**项目说明文档**

**离散数学课程实验**

**——命题逻辑联接词、**

**真值表、主范式**

作 者 姓 名 刘淑仪

学 号 2251730

指 导 教 师 唐剑锋

学 院 专 业 软件学院 软件工程



二〇二三 年 十 月 二十五 日

目录

1 项目分析 1

1.1 实验目的分析 1

1.2 实验内容概述 1

1.2.1 A题：逻辑联接词的运算 1

1.2.2 B、C题：求一个命题公式的真值表、DNF、CNF 1

2 项目设计与实现 2

2.1 实验原理 2

2.2 实验过程设计 3

2.2.1 A题 3

2.2.2 B、C题 4

2.3 项目具体实现 8

2.3.1 A题 8

2.3.2 B、C题 8

3 项目实现与测试 9

3.1 A题 9

3.2 B、C题 10

4 源代码展示 13

4.1 A题 13

4.2 B、C题 14

5 集成开发环境与编译运行环境 21

# 项目分析

## 实验目的分析

该实验课程的目标是培养学生在命题逻辑领域的技能，包括联接词的运用、真值表的构建以及主范式的应用，以便进一步将这些技能应用于解决实际问题。通过实验课程的学习，学生将提升其编写实验报告和总结实验结果的能力，同时培养他们的程序设计思维，使他们能够独立完成简单算法的设计和分析。

## 实验内容概述

### A题：逻辑联接词的运算

利用C++编写的程序，旨在实现二元逻辑运算的计算。具体地，程序能够计算两个命题变元 P 和 Q 的合取（AND）、析取（OR）、条件（IMPLIES）和双条件（IFF）表达式的真值。

1.输入命题变元 P 的真值（0 或 1），表示假或真。

2.输入命题变元 Q 的真值（0 或 1）。

3.程序计算并显示以下逻辑运算的结果：

合取 (AND): P∧Q

析取 (OR): P∨Q

条件 (IMPLIES): P->Q

双条件 (IFF): P<->Q

4.判断程序是否继续进行:

若回答 'Y' 或 'y'，程序返回到第一步。

若回答 'N' 或 'n'，程序结束运行。

### B、C题：求一个命题公式的真值表、DNF、CNF

利用C＋＋程序，实现任意输入公式的真值表，主析取范式（DNF）和主合取范式（CNF）。程序的核心是一个二进制加法器模拟器，用于为命题变元生成所有可能的真值指派。

从键盘输入一个命题公式，求任意一个命题公式的真值表（B），并根据真值表求主析取范式，主合取范式（C）。

1. 按照提示，键盘输入一个合法的不含空格的命题公式

2. 程序首先将命题变元的所有可能组合放在真值表的左侧。对于每个组合，计算合式公式的真值并将其放在真值表的右侧。根据当前的真值指派，计算逻辑公式的结果。此结果反映了在特定变元赋值下公式的真值。程序输出真值表，展示了每个真值指派及其对应的公式真值。

3. 根据生成的真值表，计算并展示公式的主析取范式和主合取范式。

4. 程序结束

# 项目设计与实现

## 实验原理

（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。

## 实验过程设计

### A题

定义了一个名为 Solution 的类，该类包含多个成员函数来实现程序的不同部分。显示主菜单（通过 Menu() 函数），然后进入一个循环，循环中不断获取用户输入的布尔值（P 和 Q），计算逻辑运算的结果，直到用户选择不继续。

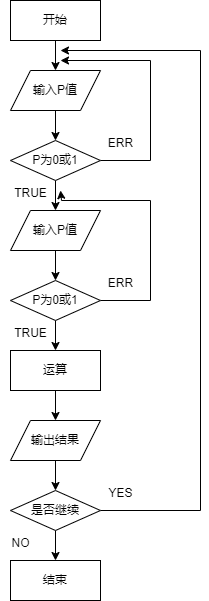
bool GetInput(const char\* prompt) const: 获取用户的输入（0 或 1），并验证输入的正确性。如果输入无效（即不是 0 或 1），会提示用户重新输入。bool AskContinue() const: 询问用户是否继续进行计算。如果用户输入 'Y' 或 'y'，则返回 true，继续运行；否则返回 false。void CalculateAndOutput(bool p, bool q) const: 根据用户输入的布尔值 P 和 Q，计算以下逻辑运算的结果，并将结果输出到控制台：

