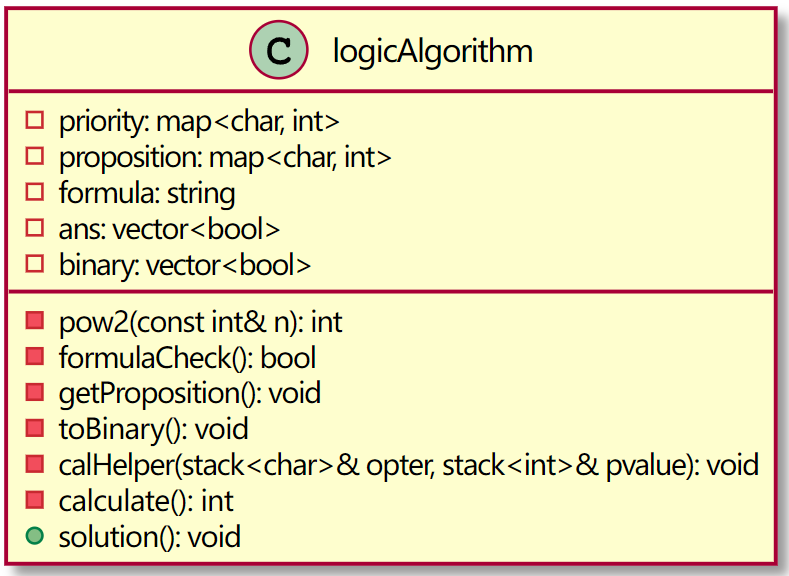
3.2 解决BC题目的类

3.2.1 说明

为了完成真值表的计算以及主范式的求取，该类需要成员变量有：map<char, int> priority存储运算符号的优先级、map<char, int> proposition存储命题变元以及对应的编号、string类型的formula用于存储输入的表达式、vector<bool> binary存储真值表的中当前遍历的行的各个命题变元的赋值、vector<bool> ans每组赋值对应的命题公式的值。

成员函数包括：计算2的n次方的值的函数、检查输入的表达式是否合法的函数、从表达式中记录命题变元的函数、二进制表示加一的函数、计算命题公式的值的函数以及对外接口。

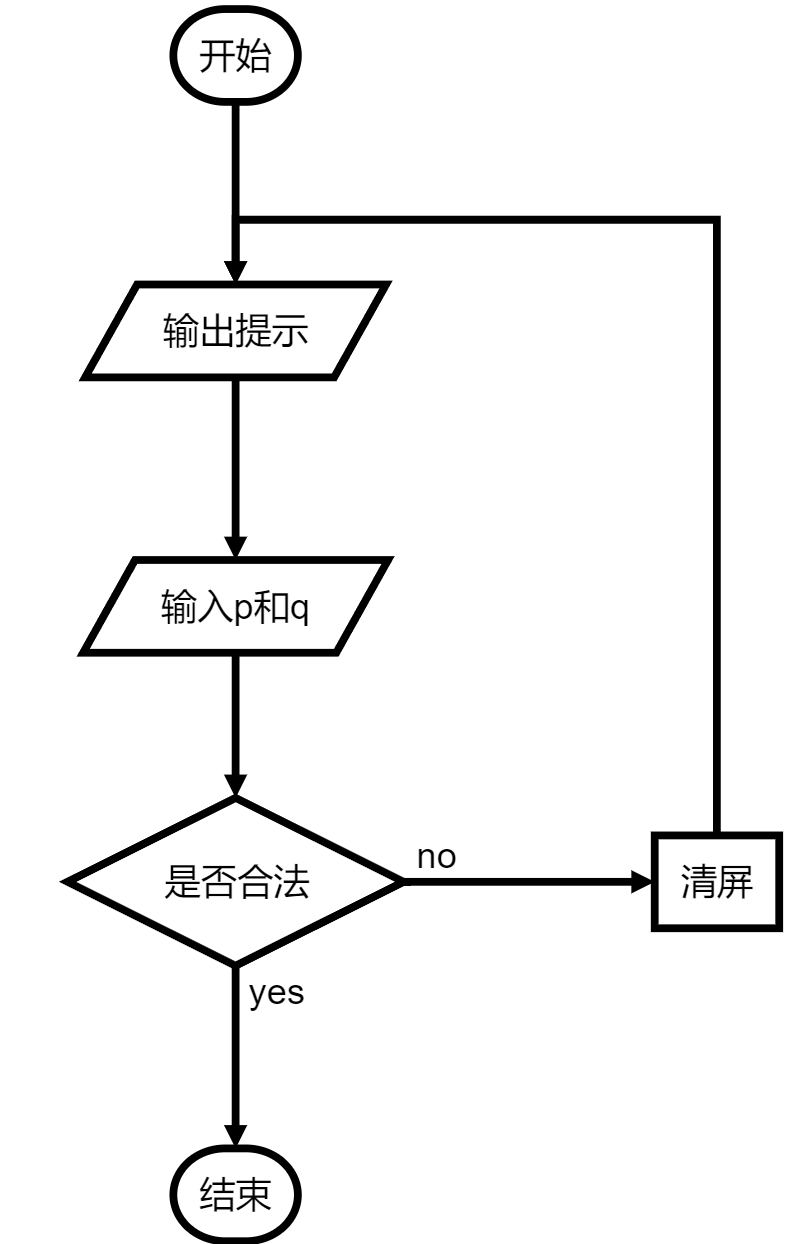
3.2.2 UML图



4 核心算法

4.1 A题目输入

4.1.1 流程图



4.1.2 说明

本函数的主要功能为接收用户输入的的命题变元p和q的值。这里我们需要对用户的输入进行输入错误判断，若用户输入了不是0或1的值，需要利用cin.clear()函数和cin.ignore()函数进行错误标志位的清除与跳过。同时，这里输出输入错误的提示，并暂停程序两秒使得用户接受到错误提示。然后条用清屏的命令，重新请用户进行输入。

4.1.3 主要代码

bool Input() {

cout << "\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*" << endl;

cout << "\*\* \*\*" << endl;

cout << "\*\* 欢迎进入逻辑运算程序 \*\*" << endl;

cout << "\*\* \*\*" << endl;

cout << "\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*" << endl;

//p的输入

cout << endl << " 请输入P的值（0或1）,以回车结束:";

cin >> p;

if (cin.fail()) {

cout << endl << " P的值输入有误,请重新输入!";

cin.clear();

cin.ignore(65536, '\n');

Sleep(2000);

system("cls"); //清屏

return true;

}

cout << endl << " 请输入Q的值（0或1）,以回车结束:";

cin >> q;

if (cin.fail())

{

cout << endl << " Q的值输入有误,请重新输入!";

cin.clear();

cin.ignore(65536, '\n');

Sleep(2000);

system("cls"); //清屏

return true;

}

//输入正确

return false;

