离散数学课程实验说明文档

——命题逻辑联接词、真值表、主范式

作 者 姓 名： 马威

学 号： 2151294

指 导 教 师： 唐剑锋

学院、 专业： 软件学院 软件工程

****

目 录

[1 项目分析 1](#_Toc495668153)

[1.1 项目背景 1](#_Toc495668154)

[1.2 项目要求 1](#_Toc495668155)

[2 项目设计 2](#_Toc495668156)

[2.1 数据结构设计 2](#_Toc495668157)

2.2 类设计 3

2.2.1 字符串类（String） 3

2.2.2 结点类（StackNode） 5

2.2.3 链式栈类（LinkedStack） 5

2.2.4 地图类（Map） 6

2.3 算法设计 6

2.3.1 算法思路 6

2.3.2 性能评估 7

2.3.3 联接词部分 7

·流程图表示 7

·代码实现 8

2.3.4 求真值表与主范式部分 10

·判断表达式是否合法代码实现 10

·一趟计算结果代码实现 11

·主过程代码实现 12

[3 项目测试 1](#_Toc495668161)5

[3.1 第一题：一般情况 1](#_Toc495668157)5

[3.2 第一题：错误输入处理 1](#_Toc495668157)6

[3.3 第二题：一般情况 1](#_Toc495668157)7

3.4 第二题：错误输入处理 18

4 心得体会 18

**1.项目分析**

1.1 项目背景

在命题逻辑中，对于联接词的理解和公式的真值计算是非常重要的，显然，对于两个命题p和q，我们关心它们在特定取值下，使用各类联接词连接起来公式的真值。同时，对于更加复杂的公式，我们关心它的成真赋值、成假赋值，自然也关心它的主析取范式和主合取范式。

1.2 项目要求

1. 逻辑联接词的运算

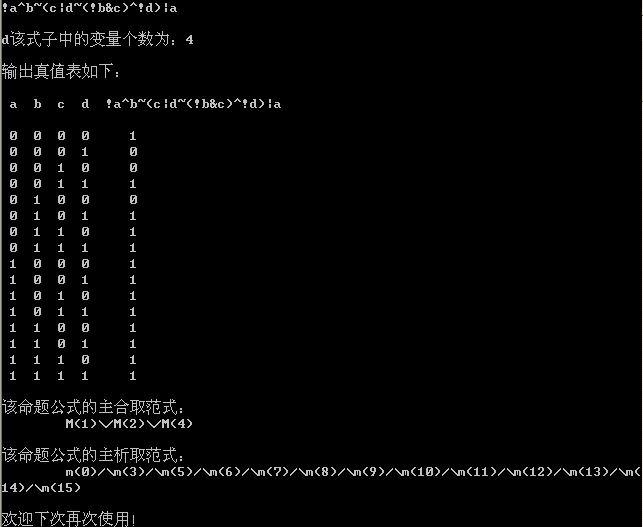
要求实现二元合取、析取、条件和双向条件表达式的计算。

2. 求任意一个命题公式的真值表和主析取范式

本实验要求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。同时在求得真值表的基础上，输出公式的主析取范式、主合取范式。

项目示例：





**2.项目设计**

2.1 数据结构设计

由项目分析可以得出，第一题无需额外的数据结构。第二题中涉及到命题变项的统计，因此需要一个能表示下标与字符（命题变项）对应的结构。同时由于运算符的优先级差异，需要将操作数和运算符暂存到栈中，故需要类似栈的数据结构。命题公式可以用字符串进行存储。（以上提到的都用stl库实现）

