同济大学

《离散数学课程项目文档》

——命题逻辑联接词、真值表、主范式

作 者 姓 名：胡峻玮

学　　号：2153393

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

1.1 项目背景

在命题逻辑中，对于联接词的使用和公式的真值计算是非常重要的，简单的，我们关注对于两个命题 p 和 q，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，对于变量更多、逻辑联接词更多的复杂公式，我们关注其成真赋值、成假赋值、主析取范式和主合取范式，因此通过本项目实现上述两项功能。

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；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。

项目示例：

```
!a^b~(c|d~(!b&c)^!d)|a

d该式子中的变量个数为：4

输出真值表如下：

 a  b  c  d  !a^b~(c|d~(!b&c)^!d)|a

 0  0  0  0     1
 0  0  0  1     0
 0  0  1  0     0
 0  0  1  1     1
 0  1  0  0     0
 0  1  0  1     1
 0  1  1  0     1
 0  1  1  1     1
 1  0  0  0     1
 1  0  0  1     1
 1  0  1  0     1
 1  0  1  1     1
 1  1  0  0     1
 1  1  0  1     1
 1  1  1  0     1
 1  1  1  1     1

该命题公式的主合取范式：
        M(1)∨M(2)∨M(4)

该命题公式的主析取范式：
        m(0)∨m(3)∨m(5)∨m(6)∨m(7)∨m(8)∨m(9)∨m(10)∨m(11)∨m(12)∨m(13)∨m(
14)∨m(15)

欢迎下次再次使用！
```

3. 项目环境

采用 C 或 C＋＋编程语言，MS Visual Studio 2019 实验环境实现。

2.项目设计

2.1 数据结构设计

通过对于题目的分析，A 题无需额外的数据结构。B C 题中涉及到命题变项的统计，因此需要一个能表示下标与命题变项对应的结构。同时由于涉及操作数和运算符的优先级问题，因此需要开辟两个类似栈的数据结构，将操作数和运算符暂存到栈中。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）

字符串是基于字符数组的数据结构，它使用动态分配的数组来存储字符序列。由于字符串常常需要进行添加、比较、赋值等操作，因此该类实现了许多运算符的重载，以便于更方便地进行这些操作。

另一个常见的字符串操作是遍历字符串的内容。为了支持这一操作，该类还包含了一个迭代器类（iterator）以及相关的运算符重载，以便于对字符串进行遍历等操作。为了避免与标准库中的类名冲突，并且由于迭代器类是特定于String 类的，我们采用了嵌套类的方式，将 iterator 类限制在 String 类内部访问。

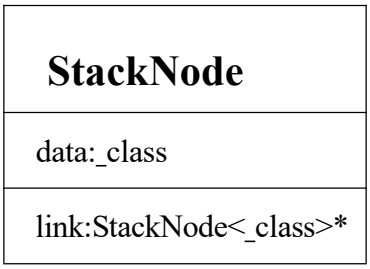
以下是该类及其内部的 iterator 类的主要函数显示如下：

```
//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



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。

其中，主要函数显示如下：

```
//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）

std::map 是 C++标准模板库（STL）中的一种关联容器，用于存储唯一的键值对（key-value pairs），并自动按键进行排序。它提供了快速的查找、插入和删除操作，通常使用红黑树数据结构来实现。可以使用迭代器来遍历 std::map 中的键值对，访问它们的键和值。

2.3 算法设计

2.3.1 算法思路

A 题在正确输入两个变量的情况下，进行四种运算即可。

B C 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，步骤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。