}

4.2 A题目运算与输出

4.1.1 说明

对于四个逻辑运算，利用C++中的布尔运算符，均可以直接或间接地实现，例如与可以利用p && q、或可以利用p || q、蕴含运算可以等价为(!p) || q以及等值运算可以等价为((!p) || q) && ((!q) || p)。将运算结果存储在数组中，在其他情况需要的时候仍可以使用。

4.1.2 主要代码

void Output() {

//计算逻辑运算并输出

ans[0] = p && q; //与运算

ans[1] = p || q; //或运算

ans[2] = (!p) || q; //蕴含运算，将其转化为与或非形式

ans[3] = ((!p) || q) && ((!q) || p); //等值运算，将其转化为与或非形式

cout << "\n\n 合取:\n P/\\Q = " << ans[0] << "\n"; //输出结果

cout << " 析取:\n P\\/Q = " << ans[1] << "\n";

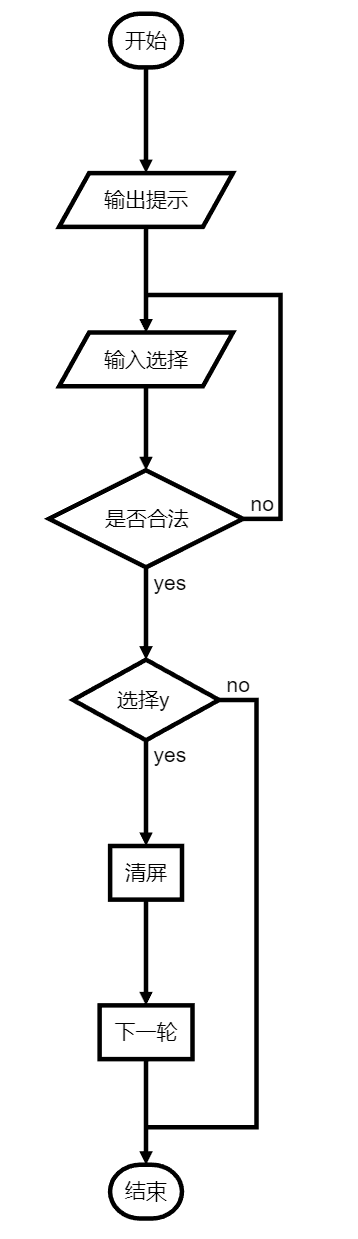
cout << " 条件:\n P->Q = " << ans[2] << "\n";

cout << " 双条件:\n P<->Q = " << ans[3] << "\n";

}

4.3 A题目是否继续下一轮

4.3.1 流程图



4.3.2 说明

本函数由用户输入y或者n决定程序是否继续运行。这里对输入的要求进行一定的放宽，当用户输入大写字母Y和N也是没有问题的。同时我们也需要对输入进行判断。对于非法输入需要利用cin.clear()函数和cin.ignore()函数进行错误标志位的清除与跳过。用户若是选择了继续运算，则调用系统的清屏函数进行清屏，然后返回true值表示继续外层循环，重新开始；用户若是选择了不继续运算，则返回false值表示结束外层循环。

4.3.3 主要代码

bool YorN() {

//暂时接受用户输入的字符型变量

char temp = '\0';

while (1)

{

cout << "\n是否继续运算?（y/n）"; //询问是否继续操作

cin >> temp;

//输入正确

if (temp == 'y' || temp == 'n'

|| temp == 'Y' || temp == 'N')

{

//进行下一轮循环

if (temp == 'Y' || temp == 'y')

{

system("cls"); //清屏

break;

}

//结束

else

{

cout << "欢迎下次再次使用!\n"; //退出

break;

}

}

//输入错误

else

{

if (cin.fail())

{

cin.clear();

cin.ignore(65536, '\n');

}

cout << "输入错误，请重新输入!" << endl;

}

}

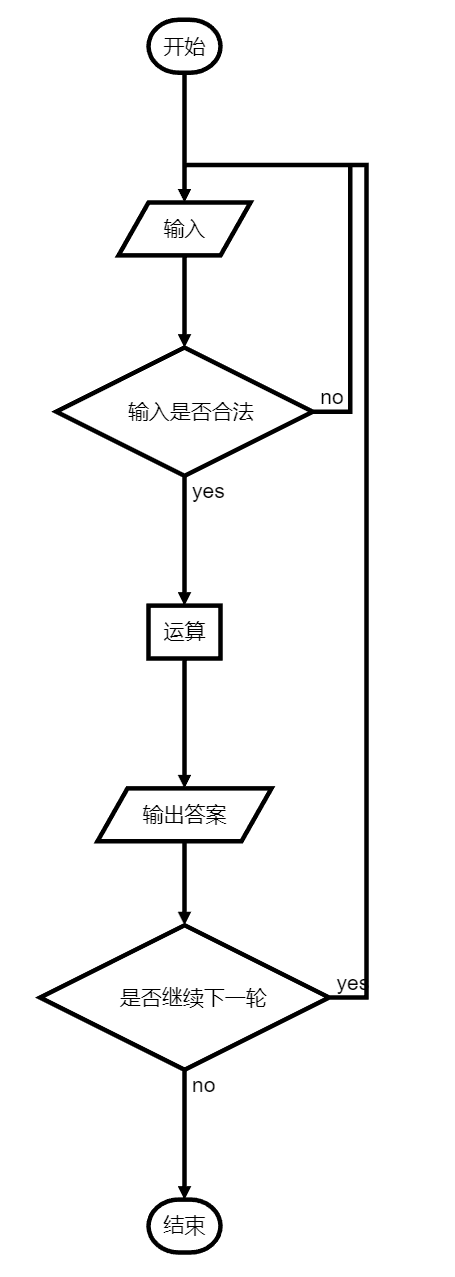
//如果是yes就返回true，表示继续下一轮

return temp == 'Y' || temp == 'y';

}

4.4 A题目解决方法整合

4.4.1 流程图



4.4.2 说明

将上述函数整合进入一个方法，作为对外的接口。利用布尔型变量loop控制最外层的循环。对于第一个步骤——输入，我们需要得到输入函数的返回值，同样声明一个布尔型变量inputFlag记录返回值，利用循环调用input()输入函数。若输入函数的返回值为false，说明输入正确，结束循环；反之输入错误，需要重新输入。第二个步骤为运算并输出，最后调用判断是否进行下一轮的函数，利用其返回值更新loop的值，由此决定最外层循环是否继续。最后若结束循环，需要利用cin.get()函数等待用户输入任意键，之后在关闭cmd窗口。

4.4.3 主要代码

void solution()

{

bool loop = true;

while (loop)

{

//设置输入标志，接受输入函数的返回值

bool inputFlag = true;

//返回值为false说明输入正确，跳出循环

while (inputFlag)

inputFlag = this->Input();

//输出答案

this->Output();

//接受返回参数决定是否继续下一轮循环

loop = this->YorN();

}

//等待用输入输入再关闭cmd窗口

cin.clear();

cin.ignore(65536, '\n');

cout << "请按任意键退出..." << endl;

cin.get();