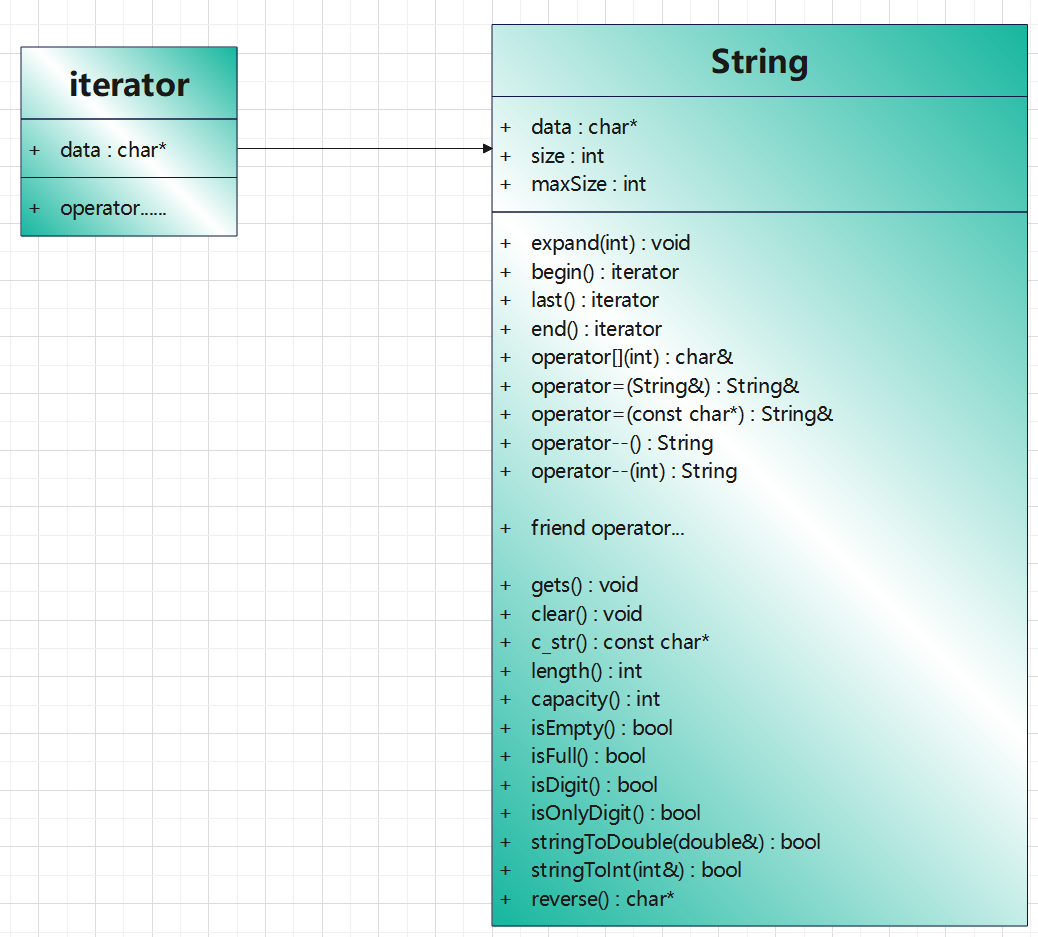
2.2 类设计

**2.2.1 字符串类（String）**

字符串本质上是通过字符数组实现的，通过动态开辟一个数组存放各个字符。由于字符串的添加、比较、赋值等操作使用得十分频繁，故该类重载了许多运算符的操作，使得这些操作使用起来更易上手。

另一种常用的字符串操作是对其的遍历，在此添加了迭代器类（iterator）和一些运算符重载，以便对字符串的遍历等操作。由于这个类名在标准库中已有实现，且此处的迭代器类为String类特有，故使用了嵌套类的形式，使得iterator只能在String类内访问。

该类和其内部的iterator类的UML图如下所示：



其中，主要函数如下：

//增加数组空间大小

void expand(const int \_timesOfExpandingDefaultSize);

//返回字符串的起始位置

inline String::iterator begin();

inline const String::iterator begin()const;

//返回字符串末尾的后一个位置

inline String::iterator end();

inline const String::iterator end()const;

//返回字符串的末尾位置

inline String::iterator last();

inline const String::iterator last()const;

//重载函数：下标访问

char& operator[](const int pos)const;

//重载函数：复制（从String对象复制）

String& operator=(const String& str);

//重载函数：复制（从const char\*变量复制）

String& operator=(const char\* str);

//输入字符串（支持空格输入）

void gets();

//将字符串置为空

void clear();

//返回字符串首地址（以const char\*形式返回）

const char\* c\_str()const;

//返回字符串长度

int length()const;

//返回字符串最大容量

int capacity()const;

//判断字符串是否为空

bool isEmpty()const;

//判断字符串是否已满

bool isFull()const;

//判断字符串是否表示一个数值

bool isDigit()const;

//判断字符串是否仅有数字

bool isOnlyDigit()const;

//将字符串转成双精度数

bool stringToDouble(double& d)const;

//将字符串转成整型数

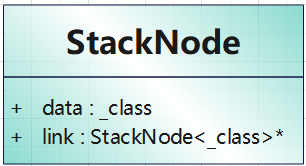
bool stringToInt(int& i)const;