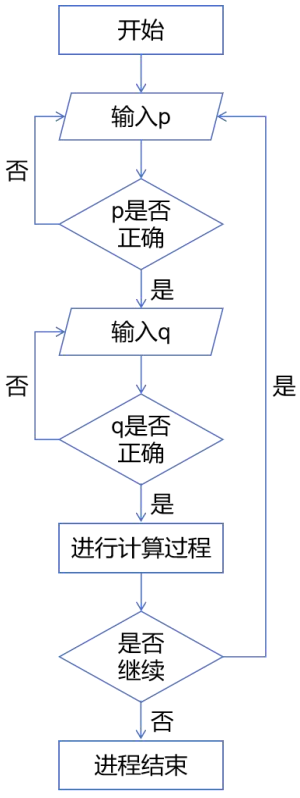
2.3.2 性能评估

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 A 题逻辑联接词的运算设计部分

·流程图表示



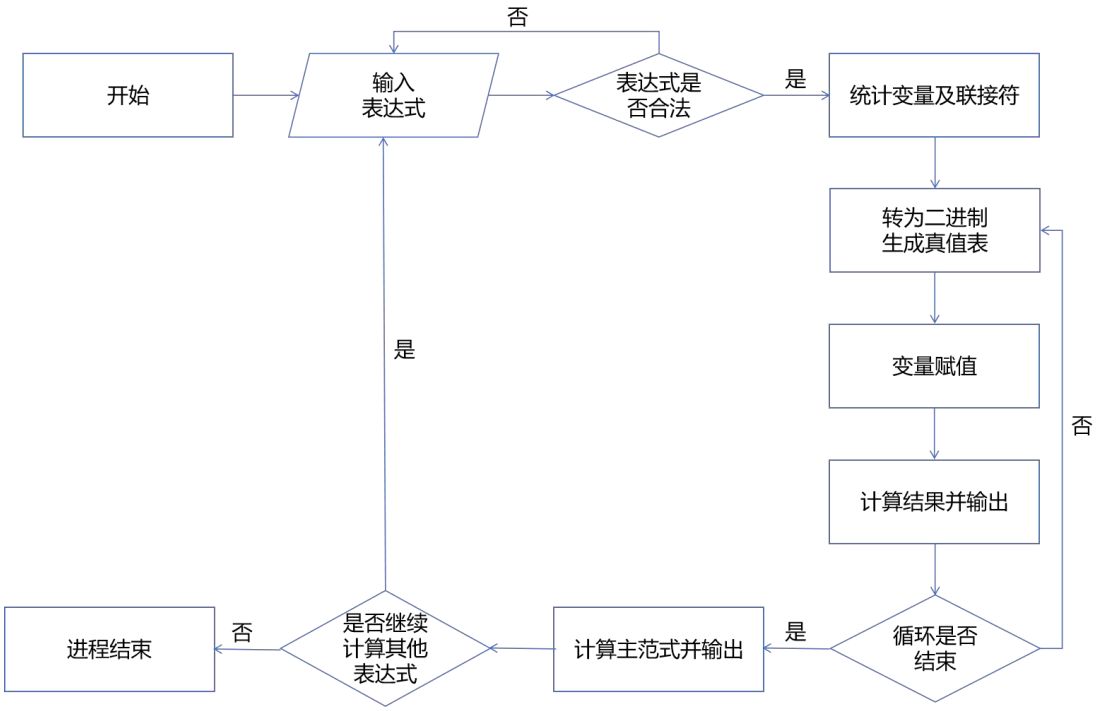
· 代码实现

```
#include <iostream>
using namespace std;
// 显示欢迎消息
void WelInfoShow()
{
    cout << "****************************************" << endl;
    cout << "**                                    **" << endl;
    cout << "**        欢迎进入逻辑运算程序        **" << endl;
    cout << "**                                    **" << endl;
    cout << "****************************************" << endl;
    cout << endl;
}
// 输入处理
int GetInput(char c)
{
    int i = -1;
    cout << "\n  请输入" << c << "的值（0或1）,以回车结束:";
    cin >> i;
    if ((i != 0 && i != 1) || cin.fail())
    {
        cout << "\n  " << c << "的值输入有误,请重新输入!" << endl;
        cin.clear();
        cin.ignore(INT_MAX, '\n');
        return GetInput(c);
    }
    // 防止读入第一个数字后方的输入影响下面输入
    cin.ignore(INT_MAX, '\n');
    return i;
}
// 判断是否继续
bool WheContinue()
{
    bool flag;
    char s;
    cout << "\n是否继续？（y/n）:";
    cin >> s;
    if (s == 'y' || s == 'n')
    {
        if (s == 'y')
            flag = true; // 继续
        else
        {
```

```
            cout << "欢迎下次再次使用!" << endl; // 退出
            flag = false;
        }
    }
    else
    {
        cout << "输入错误,请重新输入!" << endl; // 错误校验
        return WheContinue();
    }
    // 防止读入第一个字母后方的输入影响下面输入
    cin.ignore(INT_MAX, '\n');
    return flag;
}
int main()
{
    WelInfoShow();
    int a[4];
    int P, Q;
    while (1)
    {
        P = GetInput('P');
        Q = GetInput('Q');
        // 与运算
        a[0] = P && Q;
        // 或运算
        a[1] = P || Q;
        // 蕴含运算，将其转化为与或非形式
        a[2] = (!P) || Q;
        // 等值运算，将其转化为与或非形式
        a[3] = ((!P) || Q) && ((!Q) || P);
        cout << "\n\n  合取:\n       P/\\Q = " << a[0] << endl;
        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;
        if (!WheContinue())
            break;
    }
    return 0;
}
```

