****

**离散数学课程实验报告**

**命题逻辑联接词、真值表与主范式**

2351041

刘浩田

完成日期:2024.9.28

# 索引

### 实验目的

### 实验环境

### 实验要求

### 实验原理

### 实验过程

### 实验结果

### 实验小结

# 实验目的

本实验项目旨在通过设计和实现一个命题逻辑运算与真值表生成系统，深化对离散数学中命题逻辑部分的理解，特别是命题逻辑联接词、真值表和主范式的概念。通过编程实践，我们将学会如何将抽象的逻辑概念转化为具体的计算操作，这不仅有助于提升程序设计能力，也有助于加深对逻辑运算原理的理解。

实验的主要目的包括：

1.理解和应用命题逻辑联接词

通过编程实践来掌握合取（∧）、析取（∨）、条件（→）和双向条件（↔）等逻辑联接词的具体应用。通过输入两个命题变元的真值，并计算这些联接词的结果，直观地看到逻辑运算的过程和结果。

2.掌握真值表的构建和应用

真值表是理解复杂逻辑表达式的重要工具。需要编写程序来生成给定命题逻辑公式的真值表，要求我们能够处理变量的所有可能组合，并计算在每种组合下公式的真值。这个过程有助于理解逻辑表达式在不同输入情况下的行为。

3.计算主范式

主析取范式（DNF）和主合取范式（CNF）是逻辑公式的标准化形式，具有重要的理论价值和实际应用。学习如何根据真值表的结果计算这两种范式，对于简化逻辑表达式和优化逻辑运算具有重要意义。

4.培养用户体验意识

通过设计用户友好的交互界面，学会从用户的角度思考问题，提升用户体验，包括清晰的提示信息、简洁的操作流程和直观的输出结果。

通过本实验，我们将在理论与实践相结合的过程中，全面提升自己的专业技能和综合素质，为将来的学习和工作打下坚实的基础。

# 实验环境

## 系统：Windows11

## 编译器：VisualStudio2022x86

## 语言：C++

# 实验要求

## 4.1逻辑联接词运算

输入两个命题变量p和q的值（0/1），计算并输出它们的合取、析取、蕴涵和等值运算结果。若输入错误，比如输入的数字不是0或1，或者输入了字母，则提示重新输入p/q的值，直到输入正确为止。逻辑运算结束后，询问用户是否进行下一次运算（y/n），若输入y则清屏，重复上述步骤；若输入n则退出程序。如用户输入错误，则提示重新输入，直到正确为止。

## 4.2求任意命题公式的真值表

首先由用户输入一个逻辑运算表达式，其中用!表示非用&表示与用|表示或用^表示蕴含用~表示等值，括号采用英文括号，并对错误输入提示重输。此后针对此命题逻辑公式生成并显示其真值表。

真值表:对于给定的命题逻辑公式,真值表展示了在所有可能的命题变元真值组合下,公式的真值。

## 4.3求命题公式的主析取范式和主合取范式

基于真值表,程序计算并显示该公式的主析取范式和主合取范式。

主析取范式(DNF):它是一种逻辑公式的标准化形式,由若干个极小项的析取(逻辑"或")组成。极小项是一个只包含原子命题或其否定的合取(逻辑"与")表达式,且每个变量在极小项中只出现一次。通过真值表,可以确定哪些变量组合使公式成真,进而构造出相应的极小项,最后将这些项以逻辑"或"的形式组合起来,形成DNF。

主合取范式(CNF):与DNF类似,CNF是由若干个极大项的合取(逻辑"与")组成。极大项是一个只包含原子命题或其否定的析取(逻辑"或")表达式,且每个变量在极大项中只出现一次。通过真值表,可以确定哪些变量组合使公式成假,进而构造出相应的极大项,最后将这些项以逻辑"与"的形式组合起来,形成CNF。