//将字符串反转

char\* reverse();

**2.2.2 结点类（StackNode）**

链表结点存储了结点数据、后继节点位置，其UML图如下：

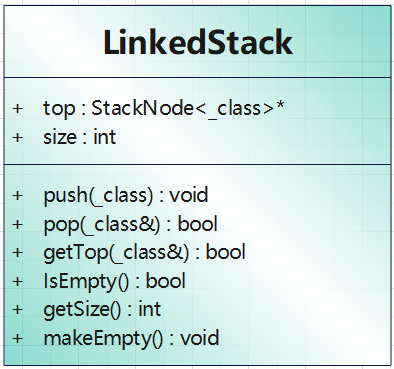


**2.2.3 链式栈类（LinkedStack）**

链式栈类本质上仍是单链表，保存了链表的基本结构。由于我们在使用栈时仅仅利用其“先进后出”的特性，所以其提供的操作并不像一般链表操作那样齐全，而是仅仅有入栈和出栈。

由于对栈的插入、删除元素等操作的位置比较固定，为了节省空间，并未采用附加头结点。

其UML图如下：



其中，主要函数如下：

//进栈

void push(const \_class x);

//出栈

bool pop(\_class& x);

//取栈顶元素

bool getTop(\_class& x)const;

//判栈空否

bool IsEmpty()const;

//返回栈元素个数

int getSize()const;

//清空栈的内容

void makeEmpty();

**2.2.4 地图类（map）**

在map类中，两个量是一一对应的（这两个量的类型可以不同），在进行各类的统计方面使用的很多。内部结构为红黑树，同时通过重载了许多运算符，添加了游标类使得对其的遍历等操作变得更简单。

由于实现较为复杂，故仅在此说明map类的性质，未画出UML图并说明。

2.3 算法设计

**2.3.1 算法思路**

第一题在正确输入两个变量的情况下，进行四种运算即可。

第二题可依据以下步骤求解：

（1）判断表达式是否合法，若不合法，输出不合法原因并结束。

（2）第k次循环中将k转化为n位二进制数，0为真1为假进行记录。此时可以打印真值表取值部分。

（3）遍历整个表达式，若读到操作数则直接取对应的真值后压入操作数栈，若读到运算符则根据栈顶运算符的优先级进行对应操作。读到运算符优先级高，则压入栈中；栈顶运算符优先级高，则其弹出。遍历完成且栈空时结果得到，此时可以打印真值表结果部分。