2.3.4 B 题求真值表与主范式设计部分

・流程图表示



・判断表达式是否合法代码实现

```
bool formulaCheck(const string& str)
{
    //游标对象
    string::const_iterator it = str.begin();
    //记录括号情况的栈
    stack<char> brackets;
    //操作符（单目、双目）个数，数字个数
    int op1Num = 0, op2Num = 0, numNum = 0;

    while (it != str.end()) {
        //若扫描到字符表示命题，则进行计数
        if (((*it) >= 'a' && (*it) <= 'z') || ((*it) >= 'A' && (*it)
<= 'Z'))
            //一个数字扫描完成，进行计数
            numNum++;
        //若遇到左括号，压入栈中
        if (*it == '(')
            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;
    }
    //所有标准均符合，则表达式合法
    return true;
}
```

・一次计算结果代码实现

```
int calculate(string formula, Map_ic pSet, Map_ii value)
{
    //存运算符的栈
    stack<char> opter;
    //存操作数的栈
    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'))
        {
            //将某个命题变项的取值找到，放入栈中
            pvalue.push(value[findProposition(pSet, c)]);
        }
        //遍历运算符
        else
        {

            //取栈顶运算符
            char tmp = opter.top();
            //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
            if (priority[tmp] > priority[c])
            {
                while (priority[tmp] > priority[c] && tmp != '(')
                {
                    check(pvalue, opter);
                    tmp = opter.top();
                    //若遇到#，则说明运算完成，返回结果
                    if (tmp == '#' && c == '#')
                    {
                        return pvalue.top();
                    }
                }
                //栈顶优先级低于 c，c 进栈
                opter.push(c);
            }
            else
                //栈顶优先级低于 c，c 进栈
                opter.push(c);
        }
    }
    return -1;
}
```

・主函数代码实现

```
int main()
{
    //运算符优先级
```

```
    priority['('] = 6;
    priority[')'] = 6;
    priority['!'] = 5;
    priority['&'] = 4;
    priority['|'] = 3;
    priority['^'] = 2;
    priority['~'] = 1;
    priority['#'] = 0;

    while (1)
    {
        PrintMenu();
        string formula;
        //输入公式
        cin >> formula;
        //检查公式是否合法
        if (!formulaCheck(formula))
        {
            cout << "输入公式不合法，请重新输入" << endl;
            cin.clear();
            cin.ignore(INT_MAX, '\n');
            Sleep(800);
            system("cls");
            continue;
        }
        //统计命题变项
        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++)
        {
            //输出所有的命题变项
            cout << proposition_set[i] << "\t";
        }
        cout << formula << endl;
        int* m;
        //该数组依次存放命题公式的各行(0 或 1)的运算结果的值
        m = (int*)malloc(sizeof(int) *
pow2(proposition_set.size()));
        //有 2^n 种取值，遍历一遍
        for (int i = 0; i < pow2(proposition_set.size()); i++)
        {
            //将取值转换为二进制
```

