同济大学
TONGJI UNIVERSITY

离散数学

课程实验报告

命题逻辑联接词、真值表、主范式

姓　　名：　　马小龙

学　　号：　　2353814

学　　院：计算机科学与技术学院（软件学院）

专　　业：　　软件工程

指导教师：　　李冰

二〇二四年十一月二十九日

目录

一 实验目的.....1
二 实验内容.....1
2.1 逻辑联接词的运算.....1
2.2 真值表与主范式.....1
三 实验环境.....1
四 实验原理.....2
4.1 合取.....2
4.2 析取.....2
4.3 条件.....3
4.4 双向条件.....3
4.5 真值表.....3
4.6 主合取范式.....4
4.7 主析取范式.....4
五 命题逻辑联接词实验过程.....4
5.1 实验思路.....5
5.2 程序框架设计.....5
5.3 程序功能实现.....5
5.3.1 逻辑值输入功能的实现.....5
5.3.2 程序退出功能的实现.....6
5.4 核心算法实现.....6
六 真值表、主范式实验过程.....6
6.1 实验思路.....7
6.2 数据结构.....7
6.3 程序框架设计.....7
6.4 程序功能实现.....8
6.4.1 合法性检查功能.....8
6.4.2 逻辑计算功能.....9
6.5 核心算法.....10
6.5.1 真值表计算.....10
6.5.2 主范式.....10
七 实验结果测试.....10
7.1 命题逻辑联接词.....10
7.2 真值表、主范式.....11
八 实验心得.....13
附录·源码.....15

命题逻辑联接词.....15
真值表、主范式.....16

一 实验目的

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。

二 实验内容

2.1 逻辑联接词的运算

本部分实验的目标是实现和理解命题逻辑中的基本运算符：合取（∧）、析取（∨）、蕴含（条件）以及双向条件（↔）。这些逻辑联接词用于连接命题，构成复合命题，并根据命题的真值计算出结果。在本部分实验中，用户将输入两个命题的真值（P 和 Q），并根据不同的联接词计算结果。通过这个实验，学生将掌握命题联接词的运算规则，并能够编写程序计算命题公式的真值。

2.2 真值表与主范式

本部分实验将扩展到任意命题逻辑公式的真值表生成，并基于该真值表计算主析取范式（DNF）和主合取范式（CNF）。

真值表：真值表显示了在所有可能的命题变元取值组合下，命题公式的真值。对于每个命题公式，程序会生成所有可能的真值组合，并计算对应的公式值。真值表是分析命题公式性质的重要工具。

主析取范式（DNF）：主析取范式是将命题公式转化为一组极小项的析取（“或”）的标准形式。每个极小项是由命题变元或其否定构成的合取（“与”）表达式。通过真值表，程序能够识别哪些变元组合使公式为真，并将其转化为 DNF。

主合取范式（CNF）：主合取范式是将命题公式转化为一组极大项的合取（“与”）的标准形式。每个极大项是由命题变元或其否定构成的析取（“或”）表达式。通过真值表，程序能够识别哪些变元组合使公式为假，并将其转化为 CNF。

三 实验环境

编程开发语言：C++

文 件 编 码：UTF-8

集成开发环境：Clion 2024

工 具 集：MinGW 11.0 w64

四 实验原理

4.1 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 为真,Q 为真时，P∧Q 为真, 而 P、Q 只要有一方为假，则 P∧Q 为假。

P	Q	P∧Q
0	0	0
0	1	0
1	0	0
1	1	1

表 4.1 合取运算真值表

4.2 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 为假,Q 为假时，P∨Q 为假, 而 P、Q 只要有一为真则 P∨Q 为真。

P	Q	P∨Q
0	0	0
0	1	1
1	0	1
1	1	1

表 4.2 析取运算真值表

4.3 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 为真, Q 为假时，P→Q 才为假, 其余均为真。

P	Q	P→Q
0	0	1
0	1	1
1	0	0
1	1	1

表 4.3 条件运算真值表

4.4 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，也称 P、Q 等价。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 为真, Q为真时P←→Q 为真, 其余均为假。

P	Q	P→Q
0	0	1
0	1	1
1	0	0
1	1	1

表 4.4 双向条件运算真值表

4.5 真值表

真值表是逻辑学中用于分析命题公式真值的重要工具，它列出命题公式在所有可能的命题变元取值组合下的真值。通过真值表，我们可以直观地看到一个逻辑公式在不同条件下的结果，从而帮助我们理解公式的逻辑性质和推理过程。

真值表的基本构成包括命题变元、逻辑运算符及其应用结果。命题变元是构成公式的基本单位，通常用字母表示（如 P、Q、R）。每个命题变元的真值可以是“真”或“假”（分别用 1 和 0 表示）。逻辑运算符，如合取（∧）、析取（∨）、条

件（→）等，根据其定义在真值表中生成不同的结果。例如，对于合取运算，只有当两个命题都为真时，合取命题才为真；否则为假。

构建真值表时，我们首先列出所有命题变元的真值组合。对于有 n 个命题变元的公式，真值表将有 2^n 行。接着，我们根据公式中的逻辑联接词计算每一行的结果，并最终得出整个公式的真值。真值表的应用不仅限于求得公式在不同输入下的真假值，还能用来验证命题公式的性质，如是否为重言式（在所有输入下都为真）或矛盾式（在所有输入下都为假）。