}

4.5 BC题目求2的n次方

4.5.1 说明

当我们得到n个命题变元的时候，我们需要2n种不同的赋值来构建真值表。因此这里利用n词循环，以求得2n的值，并且作为int整型变量返回。考虑到

4.5.2 主要代码

int pow2(const int& n)

{

int ans = 1;

for (int i = 0; i < n; ++i)

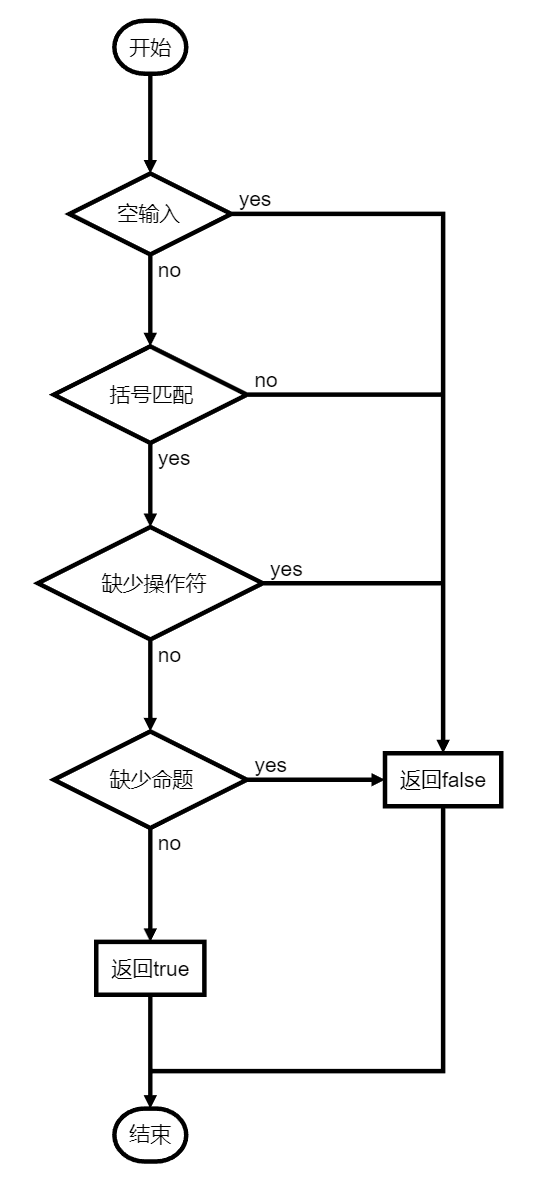
ans \*= 2;

return ans;

}

4.6 BC题目判断表达式是否合法

4.6.1 流程图



4.6.2 说明

这里对表达式判断是否合法。

进入此函数之前，我们首先将表达式中的空格去除，因此其中的字符只剩下关于运算的命题变元和运算符，无需考虑空格等其他字符。

利用循环，检查每次遍历到的字符是否为非法字符，不是非法字符才进行一下的判断，否则直接返会false值表示表达式不合法。

首先是对于括号的匹配问题，利用一个整型变量cnt记录遍历到的左括号的个数，若遇到右括号，则需要检查cnt是否大于零，若大于零则说明仍有左括号与其配对，进行--cnt操作表示配对完成；若小于等于零则说明没有左括号与当前遇到的右括号配对，右括号过多，配对失败。最后检查cnt是否等于零，等于零说明配对完成，若大于零说明左括号过多，配对失败。

然后是关于运算符缺少的问题，这里利用一个布尔类型的变量记录上一个遇到的字符是否为命题变元。若上次遇到英文字母即命题变元，本次遇到的还是命题变元，说明两个命题变元之间缺少运算符，表达式不合法。

对于操作符使用不规范问题，这里也做出了判断。我们分两类讨论，一类是二元运算符，左右两边需要两操作数，因此遍历到的时候需要检查其两边是否格子存在两个命题。这里的命题可以是单个英文字母，可以是利用括号包括起来的复合命题。同时我们还需要对出先左右括号中间没有表达式的情况进行判断，例如：“（）”，这样显然是没有意义且不和规范的。

最后我们也不能忽略有可能存在空表达式的情况，这里也需要做出判断。

4.6.3 主要代码

bool formulaCheck()

{

//空输入

if (formula.length() == 0)

return false;

bool prevIsPro = false;

//记录左括号

int cnt = 0;

for (size\_t i = 0; i < formula.length(); ++i)

{

auto ch = formula[i];

//非法字符，不是运算符也不是命题变元

if (!isalpha(ch) && !priority.count(ch))

return false;

//命题变元

if (isalpha(ch))

{

//两个命题变元在一起

if (prevIsPro)//上一个也是命题

return false;

//记录本次为命题

prevIsPro = true;

}

else

{

//左括号计数

if (ch == '(')

++cnt;

//右括号匹配

else if (ch == ')')

{

//左括号缺少

if (cnt <= 0)

return false;

--cnt;

}

switch (ch)

{

//二元运算符

case '&':

case '|':

case '^':

case '~':

if (!prevIsPro && i > 0 && formula[i - 1] != ')')//两个运算符在一起

return false;

if (i < formula.length() - 1 && !isalpha(formula[i + 1]) && formula[i + 1] != '(')

return false;//判断后置命题变项

if (i == formula.length() - 1 || i == 0)

return false;

break;

//一元

case '!':

if (prevIsPro)//前面是命题变元

return false;

if (i == formula.length() - 1)//缺少命题变元

return false;

break;

case '(':

if (i == formula.length() - 1 || formula[i + 1] == ')')//出现()中间没有表达式

return false;

default:

break;

}

//更新记录上一个是否命题

prevIsPro = false;

}

}

//检查是否左括号冗余

return cnt == 0;

}

4.7 BC题目从表达式中获得命题

4.7.1 说明

这里利用for循环遍历表达式中的每一个字符，同时利用map<char, int>记录命题变元，以保证记录的不重复性。这是因为考虑到一个命题公式中可能会产生同一个命题变元出现多次的情况，因此我们可以利用基于红黑树的map结构进行以代表命题变元的英文字母为关键码进行记录，顺便赋给每个命题变元独一无二的编号以便后面进行计算。

但同时我们知道利用红黑树单纯地记录英文字母可能并不是那么的“划算”，红黑树需要进行调整、着色和旋转等操作，在时间和空间上存在一定的必要开销。因此对于英文字母，可以根据字符ASCII码的特点，将其对应地转换为数组的索引。从而完成char到int的映射。具体实现如下：

int proposition[26 \* 2];//包括大小写的26个字母

int index = 0;//编号

for (auto& ch : formula)

{  
 if (islower(ch))

proposition[ch - ‘a’] = ++index;

else if (isupper(ch))

proposition[ch - ‘A’+ 26] = ++index;

