**离散数学课程设计**

**项目说明文档**

**命题逻辑联接词、真值表、主范式**

作 者 姓 名： 苏家铭

学 号： 2151299

指 导 教 师： 唐剑锋

学院、 专业： 软件学院 软件工程

同济大学

**Tongji University**



**目录**

[1 项目分析 1](#_Toc26778)

[1.1 项目背景 1](#_Toc28937)

[1.2项目要求 1](#_Toc17013)

[1.2.1 功能要求 1](#_Toc23537)

[1.2.2 输入要求 1](#_Toc8750)

[1.2.3 输出要求 2](#_Toc22599)

[2 项目设计及实现 2](#_Toc1865)

[2.1 原理 2](#_Toc31313)

[2.2 项目算法 3](#_Toc1350)

[2.2.1 实现思路 3](#_Toc19371)

[2.3.2 代码实现 4](#_Toc18525)

[3 项目测试 15](#_Toc4504)

[3.1 一般测试（A题） 15](#_Toc11039)

[3.2 错误处理（A题） 15](#_Toc9559)

[3.3 非运算情况（B、C题） 16](#_Toc12029)

[3.4 与运算情况（B、C题） 16](#_Toc785)

[3.5 或运算情况（B、C题） 17](#_Toc26095)

[3.6 蕴含运算情况（B、C题） 17](#_Toc15938)

[3.7 等值运算情况（B、C题） 17](#_Toc30391)

[3.8 综合运算情况（B、C题） 18](#_Toc27093)

[3.9 错误情况处理（B、C题） 18](#_Toc22391)

[4 算法性能分析 19](#_Toc9297)

[4.1 正确性 19](#_Toc31860)

[4.2 可使用性 19](#_Toc30757)

[4.3 可读性 19](#_Toc30774)

[4.4 健壮性 19](#_Toc24645)

[5 实验小结 19](#_Toc27201)

# 1 项目分析

## 项目背景

日常生活中处处充满了逻辑，例如天下雨我就打伞，智商高的人学习数学不一定好等等语句都是逻辑的体现。于是我们称能判断真假的陈述句为命题。研究命题对于逻辑思维的提高，工作的高效性完成具有非常重要的作用。

## 1.2项目要求

### 1.2.1 功能要求

### A. 逻辑联接词的运算

本实验要求利用C＋＋语言，实现二元合取、析取、条件和双向条件表达式的计算。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。本题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。

## B、C. 求任意一个命题公式的真值表和主析取\合取范式

### 本实验要求利用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；可用一维数表示合式公式中所出现的n个命题变元，同时它也是一个二进制加法器的模拟器，每当在这个模拟器中产生一个二进制数时，就相当于给各个命题变元产生了一组真值指派。

### 在求出真值表后，能使真值表最右侧一栏的真值为真的命题变元真值指派可表示为主析取范式，而能使真值表的最右侧一栏的真值为假的命题变元真值指派可轻松地表示为主合取范式。

### 1.2.2 输入要求

A:命题变元p、q的真值

B:合法的逻辑表达式

### 1.2.3 **输出要求**

A:q、pd的二元合取、析取、条件和双向条件表达式的计算结果

B:该逻辑表达式的命题变元个数、真值表、主析取范式、主合取范式

# 2 **项目设计及实现**

## 2.1 **原理**

（1）**合取**：二元命题联结词。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。

（2）**析取**：二元命题联结词。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。

（3）**条件**：二元命题联结词。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。

（4）**双向条件**：二元命题联结词。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。

（5）**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结词决定，命题联结词的真值表给出了真假值的算法。 真值表是在逻辑中使用的一类数学表，用来确定一个表达式是否为真或有效。

（6）**主范式：**

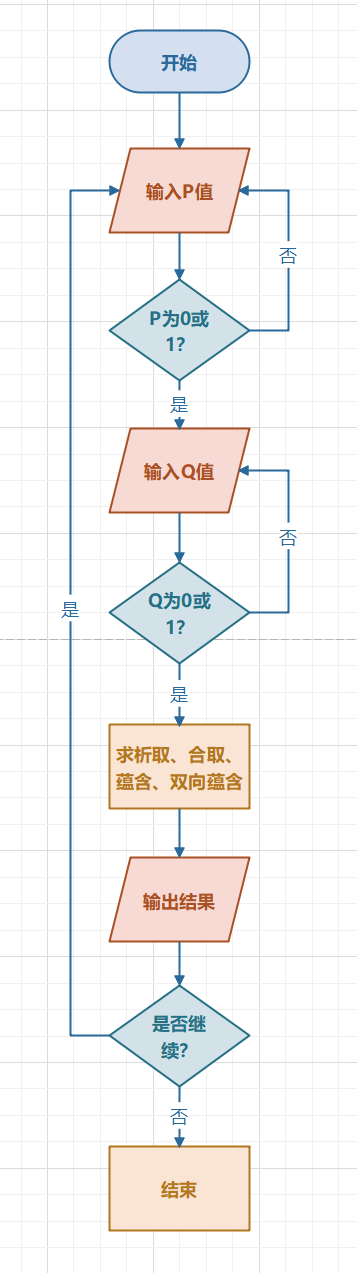
**主析取范式：**在含有n个命题变元的简单合取式中,若每个命题变元与其否定不同时存在,而两者之一出现一次且仅出现一次,则称该简单合取式为极小项。由若干个不同的极小项组成的析取式称为主析取范式;与A等价的主析取范式称为A的主析取范式。任意含n个命题变元的非永假命题公式A都存在与其等价的主析取范式,并且是惟一的。