（4）由于所有的命题取值有2**n**种，（2）需进行2**n**次循环。真值表打印完成。

（5）统计所有情况的结果，为真则归到主析取范式，为假则归到主合取范式。统计完成后把两主范式打印出来。

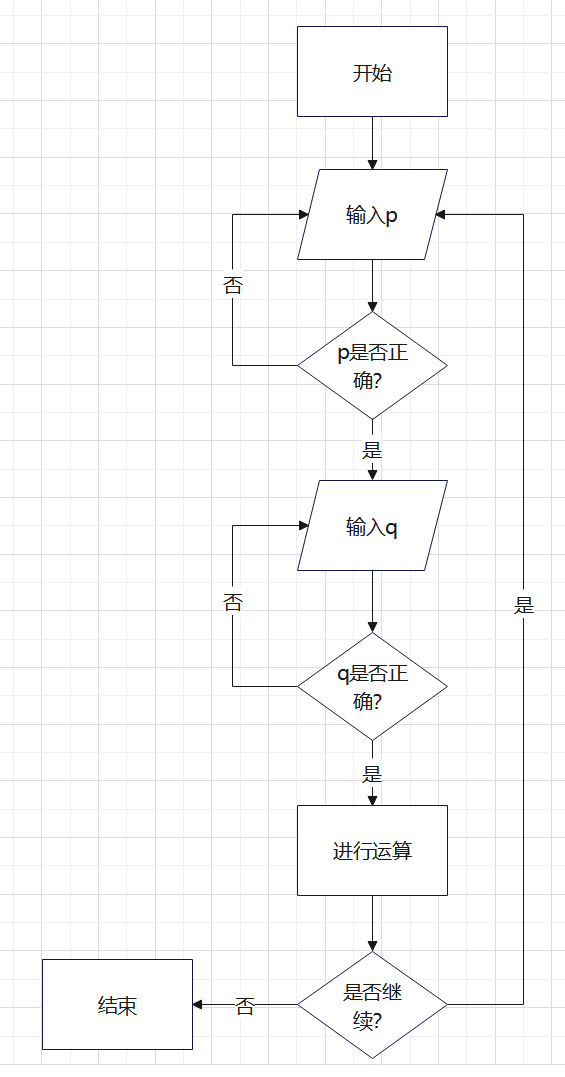
**2.3.2 性能评估**

第一题由于变量没有数量上的变化，时间固定为O(1)。

第二题较为复杂，因为并不了解map的访问等操作的内部实现，设map的各类操作时间为O(1)，同时设有k个命题变项，表达式长度为n，由于所有的命题取值有2**k**种，因此（2）需进行2**k**次循环，每次需遍历一遍表达式，需要O(n)，故总的时间复杂度为O(n\*2**k**)。

**2.3.3 联接词部分**

·流程图表示



·代码实现

int main()

{

    bool results[4];

    bool p, q;

    while (1) {

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

            << "\*\*                                   \*\*\n"

            << "\*\*        欢迎进入逻辑运算程序       \*\*\n"

            << "\*\*                                   \*\*\n"

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

        while (1) {  //正确输入p的值

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

            int num;

            cin >> num;

            if (cin.good() && (num == 0 || num == 1)) {

                p = bool(num);

                break;

            }

            cin.clear();

            cin.ignore(INT\_MAX, '\n');

            cout << "  P的值输入有误，请重新输入" << endl;

        }

        while (1) {  //正确输入q的值

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

            int num;

            cin >> num;

            if (cin.good() && (num == 0 || num == 1)) {

                q = bool(num);

                break;

            }

            cin.clear();

            cin.ignore(INT\_MAX, '\n');

            cout << "  Q的值输入有误，请重新输入" << endl;

        }

        results[0] = p && q;  //与运算

        results[1] = p || q;  //或运算

        results[2] = (!p) || q;  //蕴含运算，将其转化为与或非形式

        results[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 = " << results[0] << endl  //输出结果

            << "  析取:\n       P\\/Q = " << results[1] << endl

            << "  条件:\n       P->Q = " << results[2] << endl

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

            << "\n是否继续运算?（y/n）";

        char selection;

        while (1) {

            selection = \_getch();

            if (selection == 'y' || selection == 'n') {

                cout << selection << endl;

                break;

            }

        }

        if (selection == 'y')

            system("cls");

        else {

            cout << "欢迎下次使用！" << endl;

            break;