真值表为我们提供了一种简洁、系统的方法，帮助理解命题逻辑中的复杂公式，进行逻辑推理，并验证公式的有效性。通过真值表，我们能够更清晰地看到逻辑公式在不同条件下的表现，从而在逻辑推理和证明中起到关键作用。

4.6 主合取范式

主合取范式（CNF，Conjunctive Normal Form）是逻辑公式的一种标准化形式，它将命题公式表达为若干个极大项的合取（“与”）形式。每个极大项是由命题变元及其否定构成的析取（“或”）表达式，并且每个命题变元在一个极大项中只出现一次。主合取范式的核心思想是将一个复杂的逻辑公式转化为一系列命题变元的析取形式，然后用“与”连接这些析取项。

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。

4.7 主析取范式

主析取范式（DNF，Disjunctive Normal Form）是逻辑公式的一种标准化形式，它将命题公式表示为若干个极小项的析取（“或”）形式。每个极小项是由命题变元或其否定构成的合取（“与”）表达式，并且每个命题变元在一个极小项中只出现一次。主析取范式的核心在于将一个复杂的逻辑公式转化为若干个合取项，通过“或”连接这些合取项。

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。

五 命题逻辑联接词实验过程

5.1 实验思路

本实验旨在设计并实现一个命题逻辑运算程序，能够根据用户输入的命题变元（P 和 Q）的真值，通过逻辑运算符（与、或、条件、双条件）计算并输出结果。程序要能正确处理用户输入，确保输入有效并支持重复运算，直到用户选择退出。通过此实验，我们实现了命题逻辑运算的基本功能，并增强了对命题逻辑的理解。

5.2 程序框架设计

程序的框架大致分为几个部分：

用户输入：程序通过 Input 函数接收用户输入，确保用户输入为合法的真值（0 或 1）。如果输入无效，程序会提示并要求重新输入。

逻辑运算：程序通过布尔运算符（&&、||、!）计算输入命题变元的与、或、条件和双条件运算结果。

输出结果：每次计算完成后，程序将结果输出到屏幕，并提示用户是否继续进行运算。

退出机制：用户可以选择是否继续运算，通过输入 'Y' 或 'N' 来决定是否继续执行。

5.3 程序功能实现

5.3.1 逻辑值输入功能的实现

在本实验中，输入验证逻辑的主要任务是确保用户输入的命题变元（p 和 q）的值为有效的布尔值，即只能是 0 或 1，代表“假”或“真”。这一过程的实现依赖于对用户输入的检查和反馈机制，确保用户每次输入都符合预期的格式，主要通过 Input 函数实现，以下是详细的输入验证逻辑说明：

首先进入一个 while 循环，使用 std::cin 来接收用户输入，用户被提示输入命题变元的值（0 或 1），并在输入完成后按回车键结束输入。

用户输入的内容首先被存储在 double 类型的变量 input 中，这样做是为了处理可能的非整数输入（例如，浮动小数或错误的字符输入）。

使用 std::cin.fail() 检查输入流是否发生错误。此函数会在用户输入一个非数字字符（如字母或其他特殊符号）时返回 true。如果输入不是有效数字，程序会清除输入流并提示用户重新输入。

然后，使用逻辑条件 input != 0 && input != 1 检查用户输入是否为 0 或 1，即命题变元的合法取值。如果用户输入的值不等于 0 或 1，则输入不合法。如果检测到无效输入（包括非法字符或数字以外的值），程序会调用 std::cin.clear() 清除输入流的错误状态，恢复输入流的正常状态。随后使用 std::cin.ignore(std::numeric_limits<std::streamsize>::max(), '\n') 来丢弃输入缓冲区中的所有内容，直到遇到换行符。这保证了用户输入的下一个值不会受到错误输入的影响。同时，会输出错误提示信息（如："值输入有误，请重新输入"），然后重新进入循环，直到用户输入合法的值（0 或 1）。

当用户输入符合要求时，程序跳出输入循环，返回有效的布尔值（0 或 1）。

5.3.2 程序退出功能的实现

在每次逻辑运算完成后，程序询问用户是否继续进行计算，要求用户输入一个字符（Y、y、N 或 n），以决定是否继续执行下一轮的运算。

用户输入通过 do-while 循环进行处理。程序使用_getch() 函数获取用户的输入，当用户输入的不是 Y、y、N 或 n 四个字符之一时，do-while 会继续执行，直到用户输入有效字符。

如果用户输入的是 Y 或 y，程序会继续执行主循环，开始新的命题逻辑运算。

如果用户输入的是 N 或 n，程序退出主循环，输出提示信息并退出程序。

5.4 核心算法实现

合取运算（∧）通过代码"p && q" 计算真值。

析取运算（∨）通过代码"p || q"计算真值。

条件运算（→）通过代码 "!p || q"计算真值，即通过其等价式¬ p∨q 计算。

双向条件（↔）：通过代码"(!p || q) && (!q || p)" 计算真值，即通过其等价式（¬p∨q）∧（¬ q∨p）计算真值。

六 真值表、主范式实验过程

6.1 实验思路

本实验旨在实现一个支持命题逻辑运算的程序，用户可以输入一个包含逻辑运算符（如与、或、非、条件、双条件）和变量（如 P、Q 等）的命题公式，程序将根据该公式计算命题在各种赋值下的真值，从而生成真值表，并计算出公式的主合取范式（CNF）和主析取范式（DNF）。