**主合取范式：**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定不同时存在，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范式；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都存在与其等价的主合取范式，并且是惟一的。

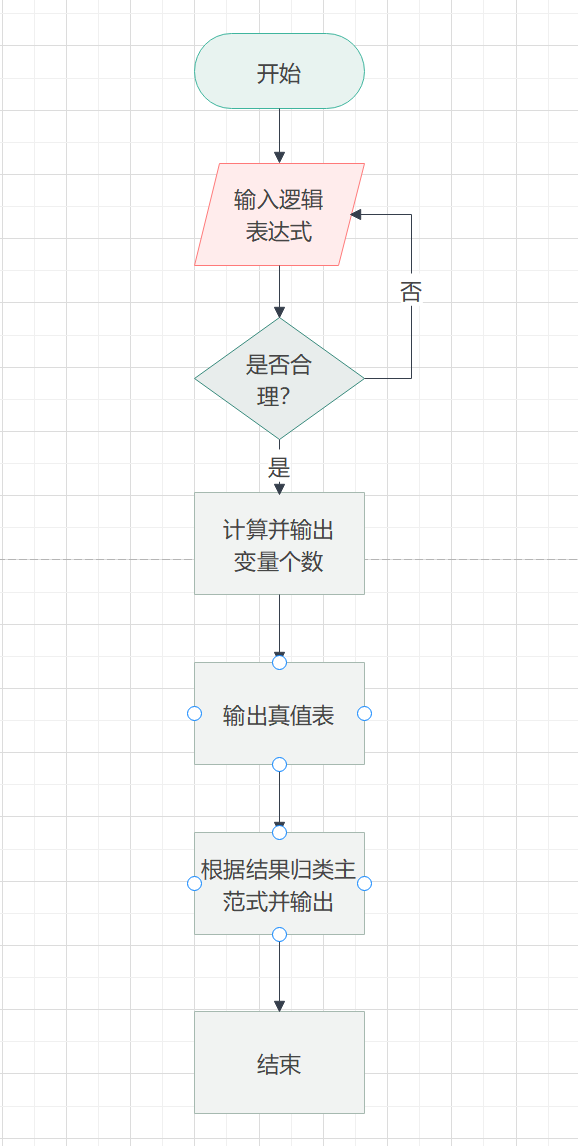
## 2.2 **项目算法**

### 2.2.1 实现思路

A题:首先是对各个输入量的处理，要确定输入的为0或1，否则则为出错，接下来就是运算处理，在C++中本身支持的有与、或、非这三种，可以用&&,||,!来表示，而在这个实验中，不是与、或、非的可以通过转化变为与、或、非的形式。



B，C题:首先输入一个式子，若式子不合法则再次输入，若合法则然后从式子中查找出变量的个数，开辟一个二进制函数，用来生成真值表，然后用函数运算，输出结果，并根据结果归类给范式，最后输出范式。



### 2.3.2 代**码实**现

A题：

#include <iostream>

#include <windows.h>

using namespace std;

int main()

{

int a[4];

int i = -1, j = -1;

char s;

tt:cout << "\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*" << endl; //标语

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

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

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

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

cout << "\n 请输入P的值（0或1）,以回车结束:";

cin >> i; //读取P的值

if (i != 0 && i != 1)

{

cout << "\n P的值输入有误,请重新输入!";

Sleep(2000);

system("cls"); //清屏

goto tt;

}

cout << "\n 请输入Q的值（0或1）,以回车结束:";

cin >> j; //读取Q的值

if (j != 0 && j != 1)

{

cout << "\n Q的值输入有误,请重新输入!";

Sleep(2000);

system("cls"); //清屏

goto tt;

}

a[0] = i && j; //与运算

a[1] = i || j; //或运算

a[2] = (!i) || j; //蕴含运算，将其转化为与或非形式

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

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

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

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

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

bb:cout << "\n是否继续运算?（y/n）"; //询问是否继续操作

cin >> s;

if (s == 'y' || s == 'n')

{

if (s == 'y')

{

system("cls"); //清屏

goto tt; //返回顶层

}

else

{

cout << "欢迎下次再次使用!" << endl; //退出

exit(0);

}

}

else

{

cout << "输入错误,请重新输入!" << endl; //错误校验

goto bb;

}

return 0;

}

B题：

#include <iostream>

#include <string>

#include <map>

#include <stack>

using namespace std;

typedef map<char, int> Map\_ci;

typedef map<int, char> Map\_ic;

typedef map<int, int> Map\_ii;

//typedef map<int, bool> Map\_ib;

Map\_ci priority;

Map\_ic getProposition(string formula);

int findProposition(Map\_ic, char p);

int pow2(int n);

