《离散数学》课程实验报告文档

题目：命题逻辑联接词、真值表、主范式

姓名：　赵卓冰

学号：　2252750

专业：　软件工程

年级：　2023 级

指导教师：唐剑锋

2024 年 10 月 21 日

1. 开发环境与编译运行环境

本项目可以在不同的开发环境和编译运行环境上运行

- Window操作系统：
  - 版本：Windows 10 x64
  - IDE: Visual Studio 2022 (Debug模式)
  - 编译器： MSVC 14.39.33519
- Linux操作系统：
  - 版本：Ubuntu 20.04.6 LTS
  - IDE： VS Code
  - 编译器：gcc version 9.4.0 (Ubuntu 9.4.0-1ubuntu1~20.04.2)

2. 实验内容1： 命题逻辑联接词的实现

2.1. 实验目的

本实验的主要目的是让掌握命题逻辑中的常见联接词（合取、析取、条件、双条件）的运算规则，通过C++实现这些运算。通过对命题变元P和Q的真值组合进行计算，理解和掌握命题逻辑的操作。

2.2. 实验简介

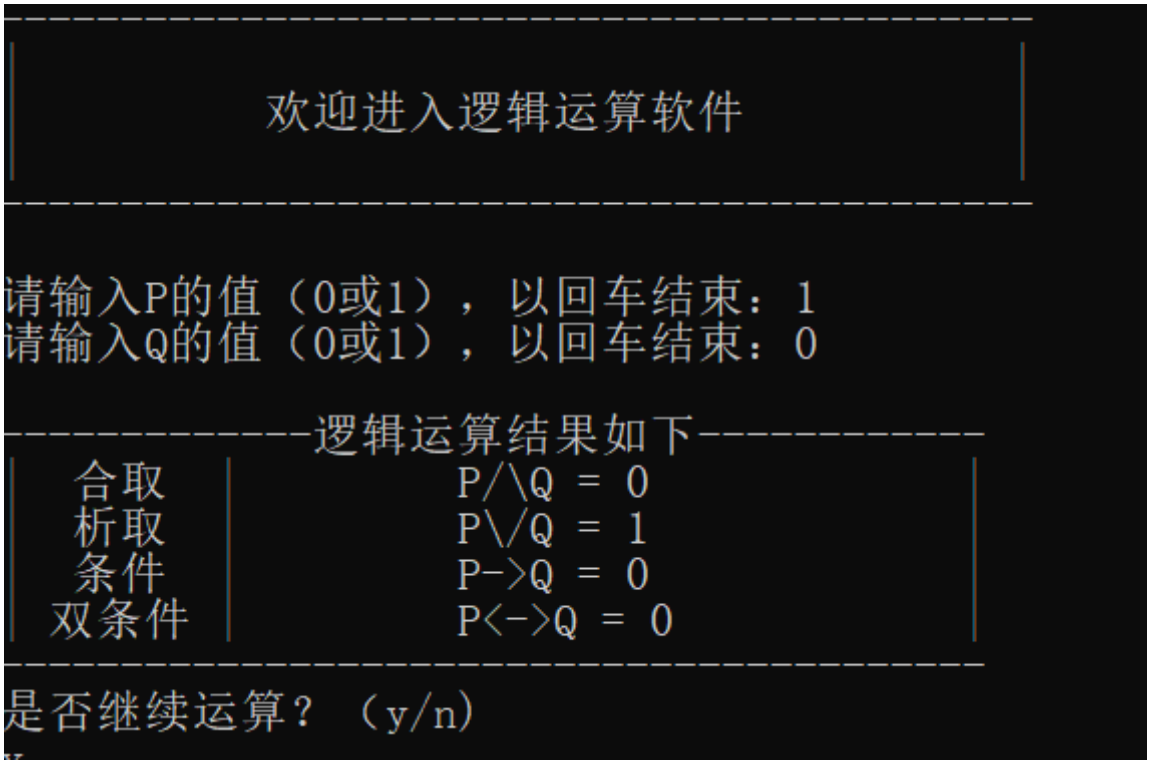
实验任务要求我们从键盘输入两个命题变元P和Q的真值，然后计算并输出它们在以下逻辑运算下的真值结果：