6.2 数据结构

栈（stack）：

std::stack<char> parentheses 存储括号，处理括号匹配问题。

std::stack<char> operators 用于存储运算符并控制运算的顺序（根据优先级）

std::stack<bool> operands 用于存储运算数和计算结果。

映射（unordered_map）：

unordered_map<char, bool> 来存储命题公式中变量与其对应的真值，方便查找命题变量真值。

字符串（string）：

std::string initial_logical_expression 存储输入的命题公式；

std::string store_variable 存储命题变量；

动态数组（vector）：

std::vector<int> conjunction 用于存储主合取范式（CNF）。

std::vector<int> conjunction, disjunction 用于存储主析取范式（DNF）

6.3 程序框架设计

1.程序首先输出欢迎界面，并提示用户输入命题公式。输入的公式会经过 LegalExpression() 函数验证，确保符合命题逻辑语法规则。如果不合法，程序会要求重新输入。

2.程序通过 TruthTableNormalForm() 函数生成真值表和主范式，该函数会计算所有可能的变量值组合，并使用 Evaluate() 函数计算每种组合下的公式真值。

3. Evaluate() 函数计算真值时使用栈来处理表达式的运算符和操作数。通过运算符优先级的控制，程序能够处理复杂的逻辑公式，支持括号、运算符优先级和非操作符的特殊处理。Calculate() 函数根据运算符类型调用对应的逻辑函数（如 AND、OR、NOT 等），实现基本的逻辑运算。

4.根据公式的真值，程序将对应的序号存储在 conjunction 和 disjunction 数组中，分别表示主合取范式和主析取范式中的项。

5.程序输出变量的取值情况和公式的真值，借助真值生成真值表，并展示公式的主合取范式和主析取范式。

6.每次运算后，程序会询问用户是否继续进行下一轮运算。如果用户输入 N 或 n，则程序结束；如果输入 Y 或 n 继续执行下一轮计算。

6.4 程序功能实现

6.4.1 合法性检查功能

合法性检查功能是本程序中非常重要的一部分，它确保用户输入的命题公式符合命题逻辑的语法规则，避免错误的输入影响后续的计算。通过检查非法字符、运算符的合法性、括号的匹配、运算符使用正确幸福等，程序能够在早期发现并阻止不合法的输入，从而保证逻辑运算的正确性和稳定性。具体错误处理逻辑如下：

首先，程序会检查输入的逻辑表达式是否为空，若为空则会直接返回错误。

若表达式不为空，程序会初始化一些辅助变量，用来存储公式中的字符和逻辑状态：parentheses 栈，用于检查括号的配对情况。previous 变量：用于记录当前检查的字符之前的一个字符，以便做前后字符的合法性检查。

在此之后，程序通过遍历输入字符串中的每一个字符，执行以下逻辑：

1.检查字符是否合法：遍历公式中的每个字符，确保它是字母、合法的逻辑运算符（!, &, |, ^, ~），或者是括号 (和)。如果字符非法，提示错误并返回 false。

2.检查公式是否以运算符开始：如果公式的第一个字符是运算符（如 &, |, ~, ^），提示并返回 false。

3.检查是否存在空括号：如果出现空括号（例如 ()），则返回错误。

4.检查连续的取非运算符：如果公式中出现连续的 ！运算符（如 !!），则提示并返回 false。

5.检查运算符前后的连接是否正确：

！运算符前不能跟字母直接相连（如 A!B）。

运算符必须与变量或括号连接，不能与其他运算符或括号直接连接（如 A&&B 或 (A&)）。

6.检查括号的匹配性：通过栈结构，确保每个左括号 (都有对应的右括号)，并且括号配对正确。如果不匹配，则返回 false。

7.检查公式是否以运算符结束：如果公式以运算符结尾，提示并返回 false。

8.检查变量与括号的连接：确保变量与括号的连接正确（如 (A) 是合法的，A() 或 ()A 是不合法的）。

如果上述检查均通过，返回 true，表示公式合法；否则返回 false，并显示相应错误提示，此时需要用户重新输入逻辑表达式。

6.4.2 逻辑计算功能

逻辑计算功能的核心目标是根据用户输入的逻辑表达式和变量的真值，计算该表达式的结果。具体功能通过栈结构和运算符优先级的管理来实现，确保所有逻辑运算按照正确的顺序进行。

1. 操作符与操作数：

程序实现了多种基本的逻辑操作符，每种操作符的功能如下：

!（取非）：对单个操作数进行取反。

&（与）：当且仅当两个操作数都为真时，结果为真。

|（或）：只要其中一个操作数为真，结果为真。

^（条件）：如果第一个操作数为假或两个操作数都为真，结果为真。

~（双条件）：当两个操作数的值相等时，结果为真。

2. 计算过程：

在处理逻辑表达式时，程序使用栈（operators）来保存操作符，操作数则通过另一个栈（operands）来存储。

程序从左到右遍历逻辑表达式，遇到变量时，将其映射作为操作数推入操作数栈，当遇到左括号是，直接入操作符栈。

当遇到右括号时，程序开始处理栈，直到操作符栈为空或操作符栈顶为左括号，处理完成后，取出左括号；当遇到操作符时，程序开始处理栈，直到操作符栈为空或当前运算符优先级比操作符栈顶操作符优先级高，将当前操作符加入操作符栈。