Map\_ii toBinary(int n\_proposition, int index);

int calculate(string formula, Map\_ic pSet, Map\_ii value);

void check(stack<int>& value, stack<char>& opter);

int judge(string str);

int main()

{

priority['('] = 6;

priority[')'] = 6;

priority['!'] = 5;

priority['&'] = 4;

priority['|'] = 3;

priority['^'] = 2;

priority['~'] = 1;

priority['#'] = 0;

//运算符优先级

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: " << endl;

string formula;

tt: cin >> formula;

//1、（）括号匹配 2、a| a& 3、|a|b 4、#错误 5、ab&c报错

int T = judge(formula);

switch (T)

{

case 1:

cout << "括号不匹配！请再次输入：" << endl;

goto tt;

break;

case 2:

cout << "末尾不应出现连接词！请再次输入：" << endl;

goto tt;

break;

case 3:

cout << "开头不应出现双目连接词！请再次输入：" << endl;

goto tt;

break;

case 4:

cout << "含有非法输入字符！请再次输入：" << endl;

goto tt;

break;

case 5:

cout << "变量或连接词不应连续！请再次输入：" << endl;

goto tt;

break;

default:

break;

}

Map\_ic proposition\_set = getProposition(formula);

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

for (unsigned int i = 0; i < proposition\_set.size(); i++){

cout << proposition\_set[i] << "\t";

}

cout << formula << endl;

int\* m;

m = (int\*)malloc(sizeof(int) \* pow2(proposition\_set.size())); //该数组依次存放命题公式的各行(0或1)的运算结果的值

// int z = pow2(proposition\_set.size());

// cout << z << endl;

for (int i = 0; i < pow2(proposition\_set.size()); i++)

{

Map\_ii bina\_set = toBinary(proposition\_set.size(), i);

for (unsigned int j = 0; j < bina\_set.size(); j++)

{

cout << bina\_set[j] << "\t";

}

int result = calculate(formula, proposition\_set, bina\_set);

//m[i] = result;

\*(m + i) = result;

cout << result << endl;

}

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

cout << "该命题公式的主析取范式：" << endl;

for (int i = 0; i < pow2(proposition\_set.size()); i++)

{

if (\*(m + i) == 1)//主析取范式，值为真

{

if (n\_m == 0)//第一个范式

{

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

}

else

{

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

}

n\_m++;

}

}

if (n\_m == 0)//没有范式

{

cout << "0";

}

cout << endl;

cout << "该命题公式的主合取范式：" << endl;

for (int i = 0; i < pow2(proposition\_set.size()); i++)

{

if (\*(m + i) == 0)//主合取范式，值为假

{

if (n\_M == 0)//第一个范式

{

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

}

else

{

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

}

n\_M++;

}

}

if (n\_M == 0)//没有范式

{

cout << "0";

}

cout << endl;

return 0;

}

int findProposition(Map\_ic pSet, char p) //返回-1，表示该命题变项尚未被遍历过，可计数；否则说明该命题变项已被遍历过，则不重复计数。另外，还可以返回指定命题变项的下标

{

Map\_ic::iterator it = pSet.begin();

while (it != pSet.end())

{

if (it->second == p)

{

return it->first;

}

it++;

}

return -1;

}

Map\_ic getProposition(string formula) //该函数返回所输入公式中的命题变项(不包括运算符)

{

Map\_ic proposition;

int n\_proposition = 0;

for (unsigned int i = 0; i < formula.length(); i++)

{

char c = formula[i];

if ((c >= 'a' && c <= 'z') || (c >= 'A' && c <= 'Z'))

{

//遍历所有命题变项

int r = findProposition(proposition, c);

//cout << r << endl;

if (r == -1)

{

//说明该命题变项尚未被遍历过

proposition[n\_proposition] = c;

n\_proposition++;

}

}

else if (!priority.count(c))

{

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

exit(2);

}

}

return proposition;

}

Map\_ii toBinary(int n\_proposition, int index) //该函数返回命题变项的二进制(0或1)取值

{

Map\_ii result;

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

{

int r = index % 2;

result[n\_proposition - 1 - i] = r;

index = index / 2;

}

return result;

}

int pow2(int n) //该函数返回指定数字的二次方的值

{

if (n == 0)

return 1;

else

return 2 \* pow2(n - 1);

}

int calculate(string formula, Map\_ic pSet, Map\_ii value) //该函数返回给定，命题变项(值取0或1，可含括号)组合的运算结果

{

stack<char> opter;

stack<int> pvalue;

opter.push('#');

formula = formula + "#";

for (unsigned int i = 0; i < formula.length(); i++)

{

char c = formula[i];

if ((c >= 'a' && c <= 'z') || (c >= 'A' && c <= 'Z'))

{

pvalue.push(value[findProposition(pSet, c)]);

}

else

{

//此时遍历的是运算符

char tmp = opter.top();

if (priority[tmp] > priority[c])

{

while (priority[tmp] > priority[c] && tmp != '(')

{

check(pvalue, opter);

tmp = opter.top();

if (tmp == '#' && c == '#')

{

return pvalue.top();

}

}

opter.push(c);

}

else

opter.push(c);

}

}

return -1;