合取 (AND): P∧Q，当且仅当 P 和 Q 都为真时结果为真。

析取 (OR): P∨Q，只要 P 或 Q 中至少有一个为真，结果就为真。

条件 (IMPLIES): P->Q，当 P 为假或 Q 为真时结果为真。

双条件 (IFF): P<->Q，当 P 和 Q 有相同的真值时结果为真。



### B、C题

定义了一个名为 PropositionalFormula 的类，专门用于处理和计算命题逻辑公式。该类包括多个方法来执行不同的功能：构造函数 PropositionalFormula(const string& f): 初始化逻辑公式，并定义运算符的优先级。bool isOperator(char ch) 和 bool isVariable(char ch): 分别用于检查字符是否为运算符和变量。set<char> getVariables() const: 提取并返回公式中所有唯一的变量。void setVariableValue(char var, bool value): 为公式中的变量赋值。bool isValid(): 检查逻辑公式的格式是否正确。int calculate(): 计算逻辑公式的真值。

此外，定义了 TruthTable 类来生成和展示 PropositionalFormula 类中公式的真值表。TruthTable 类的关键方法包括：构造函数 TruthTable(PF& f): 接收一个 PropositionalFormula 对象作为参数。void generateTruthTable(): 生成真值表。void display(): 显示真值表。string getDNF() 和 string getCNF(): 分别计算并返回公式的主析取范式和主合取范式。

用户输入的逻辑公式首先被 PropositionalFormula 类处理。如果公式有效（通过 isValid() 方法验证），则继续后续步骤——使用 TruthTable 类的实例来生成并显示逻辑公式的真值表。最后计算并展示 DNF 和 CNF。



开始

检查括号

是否是最内级括号

运算内容

是否是最后结果

返回结果

结束

开始

结束

Y

Y

N

N

非运算

与运算

或运算

蕴含运算

等值运算

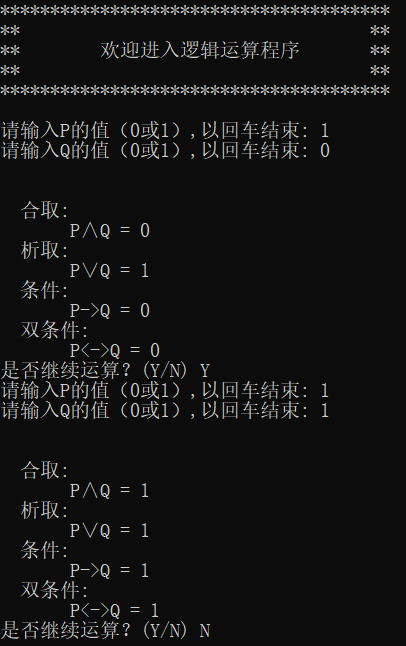
返回结果

**主运算函数**

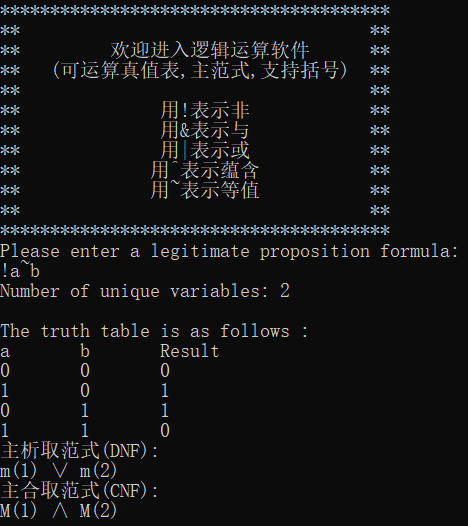
**分级运算函数**

## 项目具体实现

### A题



### B、C题



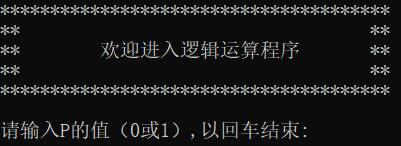
# 项目实现与测试

## A题

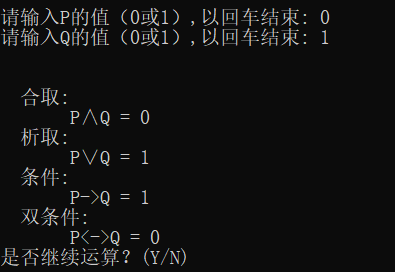
结果分析：

这道题主要是读取数值并进行计算，输出两个命题变元(P和Q)的合取，析取，条件和双向条件的真值结果，同时要注意输入的值要必须0或1，如果不是，则进行错误提示，并进行重新输入。