}

当然为了对stl库中的熟练掌握以及整个程序代码的一致性，还是利用map<char, int>进行实现。

4.7.2 主要代码

void getProposition()

{

int index = 0;

for (auto& ch : formula)

{

if (isalpha(ch))//是一个英文字母

{

//还不存在

if (proposition.count(ch) == 0)

{

//插入哈希表

proposition.insert(make\_pair(ch, index++));

}

}

//不属于合法字符

else if (priority.count(ch) == 0)

{

cout << ch << "is undefined!" << endl;

exit(2);

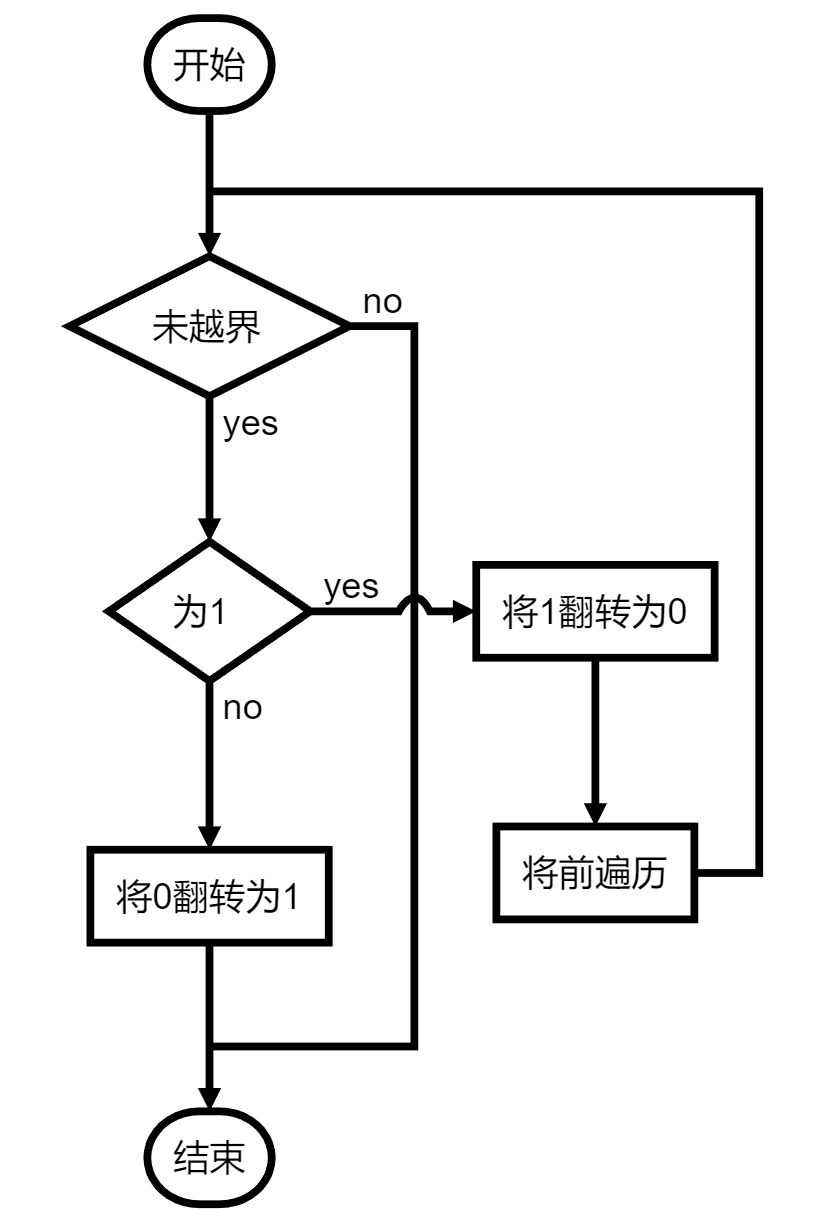
}

}

}

4.8 BC题目二进制加一

4.8.1 流程图



4.8.2 说明

这里利用另一种方法进行对向量数组中的二进制进行加一的方法。其主要思想为从后向前遍历，遇到第一个0则将其变为1，并结束该函数；若遇到1则将其变为0，继续向前遍历，直到遇到0或者越界为止。

该方法较为直接地利用数组的性质，而非利用数学方法进行计算，因此理伦上具有较高的效率。

举例：若数组中存储的值为0111

1. 第一轮改变：0110
2. 第二轮改变：0100
3. 第三轮改变：0000
4. 第四轮高边：1000

4.8.3 主要代码

void toBinary()

{

//这里使用size\_t无符号的整形

//i == 0 再--i，并不会出现负数

for (size\_t i = binary.size() - 1;; --i)

{

//最后一个零变为1结束

if (binary[i] == 0)

{

binary[i] = 1;

break;

}

//遇到1一直变为0

else

{

binary[i] = 0;

}

//到第一个结束循环

if (i == 0)

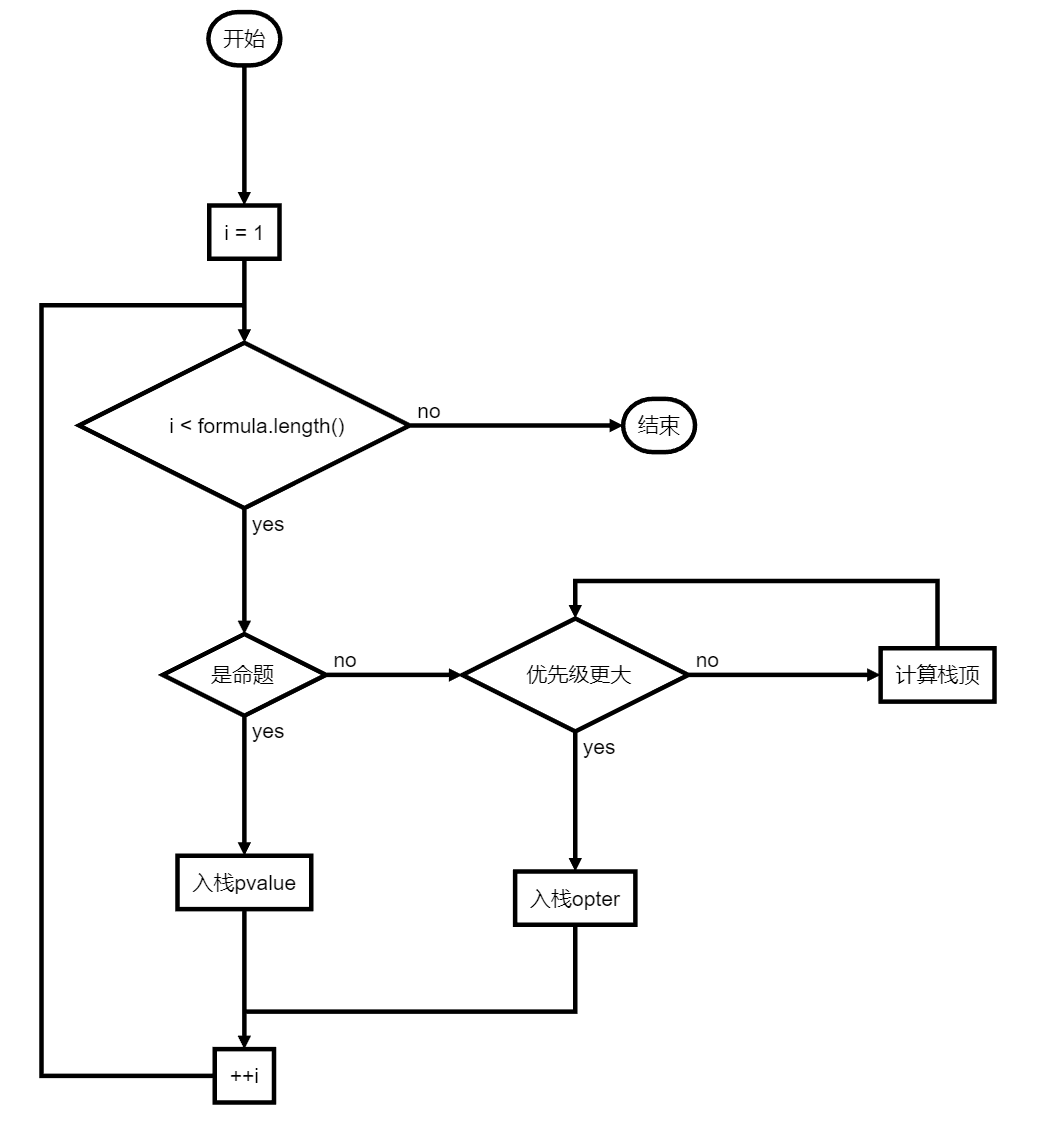
break;