}

void check(stack<int>& value, stack<char>& opter) //该函数返回两个命题变项(取值0或1)的各种运算结果(0或1)

{

int p, q, result;

char opt = opter.top();

switch (opt)

{

case '&':

p = value.top();

value.pop();

q = value.top();

value.pop();

result = p && q;

value.push(result);

opter.pop();

break;

case '|':

p = value.top();

value.pop();

q = value.top();

value.pop();

result = p || q;

value.push(result);

opter.pop();

break;

case '!':

p = value.top();

value.pop();

result = !p;

value.push(result);

opter.pop();

break;

case '^':

q = value.top();

value.pop();

p = value.top();

value.pop();

result = !p || q;

value.push(result);

opter.pop();

break;

case '~':

p = value.top();

value.pop();

q = value.top();

value.pop();

result = (!p || q) && (p || !q);

value.push(result);

opter.pop();

break;

case '#':

break;

case '(':

break;

case ')':

opter.pop();

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

{

check(value, opter);

}

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

{

opter.pop();

}

break;

default:

break;

}

}

//判断错误输入

int judge(string str) {

stack<char> st;

size\_t i = 0,len=str.length();

//检查括号是否匹配

while (i < len) {

if (str[i] == '(')//左括号

st.push(str[i]);

else if (str[i] == ')' && !st.empty())//右括号并且栈内有左括号

st.pop();

else if(str[i]==')')//栈内没左括号但有右括号

return 1;

i++;

}

if (!st.empty())

return 1;

//括号检查完毕

i = 0;

//检查a| a&错误

if (str[len - 1] == '!' || str[len - 1] == '|' ||

str[len - 1] == '&' || str[len - 1] == '~' ||

str[len - 1] == '^')

return 2;

//结尾错误检查完毕

//检查|a|b错误

if (str[0] == '|' ||str[0] == '&' || str[0] == '~' ||str[0] == '^')

return 3;

//检查开头完毕

//检查非法符号

while (i < len) {

if (str[i] == '(' || str[i] == ')' ||

(str[i] >= 'a' && str[i] <= 'z') ||

(str[i] >= 'A' && str[i] <= 'Z') ||

str[i] == '!' || str[i] == '|' ||

str[i] == '&' || str[i] == '^' ||

str[i] == '~')

i++;

else

return 4;

}

//检查非法字符完毕

//检查ab&c等相同合法字符连续的情况

i = 0;

while (i < len) {

if ((str[i] >= 'a' && str[i] <= 'z') ||

(str[i] >= 'A' && str[i] <= 'Z') )

if ((str[i + 1] >= 'a' && str[i + 1] <= 'z') ||

(str[i + 1] >= 'A' && str[i + 1] <= 'Z'))

return 5;

if (str[i] == '!' || str[i] == '|' ||

str[i] == '&' || str[i] == '^' ||

str[i] == '~')

if (str[i + 1] == '!' || str[i + 1] == '|' ||

str[i + 1] == '&' || str[i + 1] == '^' ||

str[i + 1] == '~')

return 5;

i++;