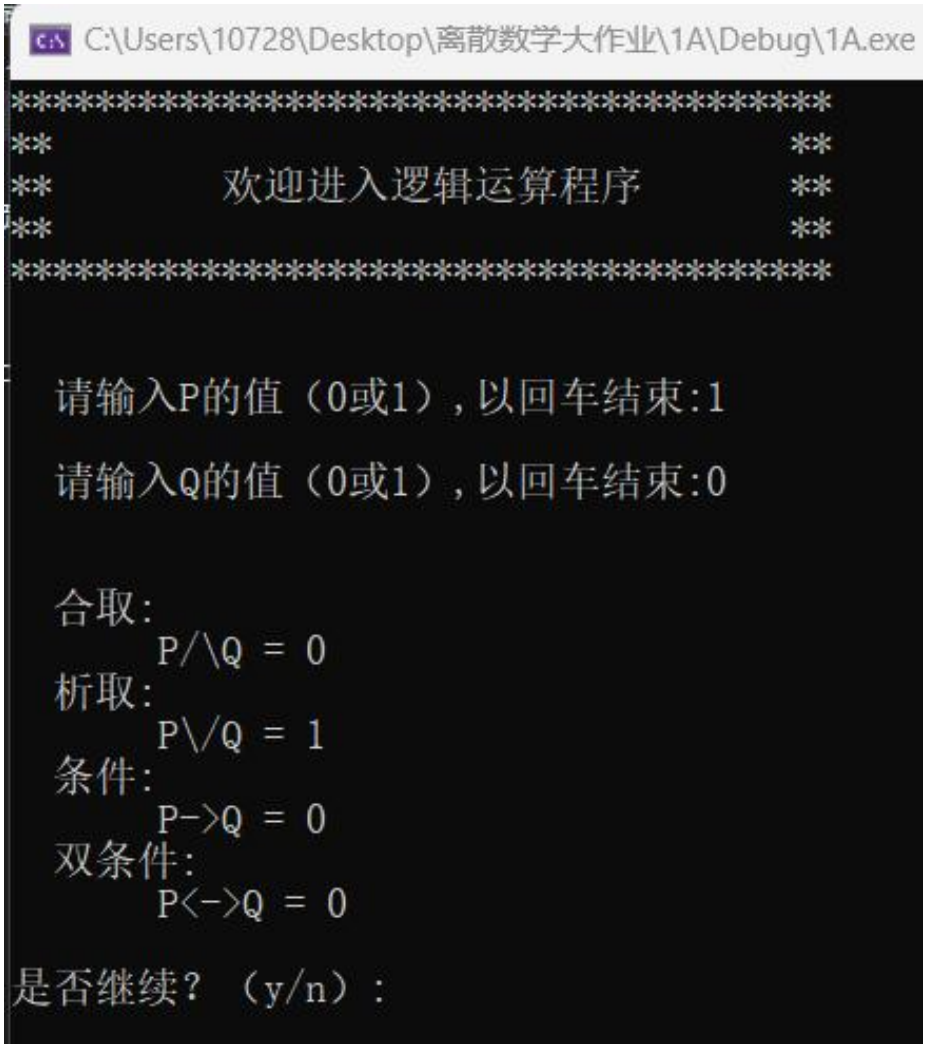
```
            Map_ii bina_set = toBinary(proposition_set.size(),
i);
            for (unsigned int j = 0; j < bina_set.size(); j++)
            {
                //将各命题变项的值打印出来
                cout << bina_set[j] << "\t";
            }
            //计算公式真值
            int result = calculate(formula, proposition_set,
bina_set);
            //记录这一次的真值
            *(m + i) = result;
            //打印这一次的计算结果
            cout << result << endl;
        }
        int n_m = 0, n_M = 0;
        cout << "该命题公式的主析取范式：" << endl;
        //统计成真赋值并打印主析取范式
        for (int i = 0; i < pow2(proposition_set.size()); i++)
        {
            if (*(m + i) == 1)
            {
                if (n_m == 0)
                    cout << "m<" << i << ">";
                else
                    cout << " \\/ m<" << i << "> ";
                n_m++;
            }
        }
        //无成真赋值情况
        if (n_m == 0)
            cout << "0";
        cout << endl;
        cout << "该命题公式的主合取范式：" << endl;
        //统计成假赋值并打印主合取范式
        for (int i = 0; i < pow2(proposition_set.size()); i++)
        {
            if (*(m + i) == 0)
            {
                if (n_M == 0)
                    cout << "M<" << i << ">";
                else
                    cout << " /\\ M<" << i << "> ";
```

```
                n_M++;
            }
            //无成假赋值情况
            if (n_M == 0)
                cout << "0";
        }
        cout << endl << endl;
        //是否再次进行运算其它表达式
        cout << "是否继续运算(Y/N)？ Y代表继续 N代表退出 大小写不敏感" << endl;
        char ch;
        while (1)
        {
            cin >> ch;
            if (ch == 'Y' || ch == 'y')
            {
                cin.clear();
                cin.ignore(INT_MAX, '\n');
                system("cls");
                break;
            }
            else if (ch == 'N' || ch == 'n')
            {
                return 0;
            }
            else
            {
                cout << "输入错误，请重新输入" << endl;
                cin.clear();
                cin.ignore(CHAR_MAX, '\n');
                continue;
            }
        }
        continue;
    }
}
```