}

}

4.9 BC题目计算命题公式的值

4.9.1 流程图



4.9.2 说明

该部分分为两个函数，一个数主要的计算函数，另一个是计算函数中用到的辅助函数。这里利用两个栈进行辅助计算，一个栈opter存储操作符，另一个栈存储命题的值。这里在表达式的最后追加一个#表示计算结束。然后开始对表达式正向遍历，如果遇到命题变元则将其对应的值加入栈；遇到操作符需要和操作符栈的顶部元素进行优先级的比较，同时注意栈顶是左括号需要特殊判断，因为左括号在入栈前后的优先级会从最高变为最低。比较的过程中，若当前的操作符的优先级比栈顶操作符的优先级大，则说明可以入栈；反之需要对栈顶的操作符运算，直到当前操作符的优先级更大，才能入栈。

其中对于操作符的运算需要区分是一元操作符还是二元操作符。若是一元操作符则需要从栈中取出两个操作数，并进行相应运算，然后入栈。而一元操作符则只需要取出一个操作数运算再加入回栈中。而对于遇到右括号的情况，我们需要调用递归函数，计算括号中的值，直到遇到左括号匹配为止。

4.9.3 主要代码

void calHelper(stack<char>& opter, stack<int>& pvalue)

{

//计算值的栈顶

int temp1 = 0, temp2 = 0;

switch (opter.top())

{

case '&':

temp1 = pvalue.top();

pvalue.pop();

temp2 = pvalue.top();

pvalue.pop();

pvalue.emplace(temp1 && temp2);

opter.pop();

break;

case '|':

temp1 = pvalue.top();

pvalue.pop();

temp2 = pvalue.top();

pvalue.pop();

pvalue.emplace(temp1 || temp2);

opter.pop();

break;

case '!':

temp1 = pvalue.top();

pvalue.pop();

pvalue.emplace(!temp1);

opter.pop();

break;

case '^':

temp1 = pvalue.top();

pvalue.pop();

temp2 = pvalue.top();

pvalue.pop();

pvalue.emplace(!temp1 || temp2);

opter.pop();

break;

case '~':

temp1 = pvalue.top();

pvalue.pop();

temp2 = pvalue.top();

pvalue.pop();

pvalue.emplace((!temp1 || temp2) && (!temp2 || temp1));

opter.pop();

break;

case '(':

case'#'://结束

break;

case ')'://计算括号内的，递归调用

opter.pop();

while (opter.top() != '(')

calHelper(opter, pvalue);

if (opter.top() == '(')

opter.pop();

break;

}

}

int calculate()

{

stack<char> opter;//操作符

stack<int> pvalue;//命题值

opter.push('#');

formula += "#";

for (auto& ch : formula)

{

if (isalpha(ch))

{

//将ch字符对应的命题变元的值加入栈中

pvalue.emplace(binary[proposition[ch]]);

}

else

{

//此时为运算符

auto temp = opter.top();

//栈顶优先级大，需要先计算

if (priority[temp] > priority[ch])

{

while (priority[temp] > priority[ch] && temp != '(')

{

calHelper(opter, pvalue);

temp = opter.top();

//计算完成到最后的#

if (temp == '#' && ch == '#')

return pvalue.top();

}

opter.emplace(ch);

}

//当前遍历到的运算符优先级大

//直接入栈

else

opter.emplace(ch);

}

}

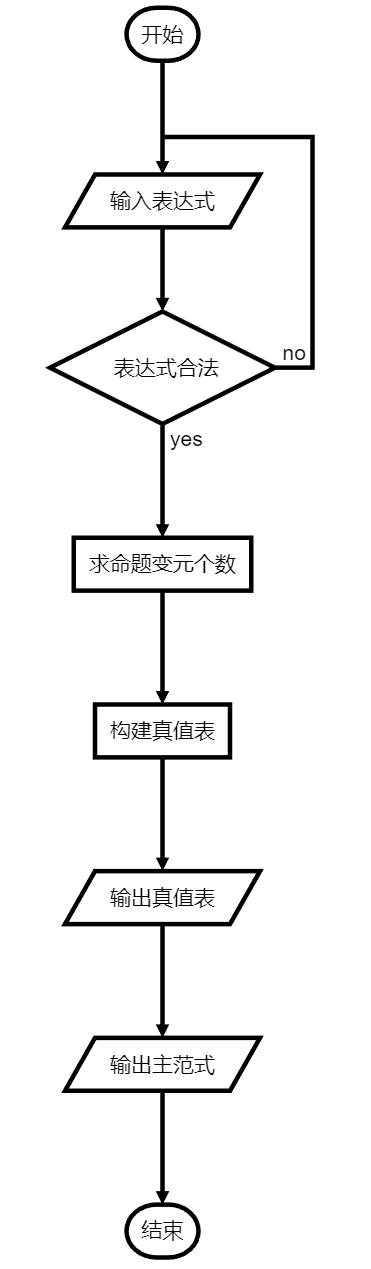
//计算出错

return -1;