}

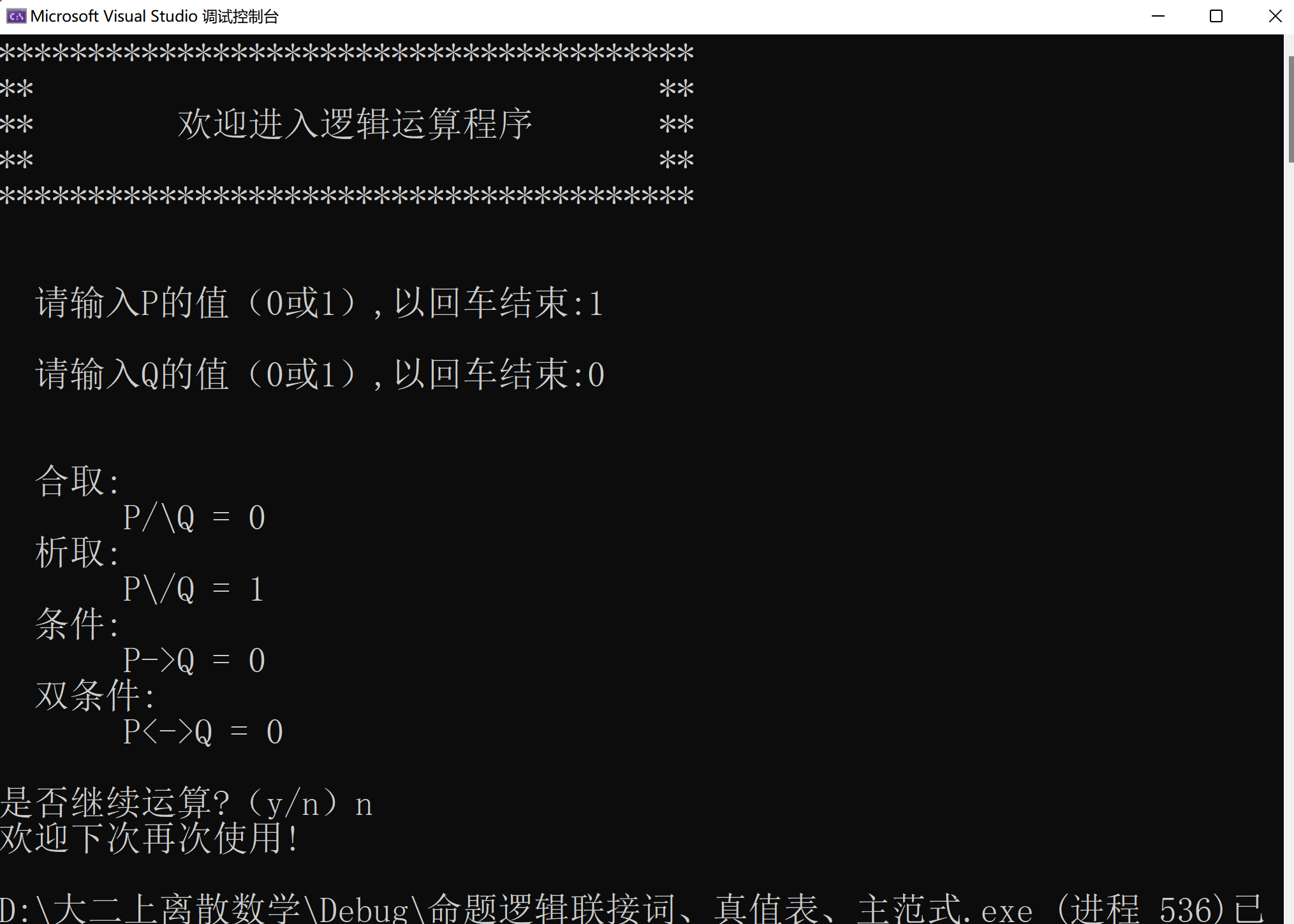
//连续合法检查完毕

return 0;