- **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）**： 当P和Q至少有一个为真时，结果为真；
- **条件（Implication, P → Q）**： 当P为真且Q为假时，结果为假，其他情况为真；
- **双条件（Biconditional, P ↔ Q）**： 当P和Q的真值相同（都为真或都为假）时，结果为真。

我们通过C++程序实现了以上运算的功能，用户可以通过多次输入命题变元的值来测试不同的逻辑运算结果。

2.3. 界面展示

2.3.1. windows界面展示



2.3.2. linux界面展示

2.4. 实验思路

1. **程序结构**:
    - 程序通过一个循环实现多次运行，用户可以在输入完P和Q的值后，选择是否继续。

- 通过封装函数的方式，模块化地处理每个步骤，如输入命题变元值、逻辑运算及结果输出。

2. **输入部分**：
   - 采用了`varInput()`函数，用户可以输入命题变元P和Q的真值（只能为0或1）。
   - 通过输入检查，保证输入的有效性，并通过`cin.clear()`和`cin.ignore()`处理错误输入。
3. **逻辑运算部分**：
   - 利用C++的位操作符和逻辑运算符实现合取（`&`）、析取（`|`）、条件（`!p | q`）、双条件（`p == q`）四种运算。
   - 这些运算分别对应命题逻辑中的常见联接词。
4. **输出部分**：
   - 通过`ResPrint()`函数将逻辑运算结果以表格的形式输出，便于用户理解。
5. **重复执行**：
   - 程序使用了一个循环，允许用户选择继续进行新的逻辑运算，直到用户输入'n'或'N'退出程序。

2.5. 所用数据结构

实验中使用的主要是基本的数据类型`int`来表示命题变元P和Q的真值，真值只能为0或1。没有使用复杂的数据结构，因为命题逻辑的基本运算涉及的只是布尔型真值（True/False）。

2.6. 核心算法

1. **合取（P ∧ Q）**：
   - 使用位运算符`&`来实现P和Q的合取运算。
   - 逻辑：只有当P和Q都为1时，结果为1；否则结果为0。

```
p & q;
```

2. **析取（P ∨ Q）**：
   - 使用位运算符`|`来实现P和Q的析取运算。
   - 逻辑：只要P或Q中有一个为1，结果为1；只有当P和Q都为0时，结果为0。

```
p | q;
```

3. **条件（P → Q）**：
   - 使用逻辑表达式`!p | q`来实现P和Q的条件运算。
   - 逻辑：当P为1且Q为0时，结果为0；其他情况为1。

```
!p | q;
```

4. **双条件（P ↔ Q）**：
   - 使用比较运算符`==`来实现P和Q的双条件运算。
   - 逻辑：当P和Q的真值相同时（都为1或都为0），结果为1；否则结果为0。

```
p == q;
```

2.7. 实验心得

通过本次实验，我加深了对命题逻辑联接词的理解。虽然逻辑运算在数学上比较简单，但在程序设计中，我们需要充分利用语言本身的运算符来简化逻辑计算过程。通过此次实验，我学会了如何设计一个模块化的程序，利用函数封装处理输入、逻辑运算和输出，增强了代码的可读性和可扩展性。