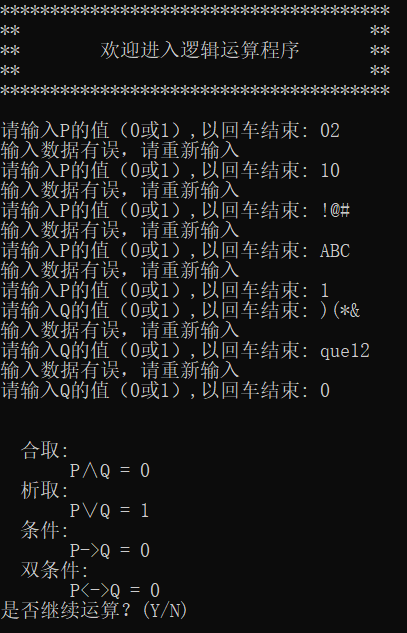
进入界面：



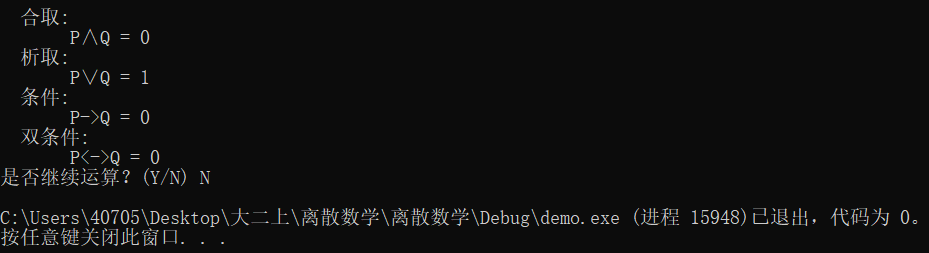
正确输入并进行运算：



错误输入处理：



退出程序：

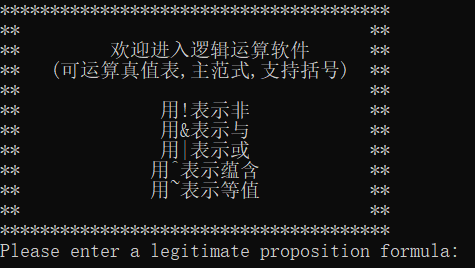


## B、C题

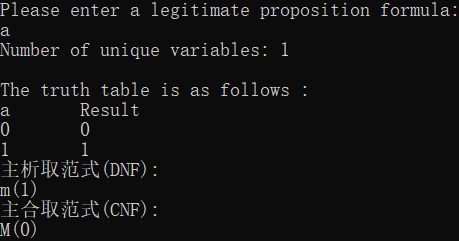
结果分析：

这段代码的核心功能是读取用户输入的命题逻辑公式，并基于该公式生成真值表，主析取范式（DNF）和主合取范式（CNF）。程序首先确保输入的逻辑公式格式正确，包括检查是否包含合法的运算符和变量，以及括号是否匹配。一旦确认公式有效，程序将生成并显示真值表、计算并显示 DNF 和 CNF。