        }

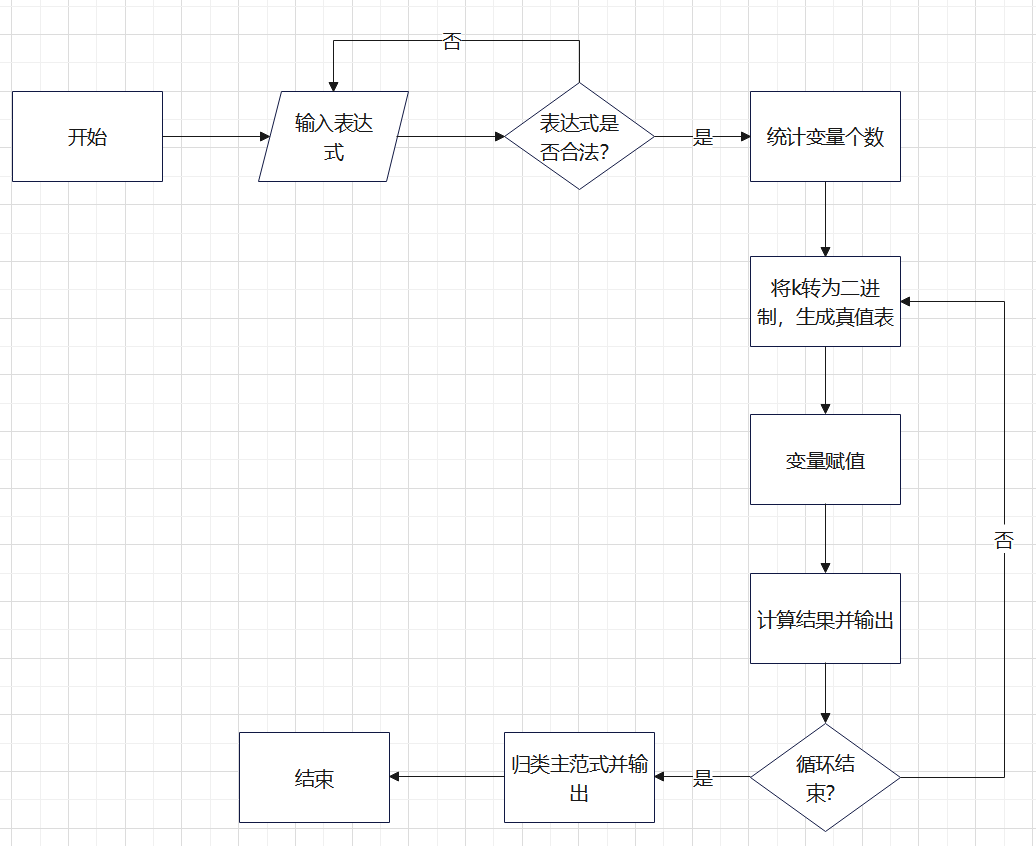
    }

    return 0;

}

**2.3.4 求真值表与主范式部分**

·流程图表示



·判断表达式是否合法代码实现

bool formulaCheck(const string& str)

{

    string::const\_iterator it = str.begin();  /\*游标对象\*/

    stack<char> brackets;  /\*记录括号情况的栈\*/

    int op1Num = 0, op2Num = 0, numNum = 0;  /\*操作符（单目、双目）个数，数字个数\*/

    while (it != str.end()) {

        /\*1、若扫描到字符表示命题，则进行计数\*/

        if (((\*it) >= 'a' && (\*it) <= 'z') || ((\*it) >= 'A' && (\*it) <= 'Z'))

            numNum++;  /\*一个数字扫描完成，进行计数\*/

        if (\*it == '(')  /\*若遇到左括号，压入栈中\*/

            brackets.push(\*it);

        else if (\*it == ')') {  /\*若遇到右括号，栈不为空则弹出一个元素；栈为空则说明右括号多余，非法\*/

            if (!brackets.empty())

                brackets.pop();

            else {

                cout << "括号不匹配" << endl;

                return false;

            }

        }

        else if (isOp(\*it)) {  /\*若遇到操作符，进行计数\*/

            if (\*it == '!')

                op1Num++;

            else

                op2Num++;

        }

        if (it != str.end())

            it++;

        else

            break;

    }

    if (!brackets.empty()) {  /\*若扫描完成栈仍不为空，则左括号多余，非法\*/

        cout << "括号不匹配" << endl;

        return false;

    }

    if (numNum != op2Num + 1) {  /\*若双目运算符个数不等于数字个数+1，则不匹配，非法\*/

        cout << "操作符与操作数数量不匹配" << endl;

        return false;

    }

    return true;  /\*所有标准均符合，则表达式合法\*/

}

·一趟计算结果代码实现

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

{

    stack<char> opter;  //存运算符的栈

    stack<int> pvalue;  //存操作数的栈

    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)]);  //将某个命题变项的取值找到，放入栈中

        }

        else

        {

            //此时遍历的是运算符

            char tmp = opter.top();  //取栈顶运算符

            if (priority[tmp] > priority[c])  //若栈顶优先级高，则运算到栈顶优先级低于c或者有括号出现

            {

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

                {

                    check(pvalue, opter);

                    tmp = opter.top();

                    if (tmp == '#' && c == '#')  //若遇到#，则说明运算完成，返回结果

                    {

                        return pvalue.top();

                    }

                }

                opter.push(c);  //栈顶优先级低于c，c进栈

            }

            else

                opter.push(c);  //栈顶优先级低于c，c进栈

        }

    }

    return -1;

}

·主过程代码实现

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;

    cin >> formula;  //输入公式

    if (!formulaCheck(formula))  //检查公式是否合法

        return 0;

    Map\_ic proposition\_set = getProposition(formula);  //统计命题变项

    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";  //输出所有的命题变项

    }

    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

    for (int i = 0; i < pow2(proposition\_set.size()); i++)  //有2^n种取值，遍历一遍

    {

        Map\_ii bina\_set = toBinary(proposition\_set.size(), i);  //将取值转换为二进制

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

        {

            cout << bina\_set[j] << "\t";  //将各命题变项的值打印出来

        }

        int result = calculate(formula, proposition\_set, bina\_set);  //计算公式真值

        \*(m+i) = result;  //记录这一趟的真值

        cout << result << endl;  //打印这一趟的计算结果

    }

    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++)  //统计成真赋值并打印主析取范式

    {

        if (\*(m + i) == 1)