在表达式遍历结束后，开始处理栈，直到操作符栈为空；此时操作数栈中应只剩下一个值，这个值即为逻辑表达式的最终计算结果。程序会从操作数栈中取出最终结果并返回。

3. 栈处理方法

从操作符栈中取出操作符并判断类型：

若为!,则取出操作数栈顶元素，进行计算，并将结果放入操作数栈中；

如果不是非，则从栈中取出两个元素，依次将其作为第二个运算数、第一个运算数，根据操作符进行不同计算，将计算结果放入操作数栈。

6.5 核心算法

6.5.1 真值表计算

真值表计算是生成真值表的基础。首先，通过 CountAndStoreVariable 函数遍历表达式并存储从表达式中提取唯一变量集合，存储在 store_variable 字符串中；然后通过 store_variable 字符串长度（变量个数）计算真值表行数 2^n，输出真值表表头。然后进入一个 for 循环计算真值表每一行。

在每一行中，为不同命题变量赋值，存储在 std::unordered_map<char, bool>values 中；输出赋值结果，使用逻辑计算功能计算当前赋值下表达式的真值，输出计算结果。即实现真值表一行的输出。

重复以上步骤 2^n 次，即可以实现真值表的输出。

赋值操作的实现：

注意到真值表中每一行的赋值与一个二进制数递增（每次加 1）的过程相似，那么，可以将循环变量 i 作为该二进制数，通过移位和按位与将其二进制表示的每一位对应赋值给命题变量，即使实现对命题的赋值。

6.5.2 主范式

主范式的计算可以在真值表计算的同时完成。在计算真值表时，通过两个动态数组，分别记录命题成真赋值和成假赋值。

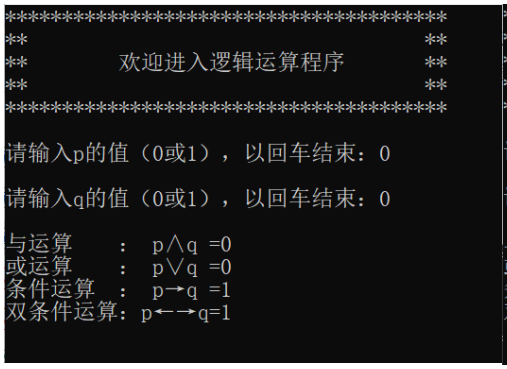
对于每个使表达式成假的赋值，通过析取将变量分别构建成子句，这些子句在逻辑上用合取（“与”）连接起来，形成主合取范式。

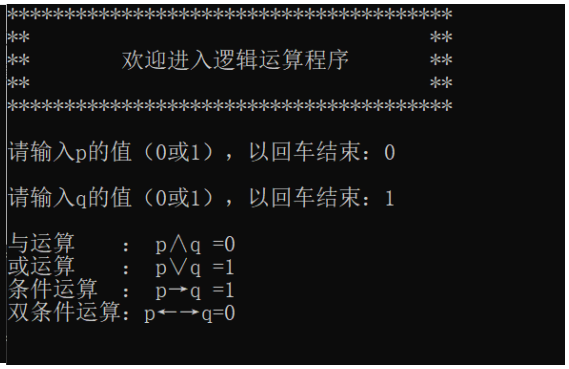
对于每个使表达式成真的赋值，通过和取将变量分别构建成子句，这些子句在逻辑上用析取（“或”）连接起来，形成主析取范式。