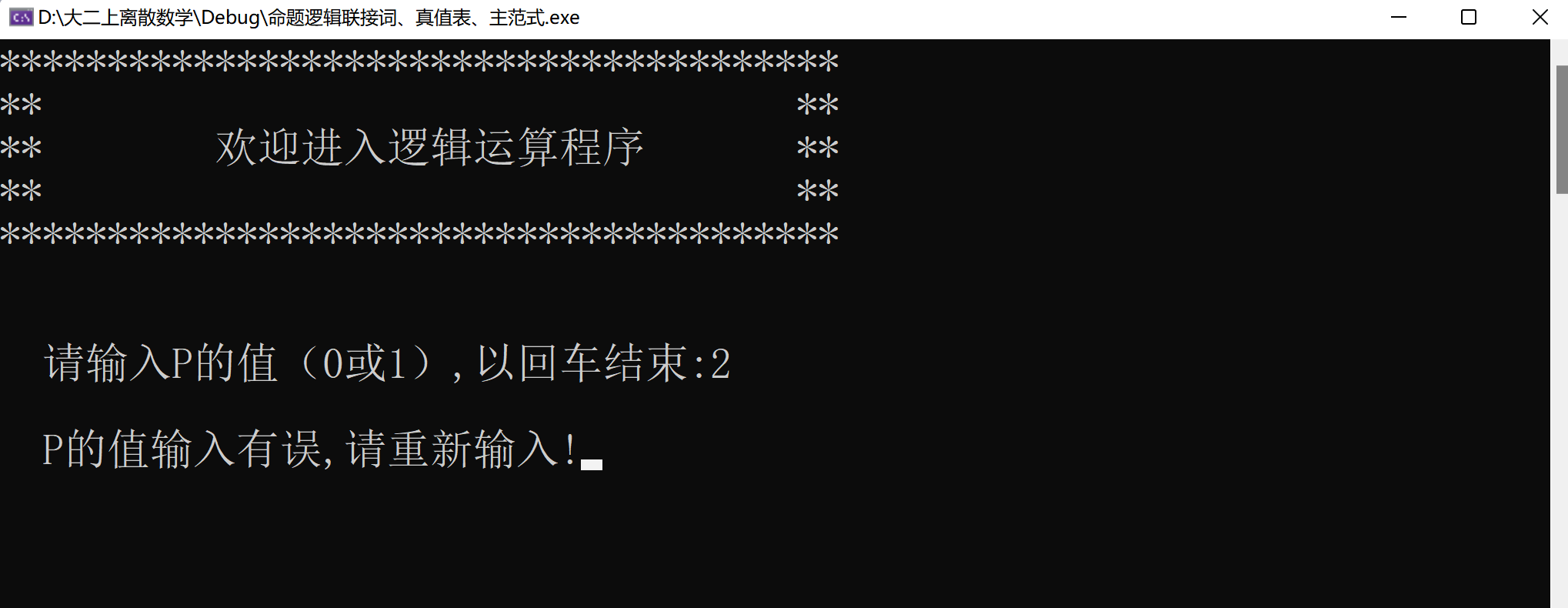
}

# 3 项目测试

## 3.1 一般测试（A题）



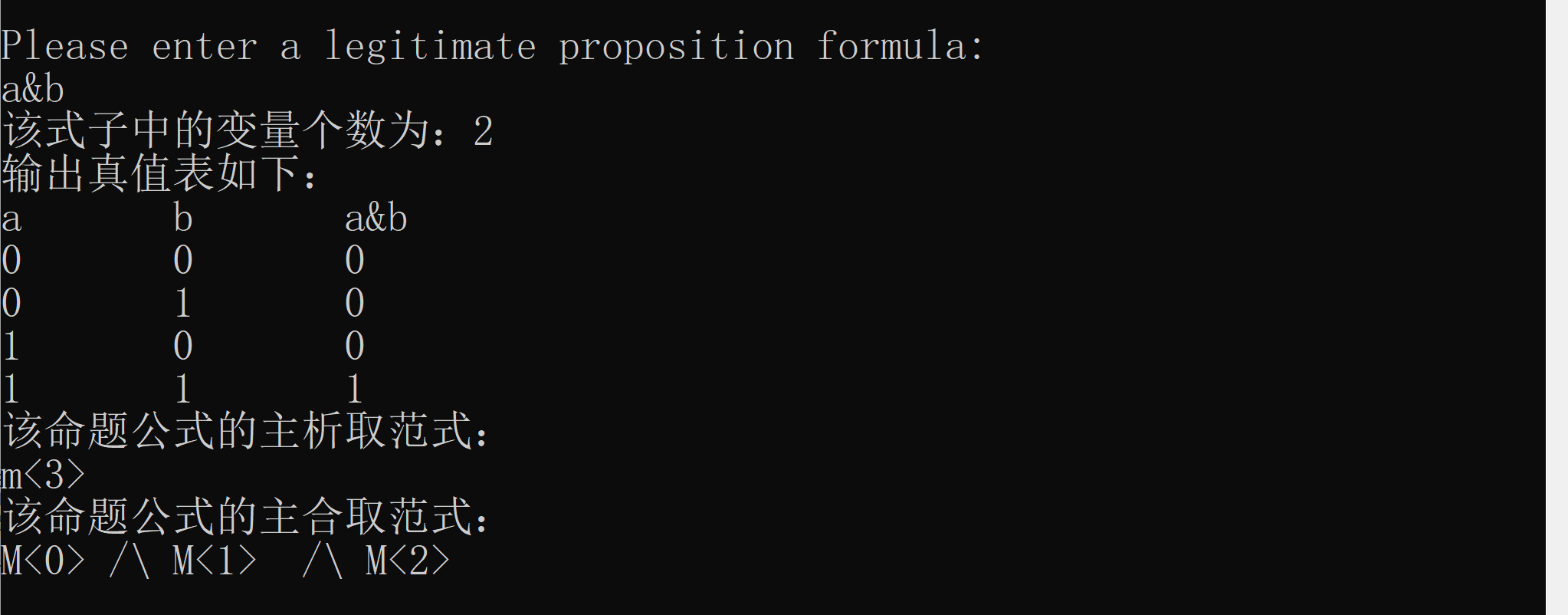
## 3.2 错误处理（A题）



## 3.3 非运算情况（B、C题）



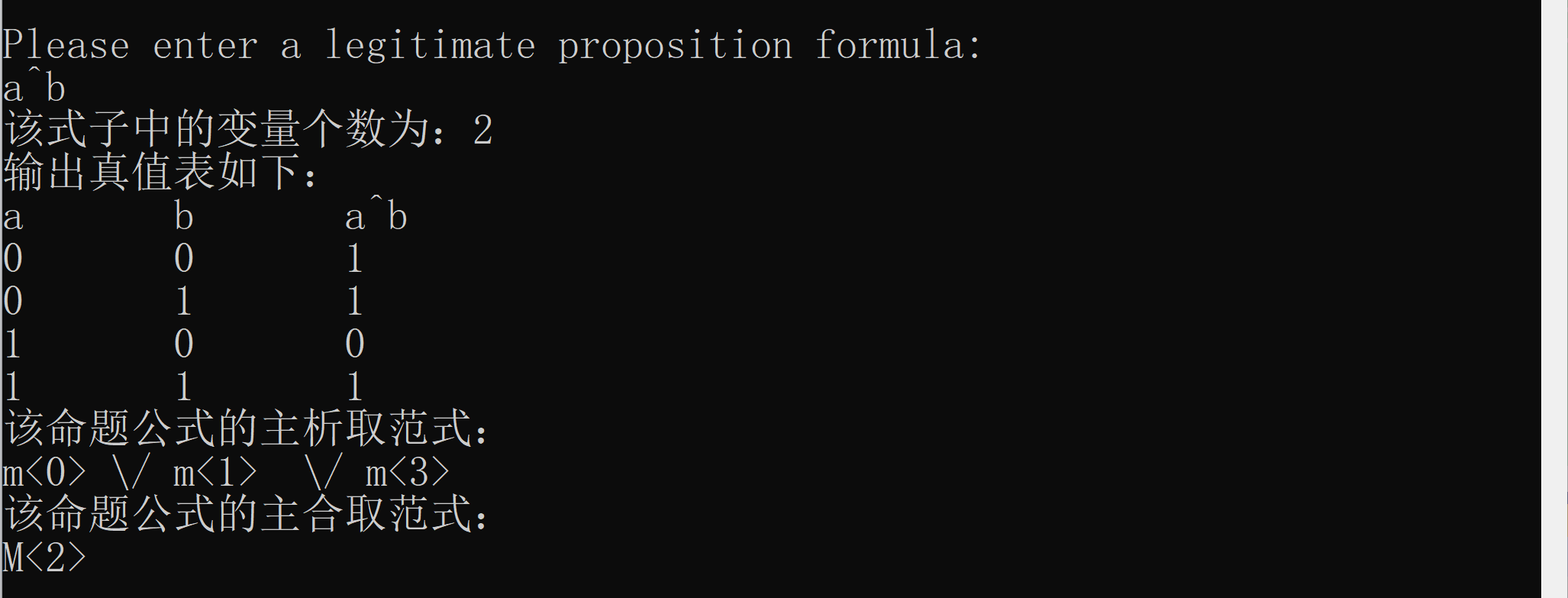
## 3.4 与运算情况（B、C题）



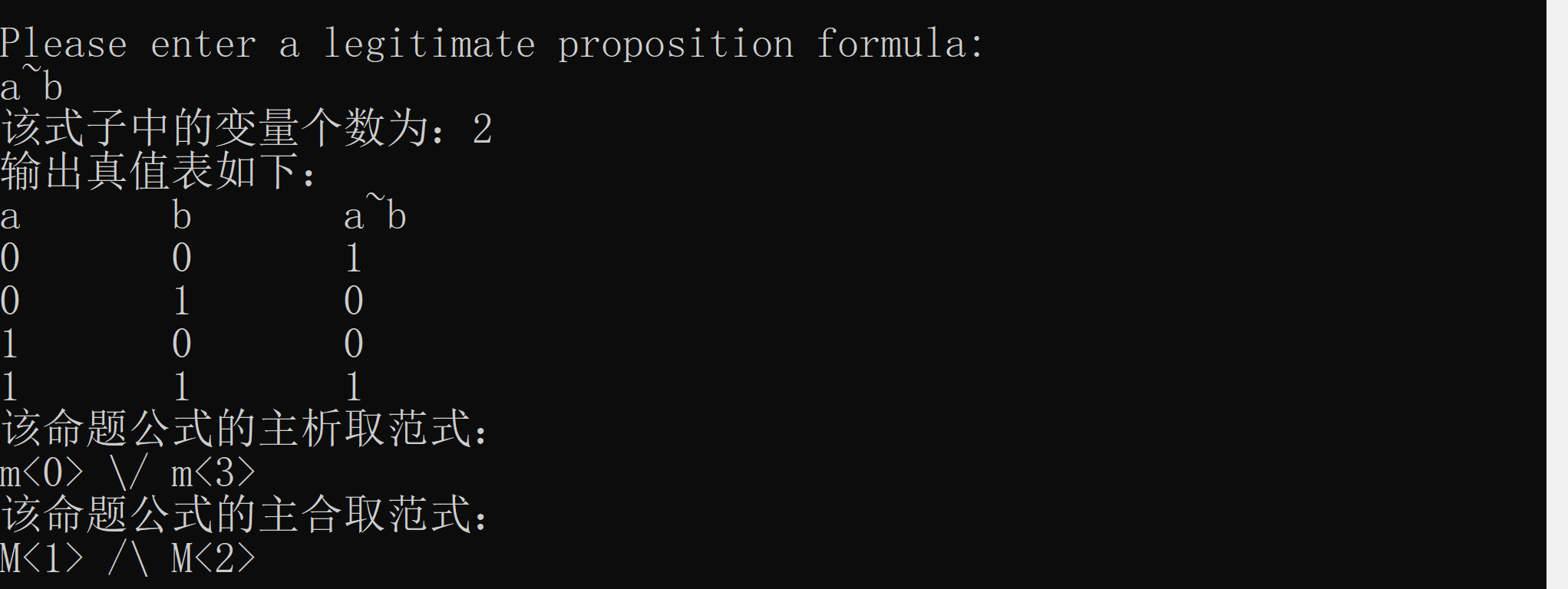
## 3.5 或运算情况（B、C题）

## 

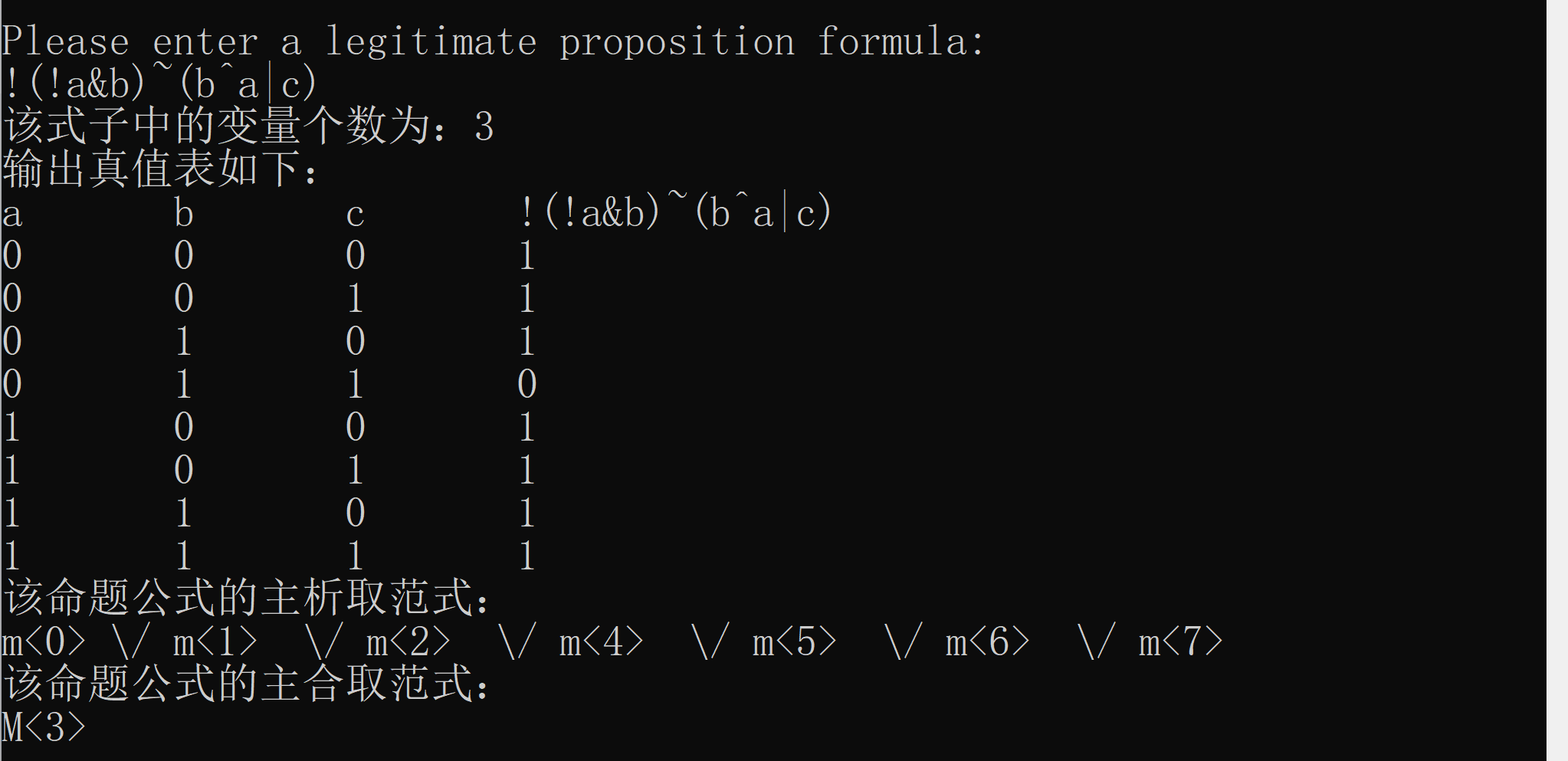
## 3.6 蕴含运算情况（B、C题）



## 3.7 等值运算情况（B、C题）



## 3.8 综合运算情况（B、C题）



## 3.9 错误情况处理（B、C题）