        {

            if (n\_m == 0)

            {

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

            }

            else

            {

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

            }

            n\_m++;

        }

    }

    if (n\_m == 0)    //无成真赋值情况

    {

        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++)   //统计成假赋值并打印主合取范式

    {

        if (\*(m + i) == 0)

        {

            if (n\_M == 0)

            {

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

            }

            else

            {

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

            }

            n\_M++;

        }

    }

    if (n\_M == 0)  //无成假赋值情况

    {

        cout << "0";

    }

    cout << endl;

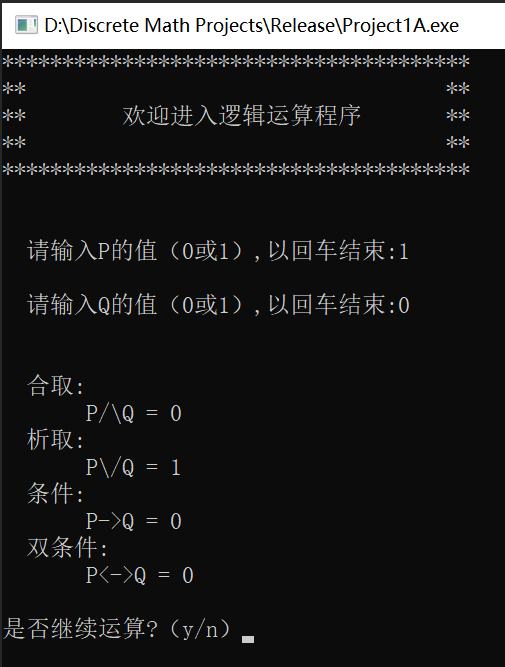
    return 0;