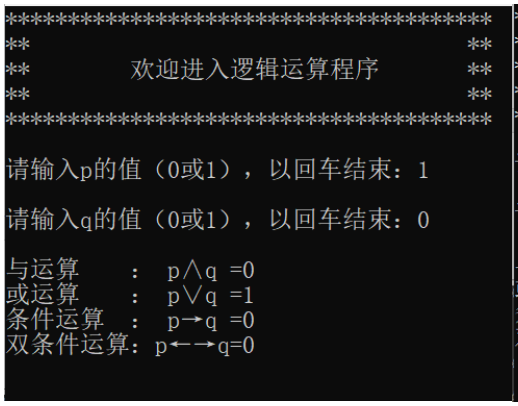
七 实验结果测试

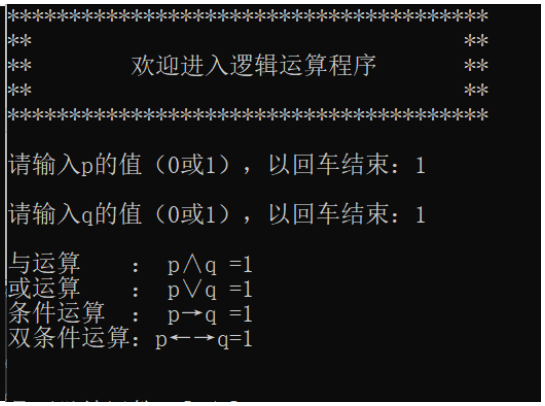
7.1 命题逻辑联接词

当输入正确的的 p、q 值时，程序可以正确地实现真值计算



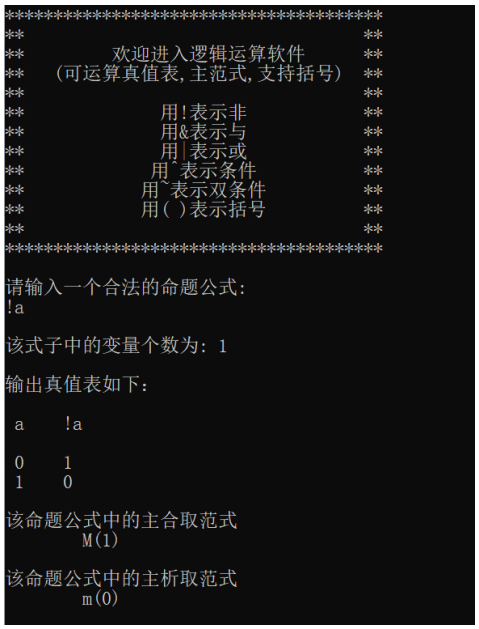


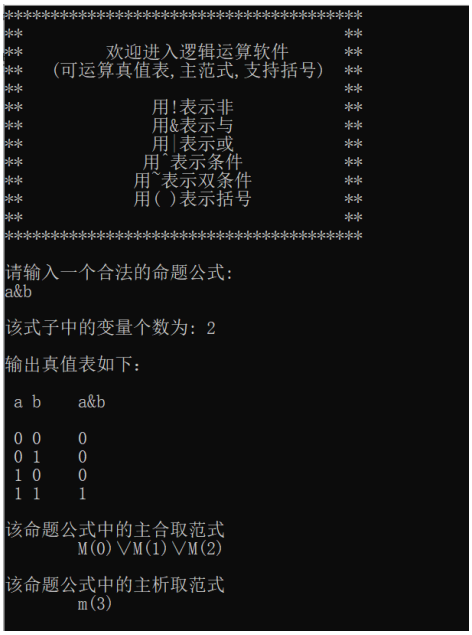




7.2 真值表、主范式

对于测试用表达式，程序可以正确的输出真值表和主范式。





```
请输入一个合法的命题公式:
a|b

该式子中的变量个数为: 2

输出真值表如下:

 a b     a|b

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

该命题公式中的主合取范式
        M(0)

该命题公式中的主析取范式
        m(1)∧m(2)∧m(3)
```

```
请输入一个合法的命题公式:
a^b

该式子中的变量个数为: 2

输出真值表如下:

 a b     a^b

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

该命题公式中的主合取范式
        M(2)

该命题公式中的主析取范式
        m(0)∧m(1)∧m(3)
```

```
请输入一个合法的命题公式:
a~b

该式子中的变量个数为: 2

输出真值表如下:

 a b     a~b

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

该命题公式中的主合取范式
        M(1)∨M(2)

该命题公式中的主析取范式
        m(0)∧m(3)
```

```
请输入一个合法的命题公式:
a^b~c|b&a^c

该式子中的变量个数为: 3

输出真值表如下:

 a b c     a^b~c|b&a^c

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

该命题公式中的主合取范式
        M(4)∨M(5)∨M(6)

该命题公式中的主析取范式
        m(0)∧m(1)∧m(2)∧m(3)∧m(7)
```

```
请输入一个合法的命题公式:
!a^b~(c|d~(!b&c)^!d)|a

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

该命题公式中的主合取范式
        M(1)∨M(2)∨M(4)

该命题公式中的主析取范式
        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)
```

对于常见输入非法，程序可以给出警告：

```
请输入一个合法的命题公式:
a*|*
命题公式存在非法字符输入，请重新输入！
请输入一个合法的命题公式:
a|()|c
命题公式存在空括号，请重新输入！
请输入一个合法的命题公式:
a!b|c
命题公式中取非运算符前不可连接变量，请重新输入！
请输入一个合法的命题公式:
a(c|d)
命题公式中变量与括号的连接不正确，请重新输入！
请输入一个合法的命题公式:
a^(|d)
命题公式中每个二元运算符前后必须连接变量，请重新输入！
请输入一个合法的命题公式:
&s
命题公式不能以二元运算符开始，请重新输入！
请输入一个合法的命题公式:
(a|b^c
命题公式括号不匹配，请重新输入！
请输入一个合法的命题公式:
a^
命题公式不能以运算符结尾，请重新输入！
请输入一个合法的命题公式:
1|0
命题公式存在非法字符输入，请重新输入！
请输入一个合法的命题公式:
!!a
命题公式存在不合法的连续取非操作，请重新输入！
请输入一个合法的命题公式:
a&c&&v
命题公式中每个二元运算符前后必须连接变量，请重新输入！
请输入一个合法的命题公式:
ab|c
命题公式仅适用于单字符变量，不适用于多字符变量，请重新输入！
```

八 实验心得

在第一个程序中，我实现了一个简单的命题逻辑计算器，支持计算基本的逻辑运算（与、或、蕴涵和双蕴涵）。用户可以通过输入两个命题的真值（0 或 1），程序计算并输出相应的逻辑运算结果。此程序的核心功能是通过命令行交互让用户输入命题的真值，然后利用布尔运算符（&&, ||, !）进行逻辑计算，并输出结果。

通过这个程序，我对如何处理用户输入和如何验证输入的合法性有了更深刻的理解。为了避免用户输入错误，程序在输入时进行了合法性检查，确保用户输入的值必须是 0 或 1，并且程序通过 std::cin 读取数据时，还加了异常处理来保证输入正确。此外，整个程序还包含了一个清屏功能，用于每次计算完后清理界面，提升用户体验。

这个程序的设计思路简单明了，核心是运算符的布尔运算实现，但让我特别深刻的是如何在简单的逻辑表达式中结合输入验证和循环控制。通过这部分的编程练习，我提升了对布尔运算、输入输出操作以及错误处理的掌握，尤其是对程序用户交互的设计有了更多的思考和实践。

第二个程序则是一个更为复杂的命题逻辑计算器，涉及到逻辑表达式的解析、真值表的生成以及主合取范式和主析取范式的提取。这个程序不仅支持逻辑运算符（与、或、非、蕴涵、双蕴涵），还能够处理带括号的复杂逻辑表达式，并生成相应的真值表及其主范式。