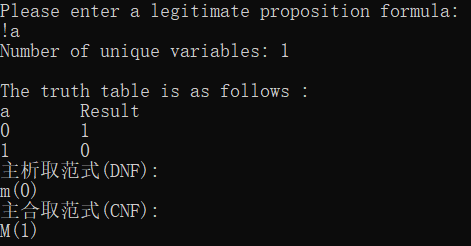
进入界面：



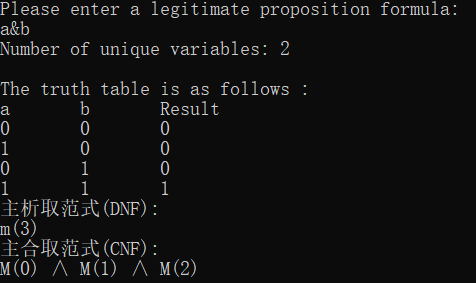
单一运算并得到相应结果：



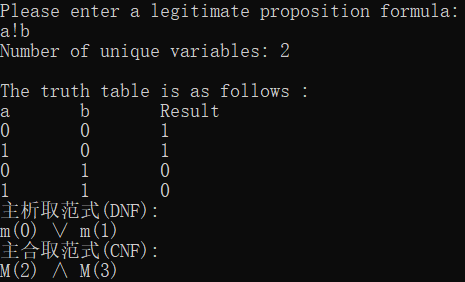
输入非运算公式并得到相应结果：



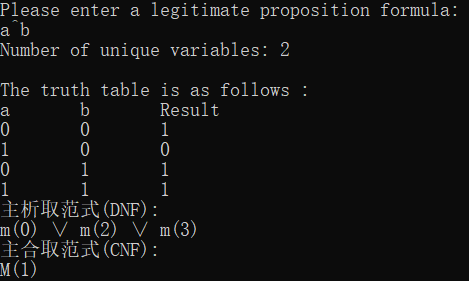
输入与运算公式并得到相应结果：



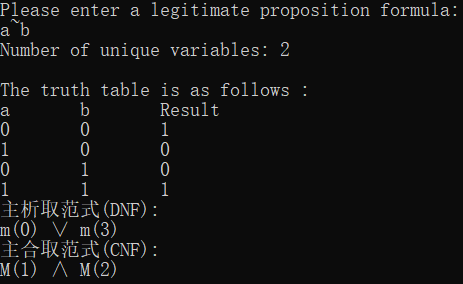
输入或运算公式并得到相应结果：



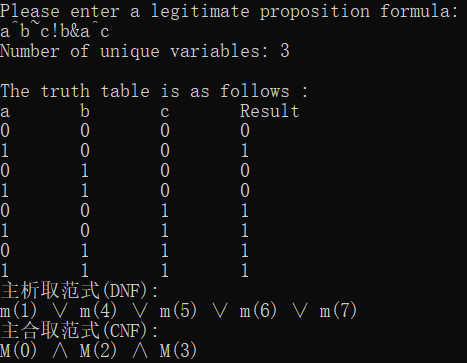
输入蕴含运算公式并得到相应结果：



输入等值运算公式并得到相应结果：



输入综合运算公式并得到相应结果：



# 源代码展示

## A题

#include <iostream>

class Solution {

public:

    void Run();

private:

    void Menu() const;

    bool GetInput(const char\* prompt) const;

    bool AskContinue() const;

    void CalculateAndOutput(bool p, bool q) const;

};

void Solution::Run() {

    Menu();

    do {

        bool p = GetInput("请输入P的值（0或1）,以回车结束: ");

        bool q = GetInput("请输入Q的值（0或1）,以回车结束: ");

        CalculateAndOutput(p, q);

    } while (AskContinue());

}

void Solution::Menu() const {

    std::cout << "\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\n"

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

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

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

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

}

bool Solution::GetInput(const char\* prompt) const {

    bool input;

    while (true) {

        std::cout << prompt;

        std::cin >> input;

        if (std::cin.fail() || (input != 0 && input != 1)) {

            std::cin.clear();

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

            std::cout << "输入数据有误，请重新输入\n";

        }

        else {

            return input;

        }

    }

}

bool Solution::AskContinue() const {

    char ch;

    std::cout << "是否继续运算？(Y/N) ";

    std::cin >> ch;

    return ch == 'Y' || ch == 'y';

}

void Solution::CalculateAndOutput(bool p, bool q) const {

    bool andResult = p && q;

    bool orResult = p || q;

    bool impliesResult = !p || q;

    bool iffResult = (p && q) || (!p && !q);

    std::cout << "\n\n  合取:\n       P∧Q = " << andResult << std::endl

        << "  析取:\n       P∨Q = " << orResult << std::endl

        << "  条件:\n       P->Q = " << impliesResult << std::endl

        << "  双条件:\n       P<->Q = " << iffResult << std::endl;