# 实验原理

## 5.1命题逻辑运算符

命题逻辑中存在四种基本的逻辑运算符:合取(∧)、析取(∨)、条件(→)和双向条件(↔)。这些运算符定义了如何将简单的命题语句组合成更复杂的命题公式。

合取(∧)是一种二元逻辑运算。将两个命题P和Q通过合取运算符"∧"联结起来,形成新的命题P∧Q。当且仅当P和Q都为真时,P∧Q才为真。如果P或Q有一个为假,那么整个合取命题也为假。合取运算的真值表如下:

|  |  |  |
| --- | --- | --- |
| P | Q | P∧Q |
| 1 | 1 | 1 |
| 1 | 0 | 0 |
| 0 | 1 | 0 |
| 0 | 0 | 0 |

析取(∨)也是一种二元逻辑运算。将两个命题P和Q通过析取运算符"∨"联结起来,形成新的命题P∨Q。只要P或Q有一个为真,P∨Q就为真。只有当P和Q都为假时,整个析取命题才为假。析取运算的真值表如下:

|  |  |  |
| --- | --- | --- |
| P | Q | P∨Q |
| 1 | 1 | 1 |
| 1 | 0 | 1 |
| 0 | 1 | 1 |
| 0 | 0 | 0 |

条件(→)是一种二元逻辑运算,也称为蕴含。将两个命题P和Q通过条件运算符"→"联结起来,形成新的命题P→Q。如果P为假或者P和Q都为真,那么P→Q为真。唯一使条件为假的情况是P为真而Q为假。条件运算的真值表如下:

|  |  |  |
| --- | --- | --- |
| P | Q | P∨Q |
| 1 | 1 | 1 |
| 1 | 0 | 0 |
| 0 | 1 | 1 |
| 0 | 0 | 1 |

双向条件(↔)是一种二元逻辑运算,也称为等价。将两个命题P和Q通过双向条件运算符"↔"联结起来,形成新的命题P↔Q。当P和Q具有相同的真值时(即都为真或都为假),P↔Q为真。双向条件运算的真值表如下:

|  |  |  |
| --- | --- | --- |
| P | Q | P↔Q |
| 1 | 1 | 1 |
| 1 | 0 | 0 |
| 0 | 1 | 0 |
| 0 | 0 | 1 |

## 5.2真值表

真值表是一个表格,用来展示逻辑运算符在其操作数(通常是命题变量)的所有可能输入组合下的输出结果。每一行代表一种输入组合,列出了在这种组合下各个操作数和结果的真值。真值表通常由行和列组成,行代表不同的输入组合,列代表这些输入和输出的真值。最左侧的列是输入变量,最右侧的列是这些输入变量经过逻辑运算后的输出。

通过分析真值表,可以直观地理解逻辑表达式在不同输入情况下的行为,并用于证明两个表达式的逻辑等价性。真值表为我们提供了一种系统地探索命题逻辑的方法。

## 5.3主合取范式和主析取范式

任何一个含有n个命题变元的非永真命题公式A都存在与之等价的主合取范式(CNF)和主析取范式(DNF)。

主合取范式(CNF)是由若干个不同的极大项通过逻辑"与"连接而成的形式。极大项是一个只包含原子命题或其否定的析取表达式,且每个变量在极大项中只出现一次。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。

主析取范式(DNF)是由若干个不同的极小项通过逻辑"或"连接而成的形式。极小项是一个只包含原子命题或其否定的合取表达式,且每个变量在极小项中只出现一次。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。

通过分析命题公式的真值表,可以确定使公式成真或成假的变量组合,从而构造出相应的极小项或极大项。最后将这些项以逻辑"或"或逻辑"与"的形式组合起来,就得到了DNF或CNF。主合取范式和主析取范式是公式的唯一标准形式,能够完整描述公式的真值特性,在简化和分析逻辑表达式中十分有用。