在这个程序中，首先我实现了对用户输入逻辑表达式的合法性检查，确保表达式符合逻辑语法规则。然后，通过栈结构来实现运算符优先级的处理，从而正确地解析表达式中的每个部分并计算出真值。此外，程序还能够自动生成真值表，并根据真值表的输出提取主合取范式和主析取范式。主合取范式和主析取范式的提取是通过遍历真值表中成真赋值和成假赋值结果，分别生成对应的合取范式和析取范式。

通过这个程序，我进一步加深了对栈结构的理解，尤其是在处理中缀表达式时，栈的应用帮助我实现了对运算符优先级的管理。同时，真值表的生成让我更好地理解了命题逻辑的真值计算，如何通过不同的变量组合计算出表达式的最终结果。此外，主范式的生成让我对逻辑公式的标准化和简化有了更直观的认识，尤其是在命题逻辑的应用中，范式化是简化和分析逻辑表达式的重要工具。

这个程序比第一个程序更加复杂，它要求我深入理解如何解析逻辑表达式、如何运用栈结构来处理运算符优先级以及如何通过真值表计算主范式。通过这个项目，我不仅学到了如何处理复杂的逻辑公式，还进一步提高了自己在复杂逻辑推理和编程实现方面的能力。

总体而言，两个程序虽然功能有所不同，但都涉及到命题逻辑的计算与处理。在第一个程序中，我主要学习了如何进行简单的布尔运算并验证输入的合法性，而在第二个程序中，我通过实现更复杂的逻辑解析、优先级管理、真值表生成和主范式提取，深刻理解了逻辑表达式的解析过程。通过这两个程序的实现，我不仅提高了自己的编程能力，还加深了对命题逻辑和计算机科学中相关算法的理解。

附录 · 源码

命题逻辑联接词

```
#include <conio.h>
#include <cstdlib>
#include <iostream>
#include <limits>


bool Input(const char ch)
{
    double input;
    while (true) {
        std::cout << "请输入"<<ch<<"的值（0 或 1），以回车结束：";
        std::cin >> input; //读取 P 的值
        std::cout << "\n";
        if (std::cin.fail()||input!=0&&input!=1) {
            std::cout <<ch<<"值输入有误，请重新输入\n";
            // 清除缓冲区
            std::cin.clear();
            std::cin.ignore(std::numeric_limits<std::streamsize>::max(), '\n');
            continue;
        }
        std::cin.ignore(std::numeric_limits<std::streamsize>::max(), '\n');
        break;
    }

    return static_cast<bool>(input);
}

/*************************************************************
* Function Name:   main
* Function:        输入两个命题的真值，并计算由与、或、蕴涵和双蕴涵构成的命题的真值
* Input Parameter:
* Returned Value:  0
*************************************************************/
int main()
{
    while (true) {
        system("cls");//清屏
```

```
        std::cout << ("**************************************\n") //标语
                  << ("**                                  **\n")
                  << ("**        欢迎进入逻辑运算程序        **\n")
                  << ("**                                  **\n")
                  << ("**************************************\n\n");

        bool p = Input('p');
        bool q = Input('q');
        std::cout << "与运算    : " << " p∧q =" << (p && q) << "\n";//与运算同时输出结果
        std::cout << "或运算    : " << " p∨q =" << (p || q) << "\n"; //或运算同时输出结果
        std::cout << "条件运算  : " << " p→q =" << (!p || q) << "\n"; //蕴涵运算，将其转化为与或非形式同时输出结果
        std::cout << "双条件运算: " << "p←→q=" << ((!p || q) && (p || !q)) << "\n\n\n";//等价运算，将其转化为与或非形式同时输出结果

        std::cout << "是否继续运算？[Y/n] ";
        char ch;
        do {
            ch = static_cast<char>(_getch());
        } while (ch != 'Y' && ch != 'y' && ch != 'N' && ch != 'n');
        std::cout << ch << '\n';

        // 如果用户输入 N 或 n，则退出循环，结束程序
        if (ch == 'N' || ch == 'n')
            break;
    }
    std::cout << "欢迎下次再次使用!\n";

    return 0;
}
```