此外，实验中对于输入检查和容错处理也让我意识到，编写一个稳定且用户友好的程序不仅仅需要关注核心功能的实现，还要考虑到各种边界条件和异常情况的处理。

2.8. 代码展示

```
#include <iostream>
using namespace std;

// 打印标题
void TitlePrint() {
  cout << "-----------------------------------------" <<
endl;
  cout << "|                                       |" <<
endl;
  cout << "|          欢迎进入逻辑运算软件          |" << endl;
  cout << "|                                       |" <<
endl;
  cout << "-----------------------------------------" <<
endl <<endl;
```

```cpp
}

// 输入变量
void VarInput(int& p, int& q) {
  cout << "请输入P的值（0或1），以回车结束：";
  cin >> p;
  while (cin.fail() || (p != 0 && p != 1)) {
    cout << endl;
    cout << "P的值输入有误，请重新输入" << endl;;
    cout << "请输入P的值（0或1），以回车结束：";
    cin.clear();
    cin.ignore(65535, '\n');
    cin >> p;
  }

  cout << "请输入Q的值（0或1），以回车结束：";
  cin >> q;
  while (cin.fail() || (q != 0 && q != 1)) {
    cout << endl;
    cout << "Q的值输入有误，请重新输入" << endl;
    cout << "请输入Q的值（0或1），以回车结束：";
    cin.clear();
    cin.ignore(65535, '\n');
    cin >> q;
  }
}

// 输出结果
void ResPrint(const int& p, const int& q) {
  printf("\n-------------逻辑运算结果如下------------\n");
  printf("|  合取  |         P/\\Q = %d             |\n", p &
q);
  printf("|  析取  |         P\\/Q = %d             |\n", p |
q);
  printf("|  条件  |         P->Q = %d             |\n", !p |
q);
  printf("| 双条件 |         P<->Q = %d            |\n", p ==
q);
  printf("----------------------------------------");
}

// 主函数
int main() {
  while (1) {
    TitlePrint();
```

```
52      int p, q;
53      VarInput(p, q);
54      ResPrint(p, q);
55
56      cout << endl << "是否继续运算？（y/n)" << endl;
57      char ch;
58      cin >> ch;
59      if (ch == 'n' || ch == 'N') {
60        break;
61      }
62    }
63    cout << "欢迎下次使用!" << endl;
64
65    return 0;
66  }
```

2.9. 总结

通过本次实验的实现，了解了命题逻辑的基本运算在计算机中的实际应用，并巩固了程序设计的思路和方法。下一步的实验内容将进一步深入逻辑公式的真值表和主范式的求解。

3. 实验内容2：逻辑表达式真值表和主范式的实现

3.1. 实验目的

计算复杂逻辑表达式的真值表和求取主范式。通过对含有多个逻辑变元的表达式进行解析，理解如何实现逻辑运算的自动化，并体会逻辑推理在编程中的具体实现方法。

3.2. 实验简介

在本实验中，实现了以下几项主要功能：

1. **逻辑表达式的解析与计算**：输入一个逻辑表达式，程序会自动分析表达式结构并进行计算。
2. **真值表的生成**：程序自动生成逻辑表达式的真值表，展示所有可能的输入组合及其对应的输出值。
3. **主范式的求取**：基于真值表，程序会生成逻辑表达式的主析取范式（DNF）和主合取范式（CNF）。