}

4.10 BC题目对外接口

4.10.1 流程图



4.10.2 说明

这里首先对表达式进行输入。调用的是fgets函数，利用标准输入，可以进行空格和换行的读取，因此虽然提示说不能包含空格，但用户执意输入空格本程序也是可以接受的。因此输入完成后我们需要对表达式进行预处理——删除其中的空格，这里利用string类中的erase(std::remove\_if(formula.begin(), formula.end(), isspace), formula.end());其中isspace表示空字符，其中包括：空格、换行等等。然后我们进行表达式是否合法的判断，若不合法则输出提示，请用户重新输入。

然后调用函数获得表达式中命题的个数，以求得真值表的大小ansSize。根据改大小利用循环，遍历每个二进制序列对应的命题公式的真值，并进行输出和存储。

最后通过真值表，使得命题公式为1的二进制序列所对应的即为极小项；反之为极大项。从而得到主范式。

最后利用cin.get()函数等待用户输入任意键，再关闭cmd窗口。

4.10.3 主要代码

void solution() {

cout << "\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\n";

cout << "\*\* \*\*\n";

cout << "\*\* 欢迎进入逻辑运算软件 \*\*\n";

cout << "\*\* (可运算真值表,主范式,支持括号) \*\*\n";

cout << "\*\* \*\*\n";

cout << "\*\* 用!表示非 \*\*\n";

cout << "\*\* 用&表示与 \*\*\n";

cout << "\*\* 用|表示或 \*\*\n";

cout << "\*\* 用^表示蕴含 \*\*\n";

cout << "\*\* 用~表示等值 \*\*\n";

cout << "\*\* \*\*\n";

cout << "\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\n\n";

cout << "Please enter a legitimate proposition formula（不能包含空格）: " << endl;

char buffer[255] = { '\0' };

while (1) {

//可用于空格输入，标准输入

fgets(buffer, 255, stdin);

formula = buffer;

//删除所有空格，isspace表示空白字符包括空格、\n、\t等等

formula.erase(std::remove\_if(formula.begin(), formula.end(), isspace), formula.end());

//检查合法性

if (formulaCheck() == false)

cout << "命题不合法！请重新输入！" << endl;

//输入合法

else

break;

}

//得到表达式中的命题变元

getProposition();

cout << "该式子中的变量个数为：" << proposition.size() << endl << "输出真值表如下：" << endl;

//输出所有命题变元

for (auto it = proposition.begin(); it != proposition.end(); ++it)

cout << it->first << '\t';

cout << endl << "读入的式子为：" << formula << endl;

//真值表大大小

int ansSize = pow2(proposition.size());

//存储真值表答案

ans.resize(ansSize, false);

//转换为二进制的真值

binary.resize(proposition.size(), 0);

for (int i = 0; i < ansSize; ++i)

{

//先得到二进制

if (i != 0)//一开始都是0，无需加一

this->toBinary();

//输出当前各个命题变元的值

for (auto& val : binary)

cout << val << '\t';

//计算结果

ans[i] = static\_cast<bool>(this->calculate());

cout << ans[i] << endl;

}

int n\_m = 0, n\_M = 0;

cout << "该命题公式的主析取范式：" << endl;

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

{

if (ans[i])

{

if (n\_m == 0)

cout << "m<" << i << ">";

else

cout << " \\/ m<" << i << "> ";

n\_m++;

}

}

//没有主析取范式

if (n\_m == 0)

cout << "0";

cout << endl;

cout << "该命题公式的主合取范式：" << endl;

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

{

if (!ans[i])

{

if (n\_M == 0)

cout << "M<" << i << ">";

else

cout << " /\\ M<" << i << "> ";

n\_M++;

}

}

if (n\_M == 0)

cout << "0";

cout << endl;

cout << "请按任意键退出..." << endl;

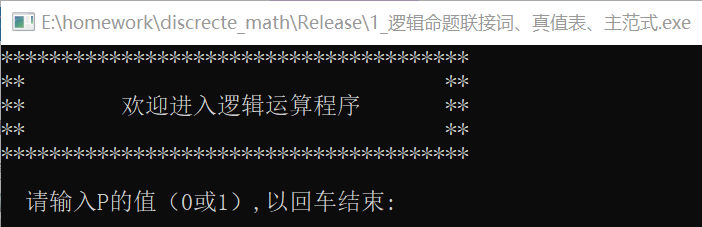
cin.get();