}

int main() {

    Solution operationOfLogicalConnectives;

    operationOfLogicalConnectives.Run();

    return 0;

}

## B、C题

#include <iostream>

#include <string>

#include <map>

#include <stack>

#include <vector>

#include <cctype>

#include <cmath>

#include <set>

using namespace std;

// 辅助函数：将中缀表达式转换为后缀表达式

vector<char> toPostfix(const string& infix, const map<char, int>& priority) {

    vector<char> postfix;

    stack<char> opStack;

    for (char token : infix) {

        if (isalpha(token)) {

            postfix.push\_back(token);

        }

        else if (token == '(') {

            opStack.push(token);

        }

        else if (token == ')') {

            while (!opStack.empty() && opStack.top() != '(') {

                postfix.push\_back(opStack.top());

                opStack.pop();

            }

            opStack.pop();  // Pop '('

        }

        else {

            while (!opStack.empty() && priority.at(token) <= priority.at(opStack.top())) {

                postfix.push\_back(opStack.top());

                opStack.pop();

            }

            opStack.push(token);

        }

    }

    while (!opStack.empty()) {

        postfix.push\_back(opStack.top());

        opStack.pop();

    }

    return postfix;

}

/\* 解析与计算公式 \*/

typedef class PropositionalFormula {

    string formula;

    map<char, int> priority;        // 存储运算符的优先级

    map<char, bool> variables; // 存储变量及其赋值

    bool isOperator(char ch) { return priority.find(ch) != priority.end(); }

    bool isVariable(char ch) { return isalpha(ch); }

public:

    // 定义运算符优先级

    PropositionalFormula(const string& f) : formula(f) {

        priority['('] = 6;

        priority[')'] = 6;

        priority['!'] = 5;

        priority['&'] = 4;

        priority['|'] = 3;

        priority['^'] = 2;

        priority['~'] = 1;

        priority['#'] = 0;

    }

    // 获取公式中的所有变量

    set<char> getVariables() const {

        set<char> vars;

        for (char ch : formula) {

            if (isalpha(ch)) {  // 直接检查字符是否为字母

                vars.insert(ch);

            }

        }

        return vars;

    }

    // 设置变量的值

    void setVariableValue(char var, bool value) {

        if (isVariable(var)) {

            variables[var] = value;

        }

    }

    // 检查公式是否有效

    bool isValid();

    // 计算真值

    int calculate();

}PF;

// 检查公式是否有效

bool PropositionalFormula::isValid()

{

    if (formula.empty()) return false;

    int balance = 0;

    bool lastWasOperator = true;  // 假设公式以运算符开始

    for (char ch : formula) {

        if (ch == '(') {

            balance++;

            lastWasOperator = true;  // 括号后应该是运算符或变量

        }

        else if (ch == ')') {

            balance--;

            if (balance < 0 || lastWasOperator) return false; // 不匹配的或空的括号

            lastWasOperator = false;

        }

        else if (isVariable(ch)) {

            if (!lastWasOperator) return false; // 变量之间缺少运算符

            lastWasOperator = false;

        }

        else if (isOperator(ch)) {

            if (lastWasOperator && ch != '!' && ch != '(') return false; // 连续的运算符（除了非运算符）

            lastWasOperator = true;

        }

        else {

            return false; // 非法字符

        }

    }

    return balance == 0 && !lastWasOperator; // 确保所有括号都闭合且不以运算符结束

}

// 计算真值

int PropositionalFormula::calculate()

{

    auto postfix = toPostfix(formula, priority);  // 将中缀表达式转换为后缀表达式

    stack<bool> valStack;  // 用于存储布尔值的栈

    // 遍历后缀表达式

    for (char token : postfix) {

        if (isalpha(token)) {  // 如果是变量，则压入其值

            valStack.push(variables.at(token));

        }

        else {  // 否则，应用逻辑运算符

            bool right = valStack.top();

            valStack.pop();

            if (token == '!') {  // 非运算符

                valStack.push(!right);