用户可以输入包含逻辑运算符和括号的复杂逻辑表达式，运算符包括：

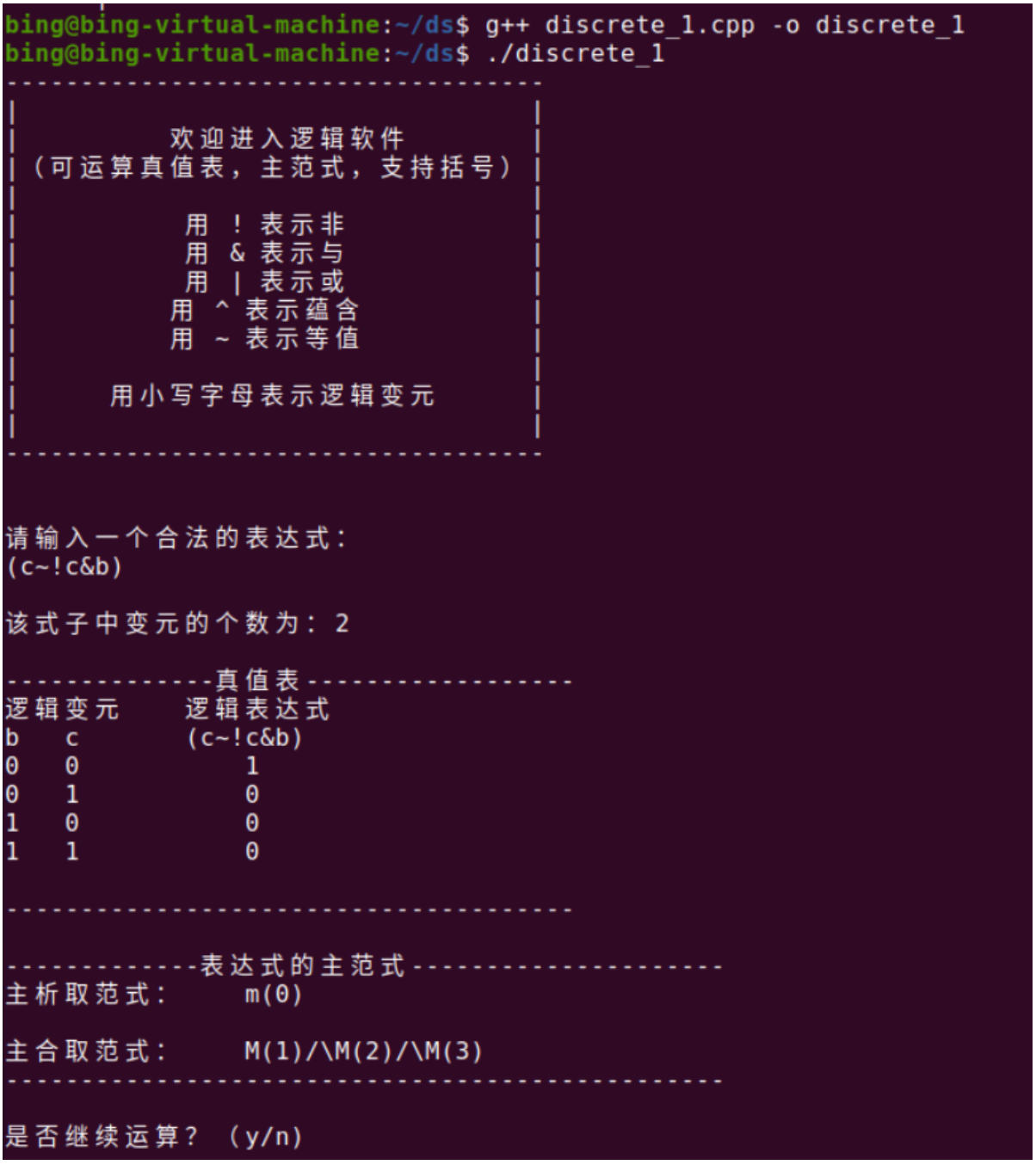
- **非（NOT, !）**
- **合取（AND, &）**
- **析取（OR, |）**
- **条件（Implication, ^）**
- **双条件（Biconditional, ~）**

3.3. 界面展示

3.3.1. windows界面展示

```
------------------------------------
|                                  |
|          欢迎进入逻辑软件        |
|  （可运算真值表，主范式，支持括号）|
|                                  |
|            用 ！ 表示非          |
|            用 & 表示与           |
|            用 | 表示或           |
|           用 ^ 表示蕴含          |
|           用 ~ 表示等值          |
|                                  |
|       用小写字母表示逻辑变元     |
|                                  |
------------------------------------

请输入一个合法的表达式：
a | b

该式子中变元的个数为：2

--------------真值表-----------------
逻辑变元      逻辑表达式
a    b        a | b
0    0          0
0    1          1
1    0          1
1    1          1

----------------------------------------


-------------表达式的主范式--------------------
主析取范式：      m(1)\/m(2)\/m(3)

主合取范式：      M(0)
-----------------------------------------------

是否继续运算？（y/n）
```