3. 项目测试

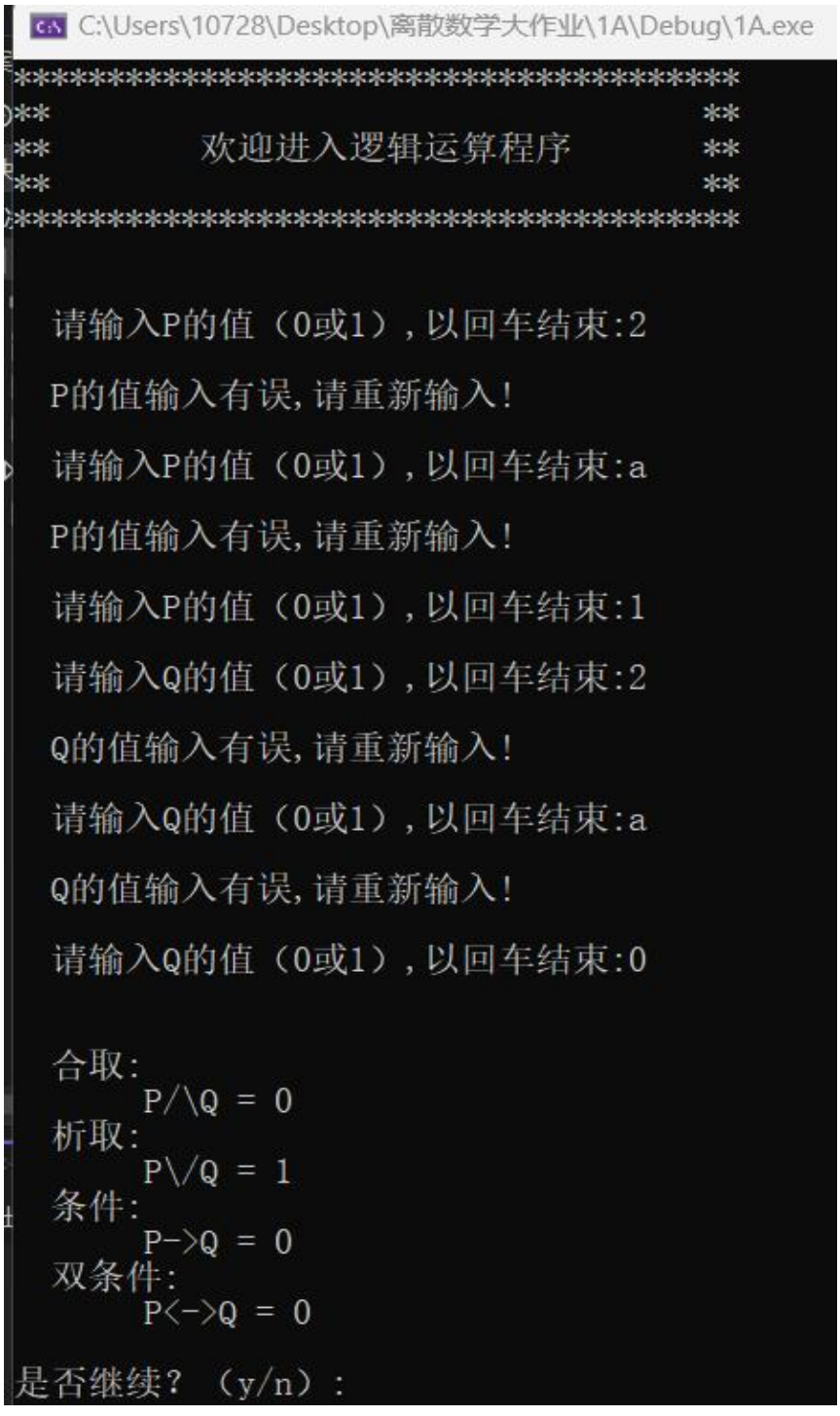
3.1 A 题：一般情况

测试结果：



3.2 A 题：错误输入处理

测试结果：

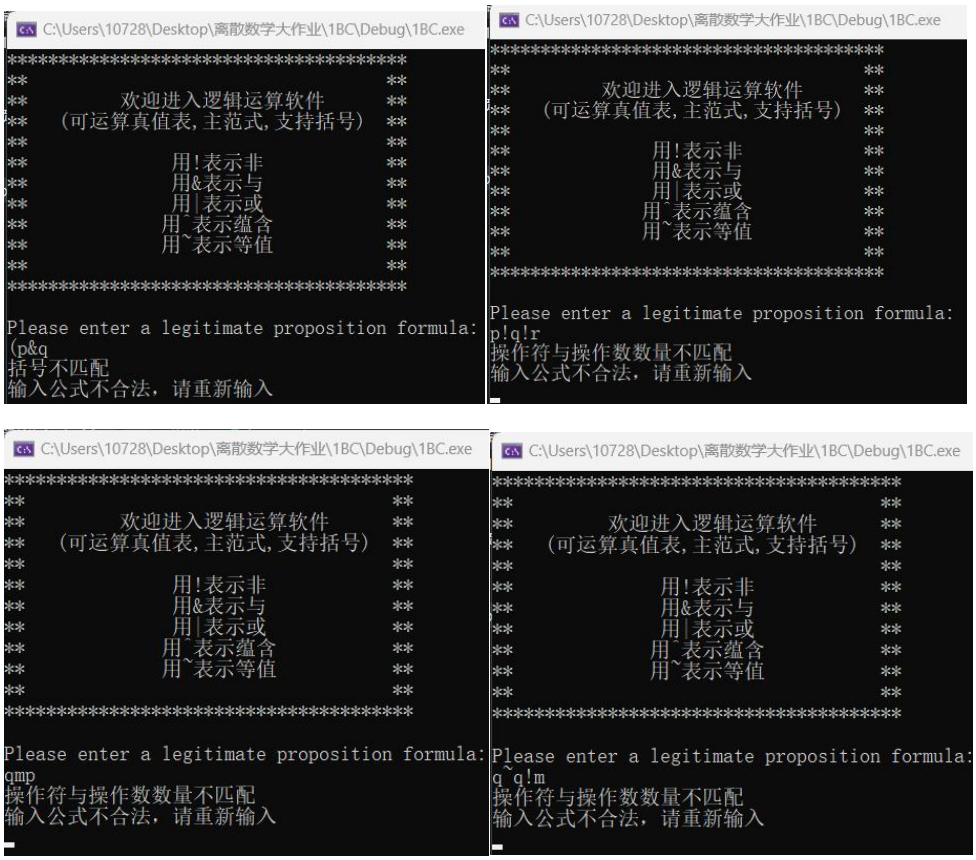


3.3 B C 题：一般情况

测试结果：

3.4 B C题：错误输入处理

测试结果：



4. 心得体会

在编写此次项目代码时，我使用自定义的数据结构来表示联接词和命题符号，这样可以更轻松地进行操作和解析。使用嵌套循环来生成所有可能的输入组合，并计算每个组合下的输出。在生成真值表时，要注意逻辑运算符的优先级和结合性。使用布尔代数规则和逻辑分配律来将逻辑表达式转化为主范式。使用已知的逻辑表达式和真值表来验证代码的正确性逻辑代码可以模块化设计，将不同的功能模块分开，使代码更易于理解和维护。考虑将逻辑代码设计为可重用的库，这样可以在多个项目中共享逻辑计算功能。

总之，编写命题逻辑的联接词、真值表和主范式的代码是一个深入理解离散数学和逻辑思维的过程。通过将理论应用到实践中，可以更好地掌握逻辑概念，并提高问题解决的能力。