# 4 算法性能分析

## 4.1 正确性

本算法能正确地执行预定的功能和性能要求，由以上版块证明本项目与实际情况比较一致，满足正确性。

## 4.2 可使用性

本算法可以很方便地使用，项目封装了基本的函数。并且该算法有良好的界面和完备的用户文档。没有使用公用变量。

## 4.3 可读性

本算法逻辑清晰、简单、且结构化，所有命名与函数名都具有实际含义，让人见名知义。且算法中包含了大量注释，简要说明了算法功能、输入与输出参数的使用规则、重要数据的作用、算法中各程序段完成的功能。

## 4.4 健壮性

本算法对于边界条件，诸如A题中：P或Q值不为布尔代数，B、C题中：逻辑表达式括号不匹配、开头结尾出现连接词、含有非法字符、变量或连接词连续的情况都有相应的错误提示和相应的弥补措施。

# 5 实验小结

本次实验可以说是十分具有综合性的实验，虽然实验程序以及给出，但是在理解代码的过程中我受益颇多。老师在课上已经花了两节课的时间去一行一行调试代码，把每一步的原理说明给大家，讲解地十分细致，本人在课上也认真听讲，把所有应该记录下来的要点都理解清楚，这样也节省了课后再看代码、理解代码的时间。

为了增加代码的健壮性，我还基于5类错误输入实现了错误判断以及处理，其中包括：括号是否匹配、末尾是否出现连接词，开头是否出现双目连接词、是否有分非法字符、是否有连续的逻辑变量。针对各类错误处理，我实现了一个void类型的judge()函数，封装性更优。

这次项目的实现让我明白了离散数学的与、或、非等联接词和逻辑运算符之间有这般相似，并且在列出真值表的每一个值时也可以利用程序的循环、函数的计算等功能轻松实现。

虽然很多问题在离散数学课堂上可以利用纸笔轻松的求解，但是作为软件工程的学生，我们的所学知识终究要回归到应用上，这次的项目实现使得离散数学第一章的最基本的逻辑运算知识映射到了程序的设计，让我们在学习知识的同时巩固了代码能力，能做到一举多得的效果。