}

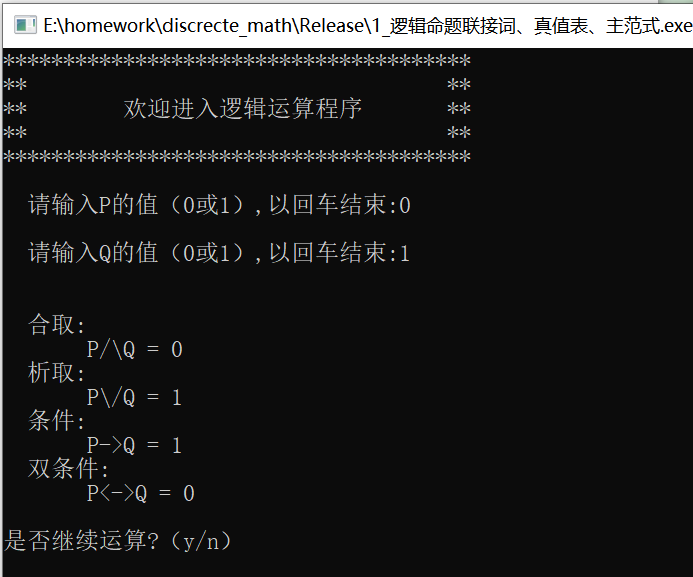
5 实验结果

5.1 A题目

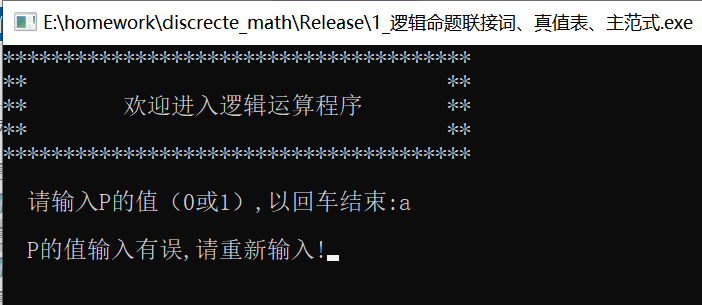
5.1.1 进入界面

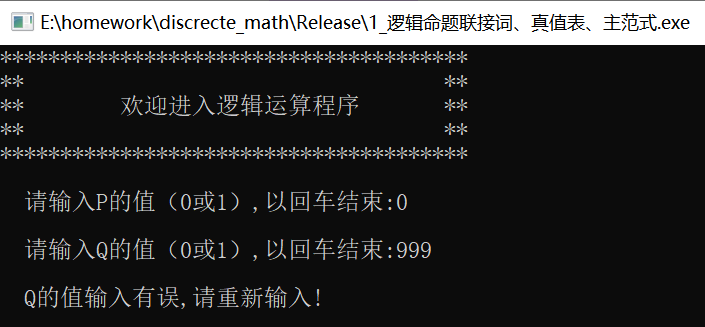


5.1.2 正确运算结果



5.1.3 错误输入

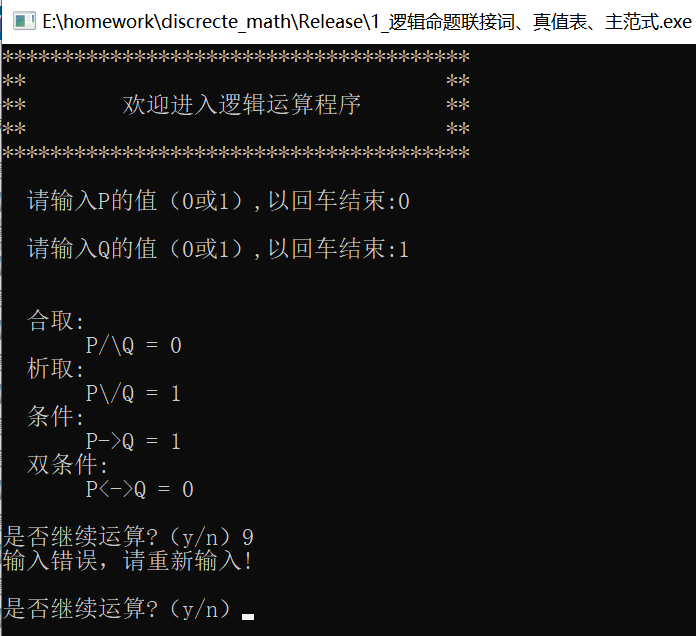




5.1.4 正确退出

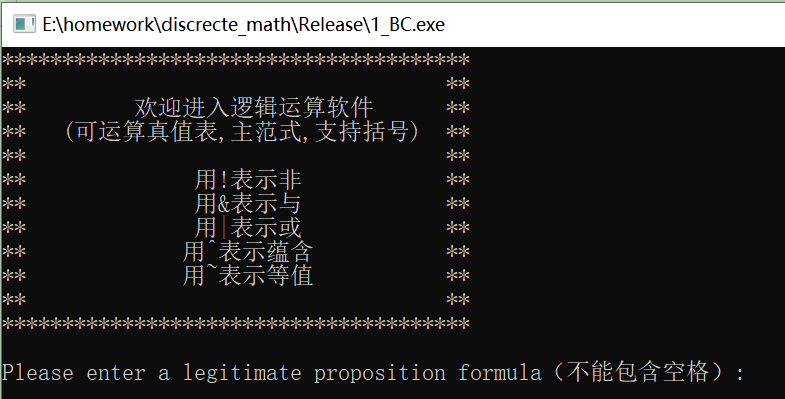


5.1.5 错误选择

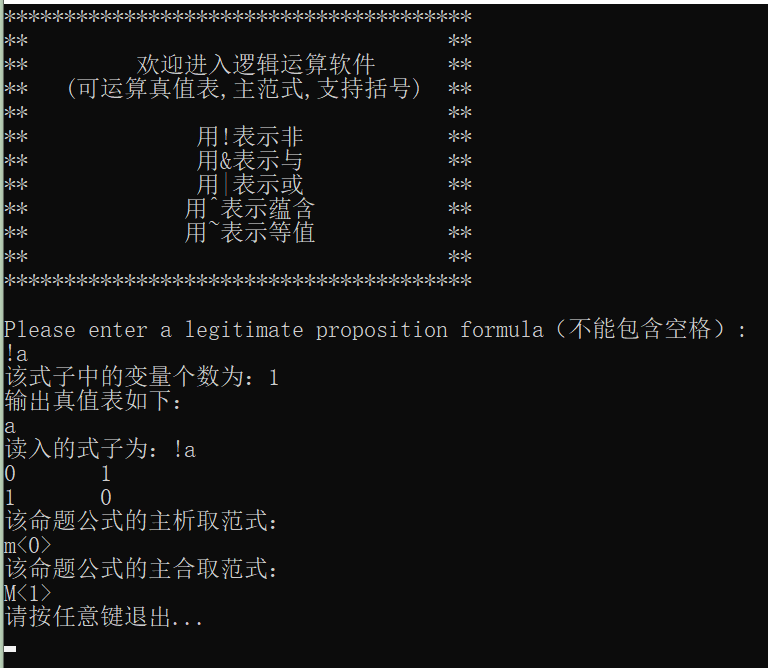


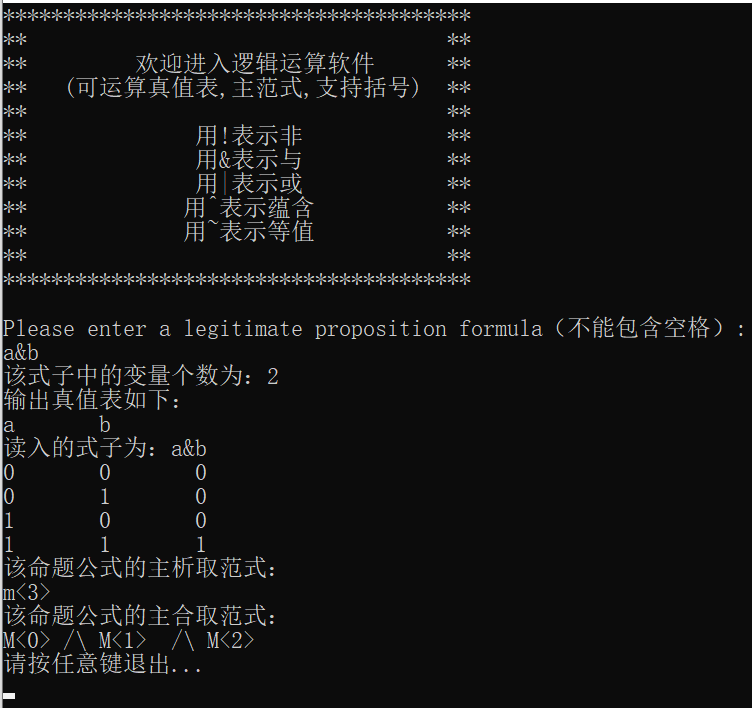
5.2 BC题目

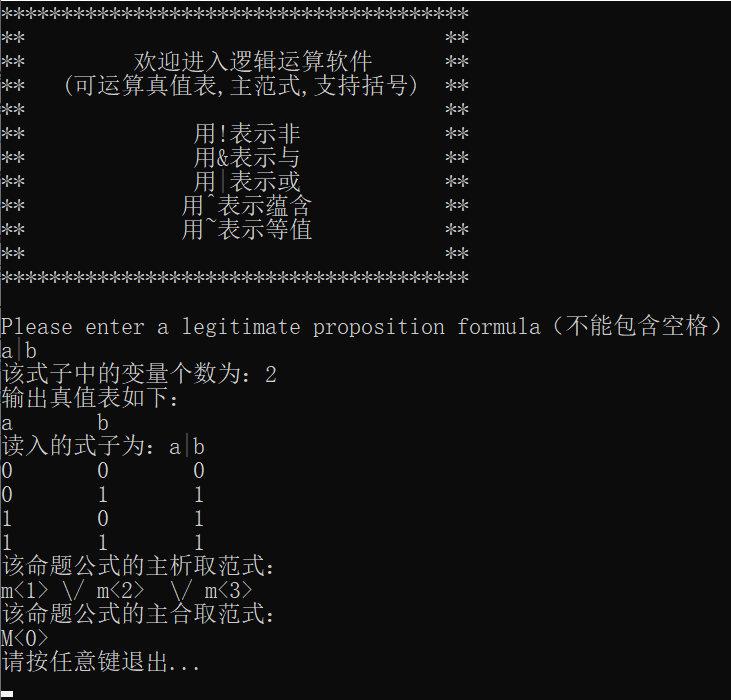
5.2.1 进入界面

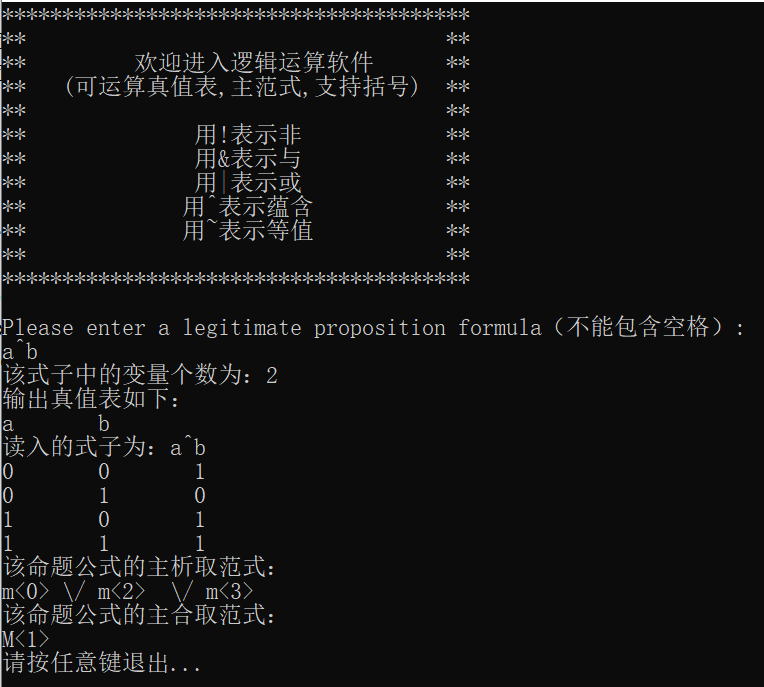


5.2.2 基本运算



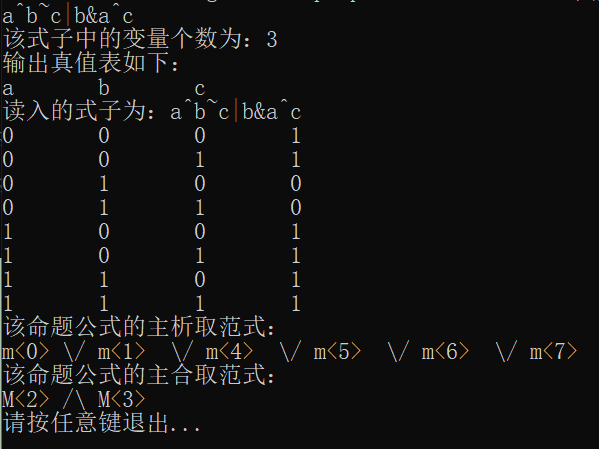




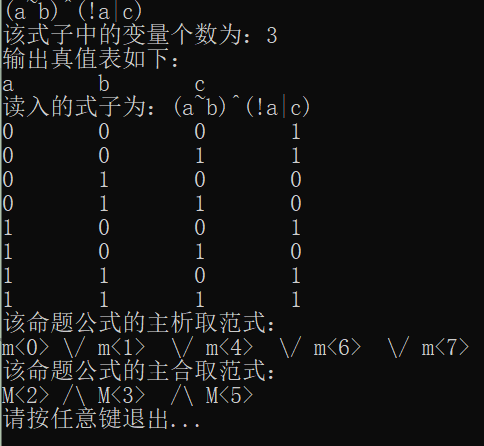




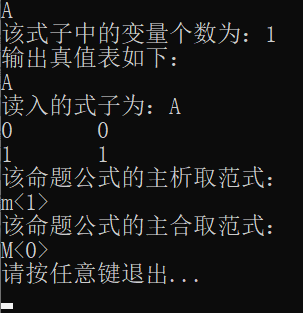
5.2.3 复杂的命题公式