真值表、主范式

```
#include <cctype>
#include <cmath>
#include <conio.h>
#include <cstdlib>
#include <iostream>
#include <limits>
#include <stack>
#include <string>
```

```
#include <unordered_map>
#include <vector>
#include <cstdlib>
#include <stack>

bool AND(const bool a, const bool b) { return a && b; } //否定
bool OR(const bool a, const bool b) { return a || b; } //合取
bool NOT(const bool a) { return !a; } //析取
bool IMPLIES(const bool a, const bool b) { return !a || b; } //蕴涵
bool EQUIVALENT(const bool a, const bool b) { return IMPLIES(a, b) && IMPLIES(b,
a); } //等价

/*************************************************************
* Function Name:   getPriority
* Function:        获取运算符的优先级
* Input Parameter: op - 运算符
* Returned Value:  运算符的优先级
*************************************************************/
int getPriority(char op)
{
    if (op == '!') return 5;
    if (op == '&') return 4;
    if (op == '|') return 3;
    if (op == '^') return 2;
    if (op == '~') return 1;

    return 0; // 无优先级
}

/*************************************************************
* Function Name:   LegalOperator
* Function:        检查表达式中非字母和括号的字符是否合法
* Input Parameter: character: 需要检查的字符
* Returned Value:  如果字符合法，返回 true；否则返回 false
*************************************************************/
bool LegalOperator(const char ch)
{
    //判断是否为逻辑运算符
    if (ch == '!' || ch == '&' || ch == '|' || ch == '^' || ch == '~')
        return true;
    return false;
}

/*************************************************************
```

```
* Function Name:   Calculate
* Function:        通过判断运算符类型，调用不同计算函数
* Input Parameter: a 第一个运算数
*                  op 运算符
*                  b 第二个运算数
* Returned Value:  运算值
*************************************************************/
bool Calculate(const bool a, const char op ,bool b)
{
    if (op == '^')
        return IMPLIES(a, b);
    if (op == '~')
        return EQUIVALENT(a, b);
    if (op == '&')
        return AND(a, b);
    if (op == '|')
        return OR(a, b);
    std::cerr << "Error! Operator '" << op << "' is not a valid binary connective."
<< std::endl;
    abort();
}



/*************************************************************
* Function Name:   Evaluate
* Function:        逻辑表达式的真值
* Input Parameter: expression: 逻辑表达式字符串
*                  values: 变量的真值映射
* Returned Value:  计算后的逻辑表达式的真值
*************************************************************/
void  ProcessStack(std::stack<char>& operators,std::stack<bool>& operands)
{
    //取出运算符
    const char op = operators.top();
    operators.pop();
    if (op == '!') {
        const bool operand = operands.top();
        operands.pop();
        const bool result = NOT(operand);
        //将运算结果放入栈中
        operands.push(result);
    }
    else {
        //取出栈中的两个运算数，后入栈作为第二个运算数（蕴含不满足交换律）
```

```
        const bool b = operands.top();
        operands.pop();
        const bool a = operands.top();
        operands.pop();
        const bool result = Calculate(a, op, b);
        //将运算结果放回运算数栈中
        operands.push(result);
    }
}

/*************************************************************
* Function Name:   Evaluate
* Function:        逻辑表达式的真值
* Input Parameter: expression: 逻辑表达式字符串
*                  values: 变量的真值映射
* Returned Value:  计算后的逻辑表达式的真值
*************************************************************/
bool Evaluate(const std::string& expression, std::unordered_map<char, bool>&
values)
{
    std::stack<char> operators;
    std::stack<bool> operands;
    for (char ch : expression) {
        if (isalpha(ch))
            operands.push(values[ch]);
        else if (ch == '(')
            operators.push(ch);
        else if (ch == ')') {
            //处理括号中的内容，直到遇到左括号或运算符栈为空
            while (!operators.empty() && operators.top() != '(') {
                ProcessStack(operators, operands);
            }
            //取出左括号
            if (!operators.empty()) operators.pop();
        }
        else if (LegalOperator(ch)) {
            //取出比当前运算符优先级高或与当前运算符优先级相同（蕴含不满足结合律）
的运算符进行计算
            while (!operators.empty() &&
getPriority(operators.top())>=getPriority(ch)) {
                ProcessStack(operators, operands);
            }
            operators.push(ch);
        }
```

```
    }

    while (!operators.empty()) {
        ProcessStack(operators, operands);
    }

    //当运算符栈为空时，运算数栈只剩一个数，即为结果
    const bool result = operands.top();
    operands.pop();
    return result;
}

/*************************************************************
* Function Name:   CountAndStoreVariable
* Function:        统计并存储逻辑表达式中的变量的种类及个数
* Input Parameter: initial_logical_expression: 初始逻辑表达式
*                  number_of_variables: 变量个数的引用
*                  store_variable: 存储变量的字符串引用
* Returned Value:
*************************************************************/
void CountAndStoreVariable(const std::string &initial_logical_expression,int&
number_of_variables, std::string& store_variable) {
    for (const auto ch:initial_logical_expression) {
       if (isalpha(ch)) {
          if (store_variable.find(ch)==std::string::npos) {
             store_variable += ch;
             number_of_variables++;
          }
       }
    }
}

/*************************************************************
* Function Name:   TruthTableNormalForm
* Function:        输出逻辑表达式的真值表和主范式
* Input Parameter: initial_logical_expression: 初始逻辑表达式
* Returned Value:  无
*************************************************************/
void TruthTableNormalForm(const std::string& initial_logical_expression)
{
    int number_of_variables = 0;
    std::string store_variable;
    CountAndStoreVariable(initial_logical_expression,number_of_variables,
store_variable);
```

```
    std::cout << "\n 该式子中的变量个数为: " << number_of_variables << "\n\n";
    const int truth_table_rows = static_cast<int>(std::pow(2,
number_of_variables)); //计算取值情况的个数
    std::vector<int> conjunction, disjunction;