            }

            else {  // 其他运算符需要两个操作数

                bool left = valStack.top();

                valStack.pop();

                switch (token) {

                case '&':

                    valStack.push(left && right);

                    break;

                case '|':

                    valStack.push(left || right);

                    break;

                case '^':  // 修改此处，将其作为蕴含运算

                    valStack.push(!left || right); // A → B 是当 A 为假或 B 为真时为真

                    break;

                case '~':

                    valStack.push(left == right);

                    break;

                }

            }

        }

    }

    return valStack.top();  // 栈顶元素就是整个表达式的结果

}

/\* 生成与显示真值表 \*/

class TruthTable {

    PF& formula;

    vector<map<char, bool>> truthTable; // 存储每行的变量赋值和结果

    vector<bool> results; // 存储每行的结果

public:

    TruthTable(PF& f) : formula(f) {}

    // 生成真值表

    void generateTruthTable();

    // 显示真值表

    void display();

    string getDNF();

    string getCNF();

};

// 生成真值表

void TruthTable::generateTruthTable()

{

    // 提取所有变量

    set<char> vars = formula.getVariables();

    int numVars = vars.size();

    int numRows = pow(2, numVars);

    // 生成并计算每个真值指派

    for (int i = 0; i < numRows; ++i) {

        map<char, bool> row;

        int mask = 1;

        for (char var : vars) {

            bool value = (i & mask) > 0;

            formula.setVariableValue(var, value);

            row[var] = value;

            mask <<= 1;

        }

        truthTable.push\_back(row);

        results.push\_back(formula.calculate());

    }

}

// 显示真值表

void TruthTable::display()

{

    cout << "The truth table is as follows : \n";

    for (const auto& var : truthTable[0]) {

        cout << var.first << "\t";

    }

    cout << "Result\n";

    // 打印每行的真值指派和结果

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

        for (const auto& var : truthTable[i]) {

            cout << var.second << "\t";  // 直接打印 1 或 0

        }

        cout << results[i] << "\n"; // 同样直接打印 1 或 0 表示结果

    }

}

// 主析取范式

string TruthTable::getDNF() {

    string dnf;

    int index = 0; // 通用索引，用于m(n)和M(n)

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

        if (results[i]) { // 如果结果为真

            if (!dnf.empty()) dnf += " ∨ ";

            dnf += "m(" + to\_string(index) + ")";

        }

        index++; // 无论结果如何，索引都递增

    }

    return dnf.empty() ? "FALSE" : dnf;

}

// 主合取范式

string TruthTable::getCNF() {

    string cnf;

    int index = 0; // 通用索引，用于m(n)和M(n)

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

        if (!results[i]) { // 如果结果为假

            if (!cnf.empty()) cnf += " ∧ ";

            cnf += "M(" + to\_string(index) + ")";

        }

        index++; // 无论结果如何，索引都递增

    }

    return cnf.empty() ? "TRUE" : cnf;

}

// 菜单

void menu()

{

    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";

    cout << "Please enter a legitimate proposition formula: \n";

}

// 检查变量个数

int countUniqueVariables(const string& expression) {

    set<char> variables;

    for (char ch : expression) {

        if (isalpha(ch)) { // 检查字符是否是字母

            variables.insert(ch); // 将字母（变量）添加到集合中

        }

    }

    return variables.size(); // 集合的大小即唯一变量的个数

}

int main()

{

    string inputFormula;

    menu();

    getline(cin, inputFormula);

    int count = countUniqueVariables(inputFormula);

    cout << "Number of unique variables: " << count << endl << endl;

    PF pf(inputFormula);

    if (pf.isValid()) {

        TruthTable tt(pf);

        tt.generateTruthTable();

        tt.display();

        cout << "主析取范式(DNF): \n" << tt.getDNF() << endl;

        cout << "主合取范式(CNF): \n" << tt.getCNF() << endl;

    }

    else {

        cout << "Invalid formula!\n";

    }

    return 0;

}

# 集成开发环境与编译运行环境

Windows系统：Windows 11 x64

Windows集成开发环境：Microsoft Visual Studio 2022 (Release模式)

Windows编译运行环境：本项目适用于x86架构和x64架构