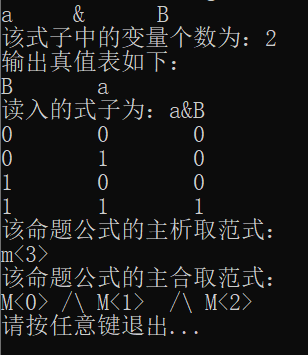
5.2.4 带括号的命题公式



5.2.5 仅有一个命题



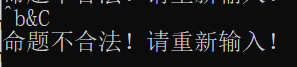
5.2.6 输入带有空格



5.2.7 括号不匹配



5.2.8 缺少操作数

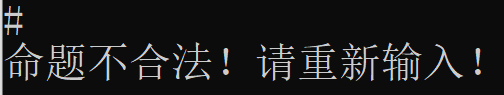
 

5.2.9 缺少操作符

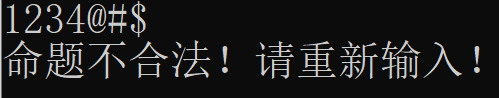


5.2.10 没有输入





5.2.11 非法字符



6 心得体会

通过本次题目的解答，更加丰富了我关于命题逻辑的知识，同时也加深了C++语言中布尔运算符和命题逻辑连接词之间的联系的认识。

其中对于第一题，我们需要对所输入的值进行错误判断，否则错误输入可能导致程序崩溃。这是我们无论在什么时候都需要注意的问题。其次是对于四种运算的在C++语言中的转换和实现，加深了我对命题逻辑连接词与计算机语言的联系。对于第二、三题，其中很重要的一部分就是对表达式合法性进行判断，在做题过程中感到比较有挑战性。我们在其中需要考虑很多中情况，包括但不限于：括号的匹配、一元运算符和二元运算符所匹配的操作数等等。同时，利用数据结构课程中所学习到的知识，利用栈的特性，对表达式进行求值也是其中的核心。其中需要注意的一点是对于括号的处理，我们需要优先计算括号中的内容。