    //输出真值表
    std::cout << "输出真值表如下: \n\n";
    for (const char ch : store_variable)
        std::cout << " " << ch;
    std::cout << "    " << initial_logical_expression << "\n\n";
    for (int i = 0; i < truth_table_rows; i++) {
        //采用映射的方式, 方便计算
        std::unordered_map<char, bool>values;
        // 设置变量的真值
        for (int j = 0; j < number_of_variables; j++)
            values[store_variable[j]] = (i >> (number_of_variables - j - 1)) & 1;//
通过移位和按位与实现为变量赋值
        // 输出变量的真值
        for (char ch : store_variable)
            std::cout << " " << values[ch];

        //计算逻辑表达式的真值
        const bool result = Evaluate(initial_logical_expression, values);
        if (result)
            disjunction.push_back(i);
        else
            conjunction.push_back(i);
        // 输出逻辑表达式的真值
        std::cout << "    " << static_cast<int>(result) << "\n";
    }

    // 输出主合取范式
    std::cout << "\n 该命题公式中的主合取范式\n        ";
    if (!conjunction.empty()) {
        for (size_t i = 0; i < conjunction.size(); i++) {
            if (i != 0)
                std::cout << "∨";
            std::cout << "M(" << conjunction[i] << ")";
        }
    }
    else
        std::cout << "该命题公式不存在主合取范式\n";

    // 输出主析取范式
```

```
    std::cout << "\n\n 该命题公式中的主析取范式\n        ";
    if (!disjunction.empty()) {
        for (size_t i = 0; i < disjunction.size(); i++) {
            if (i != 0)
                std::cout << "∧";
            std::cout << "m(" << disjunction[i] << ")";
        }
    }
    else
        std::cout << "该命题公式不存在主析取范式\n";
}

// 检查逻辑表达式是否为空
/*************************************************************
* Function Name:   LegalExpression
* Function:        检查逻辑表达式是否合法
* Input Parameter: initial_logical_expression: 逻辑表达式
* Returned Value:  如果合法返回 true; 否则返回 false
*************************************************************/
bool LegalExpression(const std::string& initial_logical_expression)
{
    // 检查逻辑表达式是否为空
    if (initial_logical_expression.empty()) {
        std::cout << "命题公式为空, 请重新输入! \n";
        return false;
    }

    std::stack<char> parentheses;// 用于存储括号
    char previous = '\0';// 存储前一个字符
    for (char ch : initial_logical_expression) {
        // 检查字符是否合法
        if (!(isalpha(ch)) && !LegalOperator(ch) && ch != '(' && ch != ')') {
            std::cout << "命题公式存在非法字符输入, 请重新输入! \n";
            return false;
        }

        // 检查公式是否以二元运算符开始
        if (previous == '\0' && (ch == '&' || ch == '|' || ch == '~' || ch == '^'))
{
            std::cout << "命题公式不能以二元运算符开始, 请重新输入! \n";
            return false;
        }

        // 检查空括号
```

```
        if (previous == '(' && ch == ')') {
            std::cout << "命题公式存在空括号，请重新输入！\n";
            return false;
        }

        // 检查不合法的连续取非操作
        if (previous == '!' && ch == '!') {
            std::cout << "命题公式存在不合法的连续取非操作，请重新输入！\n";
            return false;
        }

        // 检查取非运算符前的字符是否合法
        if (ch == '!' && isalpha(previous)) {
            std::cout << "命题公式中取非运算符前不可连接变量，请重新输入！\n";
            return false;
        }

        // 检查变量与括号的连接
        if ((isalpha(ch) && previous == ')') || (ch == '(' && isalpha(previous))) {
            std::cout << "命题公式中变量与括号的连接不正确，请重新输入！\n";
            return false;
        }

        // 存储左括号
        if (ch == '(')
            parentheses.push(ch);

        // 处理右括号
        else if (ch == ')') {
            if (parentheses.empty()) {
                std::cout << "命题公式括号不匹配，请重新输入！\n";
                return false;// 右括号没有匹配的左括号
            }
            parentheses.pop();// 匹配到一对括号，弹出栈顶元素
        }

        // 检查多字符变量
        if (isalpha(ch) && isalpha(previous)) {
            std::cout << "命题公式仅适用于单字符变量，不适用于多字符变量，请重新输
入！\n" ;
            return false;
        }

        // 检查运算符前后的连接
```

```
        if ((ch == '&' || ch == '|' || ch == '~' || ch == '^') &&
(!isalpha(previous) && previous != ')')) {
            std::cout << "命题公式中每个二元运算符前后必须连接变量，请重新输入!
\n";
            return false;
        }
        previous = ch;
    }

    // 检查公式是否以运算符结尾
    if (LegalOperator(previous)) {
        std::cout << "命题公式不能以运算符结尾，请重新输入! \n";
        return false;
    }

    // 检查括号是否匹配
    if (!parentheses.empty()) {
        std::cout << "命题公式括号不匹配，请重新输入! \n";
        return false;
    }

    return true;// 表达式合法
}

int main()
{
    while (true) {
        system("cls");//清屏
       std::string initial_logical_expression, store_variable;
        std::cout << "**************************************\n"
                  << "**                                  **\n"
                  << "**          欢迎进入逻辑运算软件      **\n"
                  << "**    (可运算真值表,主范式,支持括号)  **\n"
                  << "**                                  **\n"
                  << "**              用!表示非           **\n"
                  << "**              用&表示与           **\n"
                  << "**              用|表示或           **\n"
                  << "**              用^表示条件         **\n"
                  << "**             用~表示双向条件      **\n"
                  << "**              用( )表示括号       **\n"
                  << "**                                  **\n"
                  << "**************************************\n\n";
        do {
            std::cout << "请输入一个合法的命题公式:\n";//输入式子
```

```
            std::getline(std::cin, initial_logical_expression);//读取式子
        } while (!LegalExpression(initial_logical_expression));//判断式子是否符合规
范
        TruthTableNormalForm(initial_logical_expression);//调用真值表和主范式功能函
数
        std::cout << "\n\n 是否继续运算？[Y/n] ";

        char ch;
        do {
            ch = static_cast<char>(_getch());
        } while (ch != 'Y' && ch != 'y' && ch != 'N' && ch != 'n');
        std::cout << ch << '\n';

        // 如果用户输入 N 或 n，则退出循环，结束程序
        if (ch == 'N' || ch == 'n')
            break;
    }
    std::cout << "\n\n 欢迎下次使用！ \n";

    return 0;
}
```