}

**3.项目测试**

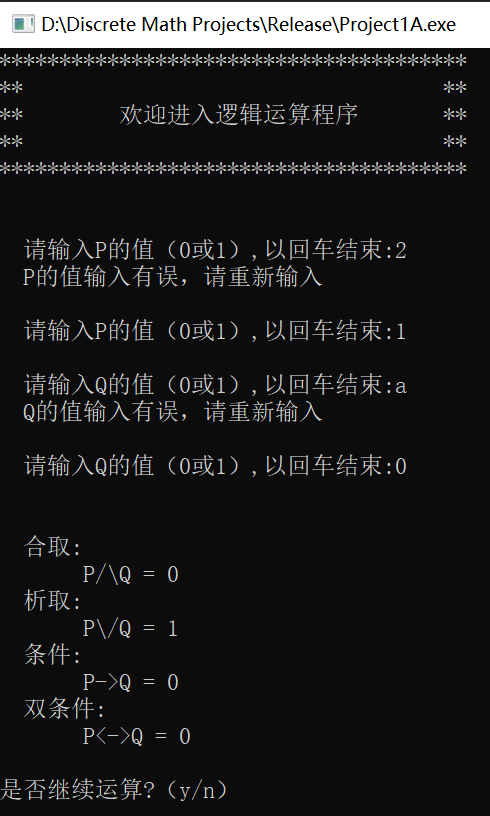
3.1 第一题：一般情况

测试结果：



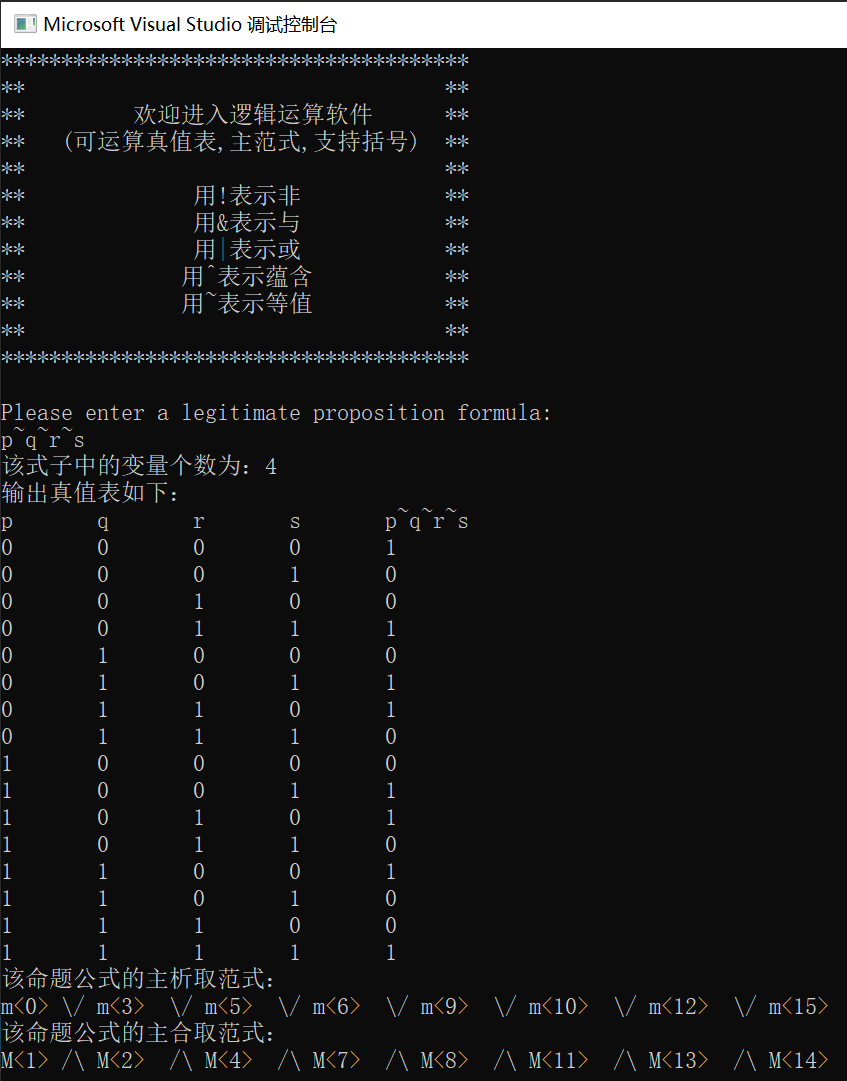
3.2 第一题：错误输入处理

测试结果：



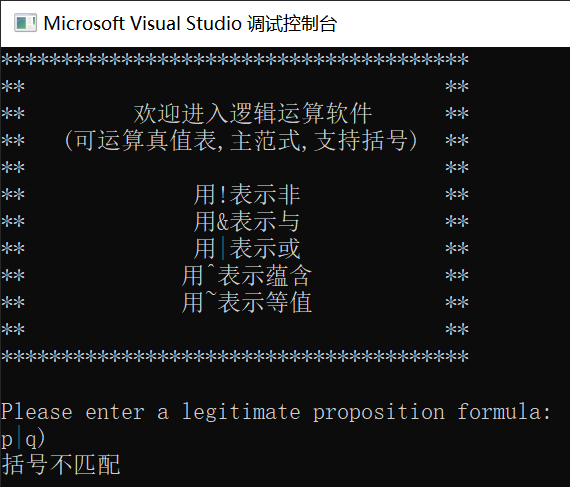
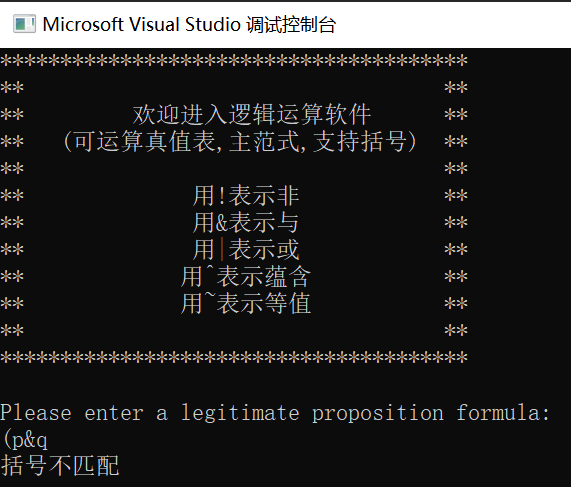
3.3 第二题：一般情况

测试结果：



3.4 第二题：错误输入处理

测试结果：





**4.心得体会**

在完成第二题后，我对于命题公式中不同的运算符的优先级有了更深刻的了解，同时对于成真赋值、成假赋值和主范式有了更为准确地理解。

第二题的编码过程中，在如何用一个结构表示下标与命题变项之间的关系上我犯了难，因为除map外的其余数据结构很难直接用于统计，基本上都是时间复杂度太高。而map的出现解决了这一问题，在这次作业后我对map的性质和用法也学习到了不少，或许在今后的编程中能有所帮助。