3.3.2. linux界面展示



3.4. 实验思路

1. **程序结构设计**：
   - 通过输入解析、逻辑变元提取、表达式计算、真值表生成及主范式求取等模块实现逻辑表达式的完整计算流程。
   - 用户输入表达式后，程序逐步解析表达式中的逻辑变元及运算符，最终生成计算结果。
2. **逻辑表达式的解析与运算**：
   - **栈操作**：采用栈数据结构实现对表达式的计算，类似于逆波兰表达式求值的方法，支持括号的优先级处理。

- **运算符优先级**：根据逻辑运算符的优先级（! > & > | > ^ > ~），程序依次进行计算。使用函数来处理栈顶操作符，保证各运算符按照优先级顺序正确执行。

3. **真值表生成**：
   - 首先提取表达式中的所有逻辑变元，并将它们按照字母顺序排序。
   - 根据逻辑变元的数量，生成所有可能的布尔值组合（即2^n种情况），并计算出每种组合下表达式的值。
4. **主范式的求取**：
   - **主析取范式（DNF）**：根据真值表中所有为真的组合生成。
   - **主合取范式（CNF）**：根据真值表中所有为假的组合生成。

3.5. 核心算法与实现

1. **逻辑表达式的求值**：
   - 使用两个栈来实现表达式求值，一个存储操作数，另一个存储操作符。
   - 通过遍历表达式中的每个字符，对不同类型字符（左括号、右括号、逻辑变元、运算符）进行不同的处理：
     - 如果是**左括号**，直接压入操作符栈。
     - 如果是**右括号**，不断从操作符栈中弹出操作符并计算，直到遇到左括号为止。
     - 如果是**逻辑变元**，将其对应的布尔值压入操作数栈。
     - 如果是**运算符**，则根据其优先级处理栈顶操作符。
2. **真值表的生成**：
   - 根据逻辑变元的数量，通过位运算生成所有可能的输入组合，每个组合对应一行真值表。
   - 对每一行输入组合，使用之前实现的表达式求值函数，得到相应的输出值。
3. **主范式的生成**：
   - **主析取范式**：当表达式在某一组合下为真时，记录该组合的对应极小项，并将其用析取符号连接起来。
   - **主合取范式**：当表达式在某一组合下为假时，记录该组合的对应极大项，并将其用合取符号连接起来。

3.6. 表达式合法性检验

3.6.1. 概述

使用 `ExpressionInput()` 函数来验证用户输入的表达式是否是合法的逻辑表达式。函数通过一系列步骤确保表达式的结构和字符符合规定，并在发现错误时提示用户重新输入。最终，当表达式符合要求时，函数返回合法的表达式。

3.6.2. 函数功能

`ExpressionInput()` 函数负责从用户输入中获取一个逻辑表达式并对其进行格式化和合法性检查。其主要功能包括：

1. **移除空格**：去除表达式中的所有空格，确保后续处理时的表达式纯净。
2. **括号匹配检验**：确保括号成对出现且位置正确。
3. **字符和结构检查**：验证表达式中的字符是否合法（包括变元和运算符），以及符号的排列是否符合逻辑表达式的语法规则。
4. **错误提示**：在表达式不合法时输出错误信息，提示用户重新输入。

3.6.3. 主要步骤

3.6.3.1. 1. 移除空格

函数首先使用 `erase` 和 `remove` 函数将输入的表达式中的空格移除，以方便后续处理表达式的字符。这个操作确保用户输入时的多余空格不会影响表达式验证。

3.6.3.2. 2. 括号匹配

使用一个栈（`stack<char>`）来检查括号是否匹配。具体规则如下：

- 当遇到左括号 `'('` 时，将其压入栈中。
- 当遇到右括号 `')'` 时，检查栈是否为空，如果为空，则说明右括号多于左括号，输出错误提示。
- 遇到右括号时，栈不为空则弹出栈顶的左括号，继续匹配。
- 如果表达式处理完毕后栈中仍有左括号，则提示左括号多于右括号。