# 实验过程

## 6.1命题逻辑联接词实验过程

### 6.1.1 实验思路

本实验旨在开发一个交互式程序，用于演示和计算基本的命题逻辑联接词。实验的主要目标包括：

设计一个用户友好的界面，允许用户输入两个命题变元P和Q的真值。

实现四种基本的逻辑联接词运算：合取(∧)、析取(∨)、条件(→)和双向条件(↔)。

清晰地展示各种逻辑运算的结果。

提供程序循环执行的功能，允许用户多次尝试不同的输入组合。

实现简单而有效的输入验证机制，确保用户输入的正确性。

### 6.1.2 实验设计

1、数据结构设计

考虑到程序的需求和简洁性，我们使用以下基本数据类型：

int: 用于存储逻辑值和用户选择。

p和q: 存储命题变元P和Q的真值（0表示假，1表示真）。

continue\_value: 存储用户是否继续程序的选择（0表示退出，1表示继续）。

char: 用于临时存储用户的键盘输入。

invalue: 在输入函数中用于捕获用户的单字符输入。

这种设计简化了数据处理，同时满足了程序的所有功能需求。

2、 函数设计

程序包含以下三个主要函数：

void InputValue(int &value)

目的：接收并验证用户输入的逻辑值（0或1）。

参数：value - 引用参数，用于存储验证后的输入值。

void InputChoice(int& value)

目的：接收并验证用户的继续/退出选择（y或n）。

参数：value - 引用参数，用于存储转换后的选择值（1表示继续，0表示退出）。

int main()

目的：作为程序的主函数，包含主要的程序逻辑和流程控制。

3、 程序主体架构设计

程序的主体架构采用了简单而有效的循环结构：

主循环（无限循环）：

清屏并显示程序标题

接收用户输入的P和Q的值

计算并显示逻辑运算结果

询问用户是否继续

根据用户选择决定继续或退出

输入处理：

使用专门的输入函数处理用户输入

实现输入验证，确保只接受有效输入

逻辑计算：

在主循环中直接进行逻辑运算

使用C++内置的逻辑运算符实现各种逻辑联接词

这种架构设计使得程序流程清晰，易于理解和维护。

### 6.1.3 程序功能实现

1、 逻辑值输入功能的实现

InputValue函数实现了逻辑值的输入功能。其主要特点包括：

使用\_getch()函数捕获单个字符输入，无需按回车键。

仅接受'0'或'1'作为有效输入。

立即显示有效输入，提供即时反馈。

将字符输入转换为对应的整数值。

实现代码：

|  |
| --- |
| 附件1、InputValue函数 |
| void InputValue(int &value)  {  char invalue;  while (1) {  invalue = \_getch();  if (invalue == '0' || invalue == '1') {  cout << invalue << endl;  break;  }    }  value = (int(invalue) - 48);  return;  } |

这种实现方式确保了输入的准确性和用户体验的流畅性。

2、用户选择输入功能的实现

InputChoice函数实现了用户选择的输入功能。其主要特点包括：

同样使用\_getch()函数捕获单个字符输入。

仅接受'y'或'n'作为有效输入。

立即显示有效输入，并将其转换为对应的整数值（y转为1，n转为0）。

实现代码：

|  |
| --- |
| 附件2、InputChoice函数 |
| void InputChoice(int& value)  {  char invalue;  while (1) {  invalue = \_getch();  if (invalue == 'y') {  cout << invalue << endl;  value = 1;  break;  }  else if (invalue == 'n') {  cout << invalue << endl;  value = 0;  break;  }  }    return;  } |