3.6.3.3. 3. 表达式结构检查

接下来，逐个检查表达式的字符，确保其合法性。主要规则如下：

- **变元**：期望变元出现在某些位置，如运算符或左括号之后。如果期望的位置不出现变元，输出错误提示。

- **运算符**：运算符需要出现在合法的位置，如两个变元之间，或右括号和变元之间。单目运算符 `!` 是例外，它可以在变元或左括号前。否则输出错误提示。
- **括号**：左括号应该出现在某些期望的位置，右括号则需要在变元或另一个右括号之后。

在表达式的末尾，函数检查最后一个字符，如果它是运算符则视为非法，提示用户重新输入。

3.6.3.4. 4. 返回合法表达式

当表达式通过了所有的检查，函数将返回经过验证的合法表达式。

3.6.4. 错误提示

在输入过程中，函数会对各种错误情况给出详细的提示，包括：

- **括号不匹配**：提示左括号或右括号的数量不匹配。
- **非法字符**：提示表达式中包含非法字符。
- **运算符位置错误**：提示运算符前后缺少应有的变元或括号。
- **表达式结尾错误**：提示表达式不能以运算符结尾。

3.6.5. 示例

```
请输入一个合法的表达式：
a + (b * c)
```

如上输入为一个合法的表达式，函数将返回 `"a+(b*c)"`。

```
请输入一个合法的表达式：
a + b * )
错误：右括号多于左括号。
```

如上输入中，右括号不匹配，函数将提示错误并要求重新输入。

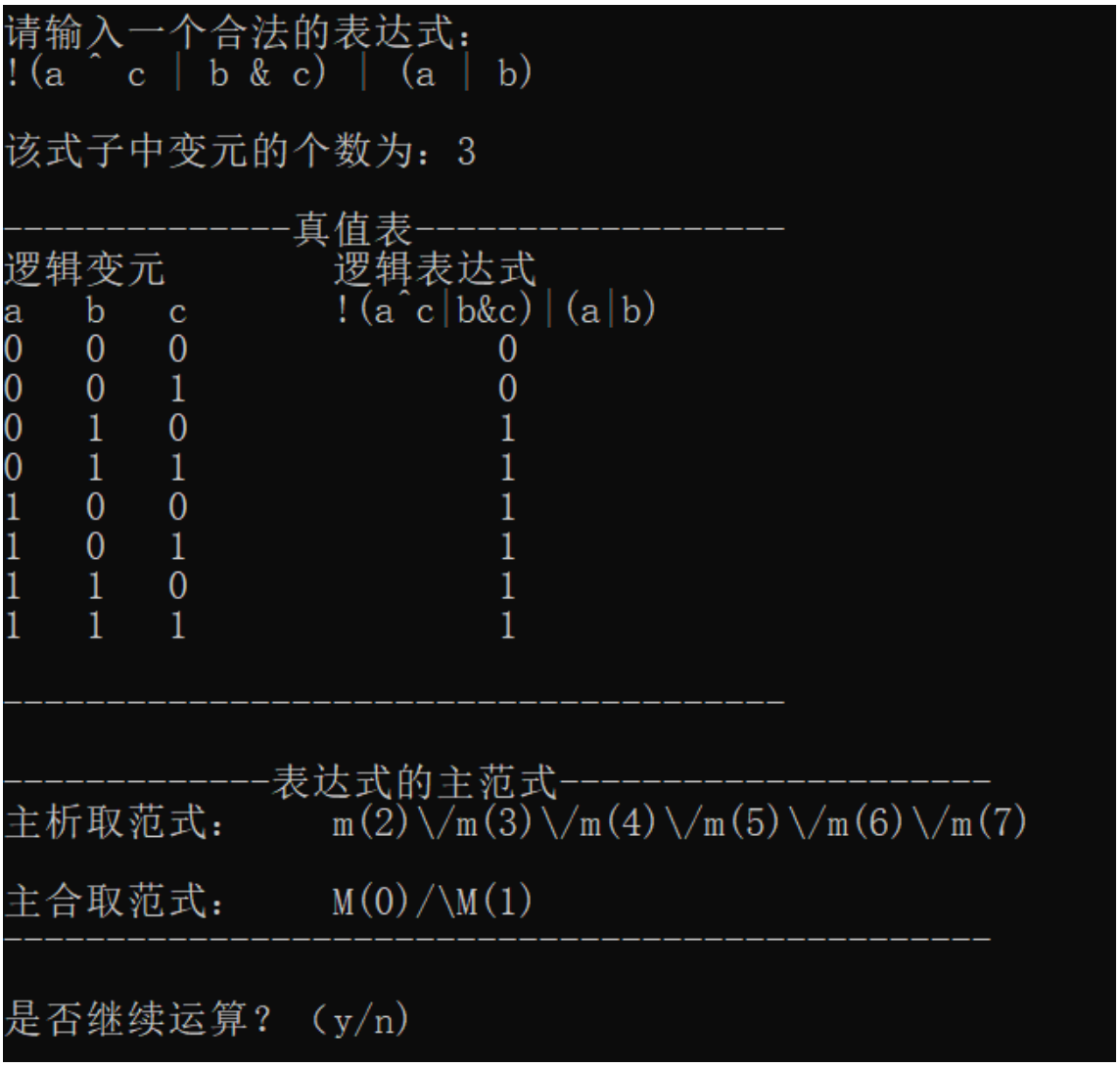
3.7. 测试

3.7.1. 正常测试

- 测试用例1

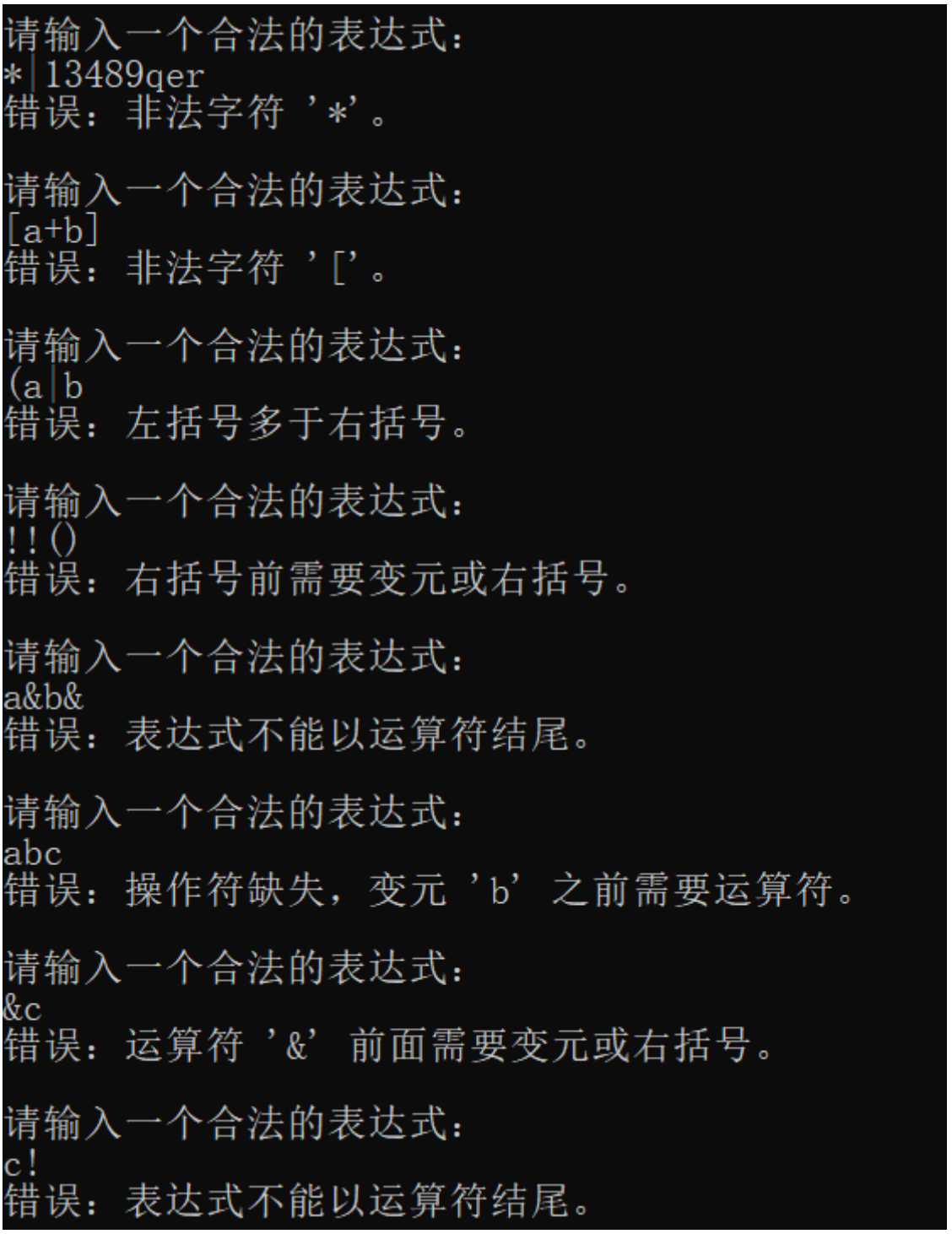
```
------------------------------------
|                                  |
|          欢迎进入逻辑软件          |
|   （可运算真值表，主范式，支持括号）   |
|                                  |
|            用 ！ 表示非           |
|            用 & 表示与            |
|            用 | 表示或            |
|           用 ^ 表示蕴含           |
|           用 ~ 表示等值           |
|                                  |
|       用小写字母表示逻辑变元        |
|                                  |
------------------------------------

请输入一个合法的表达式：
a | b

该式子中变元的个数为：2

--------------真值表------------------
逻辑变元     逻辑表达式
a    b       a | b
0    0         0
0    1         1
1    0         1
1    1         1

---------------------------------------

-------------表达式的主范式---------------------
主析取范式：      m(1)\/m(2)\/m(3)

主合取范式：      M(0)
-----------------------------------------------

是否继续运算？（y/n）
```

- 测试用例2



3.7.2. 输入错误测试

- 测试用例

- 

3.7.3. 总结

`ExpressionInput()` 函数通过对括号、字符和表达式结构的检查，确保用户输入的表达式是合法的。如果发现任何不合法的情况，函数将输出错误信息，并要求用户重新输入直至提供一个合法的表达式。

3.8. 实验心得

通过本次实验，我深刻理解了逻辑运算在编程中的具体实现方式，尤其是利用栈来处理逻辑表达式求值的问题。相较于实验1中简单的逻辑运算，本实验增加了对复杂逻辑表达式的支持，逻辑变元的数量和表达式的复杂性都显著提高了。通过模块化的设计，我实现了灵活可扩展的代码结构，使得用户能够输入任意合法的逻辑表达式进行

计算。

此外，通过编写真值表生成及主范式求取部分，我理解了逻辑运算的另一种表示形式——主范式。掌握如何将逻辑表达式转换为主析取范式和主合取范式是学习命题逻辑的重要部分，这些不同的表达形式使得逻辑公式的理解和分析更加直观和形式化。

在本次实验中，我还学会了如何处理用户输入的合法性，尤其是在涉及到复杂表达式时。通过反复测试和调试，保证程序能够正确处理括号嵌套、逻辑变元与运算符的组合，增强了程序的稳定性和用户友好性。

3.9. 总结

本次实验涉及了复杂逻辑表达式的解析和求值。通过利用栈结构、运算符优先级、真值表生成和主范式求取等方法，我加深了对逻辑表达式计算原理的理解，并掌握了如何通过程序实现这些运算。在下一步的实验中，我希望能够探索更多关于逻辑运算自动化和逻辑电路设计的相关内容，这对于我在计算机科学中的应用具有非常重要的意义。