这个函数有效地处理了程序的流程控制，使用户可以方便地选择继续或退出程序。

3、 主程序逻辑的实现

主函数main()实现了整个程序的核心逻辑。其主要特点包括：

使用无限循环实现程序的重复执行。

每次循环开始时清屏并显示醒目的标题。

依次获取用户输入的P和Q的值。

计算并以清晰的格式显示各种逻辑联接词的结果。

在每次循环结束时询问用户是否继续，实现程序的可控退出。

关键实现代码片段：

|  |
| --- |
| 附件3、main函数关键实现代码片段 |
| while (1) {  system("cls");  cout << "+-----------------------------------+" << endl;  cout << "| 命题逻辑联接词演示 |" << endl;  cout << "+-----------------------------------+" << endl << endl;  int p, q;  cout << "请输入 P 的值 (0或1): ";  InputValue(p);  cout << "请输入 Q 的值 (0或1): ";  InputValue(q);  // 显示逻辑运算结果  cout << "合取" << endl;  cout << " P ∧ Q = " << (p && q) << endl;  // ... 其他逻辑运算结果显示 ...  int continue\_value;  cout << "是否继续演示？ （y/n）: ";  InputChoice(continue\_value);  if (!continue\_value) {  break;  }  } |

这种实现方式使得程序具有良好的交互性和可用性。

4、 核心算法实现

程序的核心算法主要涉及逻辑联接词的实现：

合取(∧)：

实现：p && q

原理：利用C++的逻辑与运算符，当且仅当p和q都为真（1）时，结果为真。

析取(∨)：

实现：p || q

原理：利用C++的逻辑或运算符，当p或q至少有一个为真时，结果为真。

条件(→)：

实现：!p || q

原理：利用逻辑或和逻辑非的组合，实现了"如果p则q"的语义。

双向条件(↔)：

实现：(!p || q) && (!q || p)

原理：结合了两个条件语句，实现了"p当且仅当q"的语义。

这些实现直接利用了C++的逻辑运算符，简洁而高效地表达了各种逻辑联接词的语义。通过在输出语句中直接进行这些运算，程序实现了实时的逻辑计算和结果展示。

通过这种方式，程序能够准确计算并清晰展示不同逻辑联接词在给定输入下的结果，有效地达到了实验目标，加深了对命题逻辑基本概念的理解。

## 6.2真值表与主范式实验过程

### 6.2.1实验思路

本实验旨在开发一个交互式程序，用于计算布尔表达式的真值表和主范式。实验的主要目标包括：

1. 设计一个用户友好的界面，允许用户输入复杂的布尔表达式。

2. 实现表达式解析和计算功能，支持多种逻辑运算符（非、与、或、蕴含、等值）。

3. 生成并显示完整的真值表。

4. 计算并显示主析取范式和主合取范式。

5. 提供程序循环执行的功能，允许用户多次尝试不同的表达式。

6. 实现全面的输入验证机制，确保用户输入的正确性和有效性。

通过实现这些功能，本实验旨在深化对布尔代数和命题逻辑的理解，并提高C++编程技能，特别是在复杂算法实现、字符串处理和用户交互方面。

### 6.2.2实验设计

1、 数据结构设计

考虑到程序的需求和复杂性，我们使用以下数据结构：

1. string：用于存储用户输入的布尔表达式。

2. set<char>：用于存储表达式中的变量。

3. map<char, bool>：用于存储变量及其对应的布尔值。

4. vector<bool>：用于存储真值表的结果。

5. stack<char>：用于在输入验证过程中检查括号匹配。

这种设计允许灵活处理不同长度和复杂度的表达式，同时提供了高效的数据操作和访问。

2、 函数设计

程序包含以下主要函数：

1. bool CalculateExpression(const string& input, const map<char, bool>& values)

目的：计算给定变量值下布尔表达式的结果。

2. bool ParseExpression(const string& expr, const map<char, bool>& values, size\_t& pos)

目的：解析表达式，处理或、等价、蕴含运算。

3. bool ParseTerm(const string& expr, const map<char, bool>& values, size\_t& pos)

目的：解析表达式中的项，处理与运算。

4. bool ParseFactor(const string& expr, const map<char, bool>& values, size\_t& pos)

目的：解析表达式中的因子，处理非运算、括号和变量。

5. set<char> GetVariables(const string& input)

目的：从表达式中提取所有变量。

6. void CalculateAndPrint(const string& input)

目的：计算并打印真值表和主范式。

7. bool IsValidInput(const string& input)

目的：验证用户输入的有效性。

8. string GetCanonicalForm(const vector<bool>& results, bool is\_true\_form)

目的：生成主析取范式或主合取范式。

这种函数设计促进了代码的模块化和可读性，同时通过递归下降的方法有效地处理了复杂的布尔表达式。

3、程序主体架构设计

程序的主体架构采用了以下结构：

1. 主循环：

- 清屏并显示程序标题

- 提示用户输入布尔表达式

- 验证输入的有效性

- 计算并显示真值表和主范式

- 询问用户是否继续

2. 表达式解析和计算：

- 使用递归下降方法解析表达式

- 逐层处理不同优先级的运算符

3. 真值表生成：

- 生成所有可能的变量值组合

- 对每种组合计算表达式结果

- 格式化显示真值表

4. 主范式生成：

- 基于真值表结果生成主析取范式和主合取范式

- 使用标准格式显示范式

这种架构设计使得程序能够处理复杂的布尔表达式，同时保持代码的清晰性和可维护性。

### 6.2.3 程序功能实现

1、 布尔表达式输入及验证功能的实现

IsValidInput函数实现了布尔表达式的输入验证功能。其主要特点包括：

1. 检查输入是否为空。

2. 验证字符的有效性（字母、运算符、括号）。

3. 检查语法错误（如运算符位置、重复运算符等）。

4. 使用栈检查括号匹配。

实现代码片段：

|  |
| --- |
| 附件4、IsValidInput函数 |
| bool IsValidInput(const string& input) {  if (input.empty()) { // 如果输入为空  cout << "输入无效" << endl;  return false;  }  stack<char> parentheses; // 用于检查括号匹配  char previous = '\0'; // 存储前一个字符  for (size\_t i = 0; i < input.length(); i++) { // 遍历输入字符串  char ch = input[i]; // 获取当前字符  // 检查字符是否有效  if (!((ch >= 'A' && ch <= 'Z') || (ch >= 'a' && ch <= 'z') ||  IsValidOperator(ch) || ch == '(' || ch == ')')) {  cout << "输入无效" << endl;  return false;  }  // 检查语法错误  if ((i == 0 && (ch == '&' || ch == '|' || ch == '~' || ch == '^')) ||  (previous == '(' && ch == ')') ||  (previous == '!' && ch == '!') ||  (ch == '!' && isalpha(previous)) ||  ((isalpha(ch) && previous == ')') || (ch == '(' && isalpha(previous))) ||  (isalpha(ch) && isalpha(previous)) ||  ((ch == '&' || ch == '|' || ch == '~' || ch == '^') && (!isalpha(previous) && previous != ')'))) {  cout << "输入无效" << endl;  return false;  }  // 检查括号匹配  if (ch == '(') {  parentheses.push(ch);  }  else if (ch == ')') {  if (parentheses.empty()) {  cout << "输入无效" << endl;  return false;  }  parentheses.pop();  }  previous = ch; // 更新前一个字符  }  // 检查最后一个字符和括号匹配  if (IsValidOperator(previous) || previous == '!' || !parentheses.empty()) {  cout << "输入无效" << endl;  return false;  }  return true; // 如果通过所有检查，返回true  }  // 获取用户选择（y/n）  void InputChoice(int& value) {  char in\_value;  while (1) { // 循环直到获得有效输入  in\_value = \_getch(); // 获取用户输入的字符  if (in\_value == 'y') {  cout << in\_value << endl;  value = 1;  break;  }  else if (in\_value == 'n') {  cout << in\_value << endl;  value = 0;  break;  }  }  return;  } |

这种实现确保了用户输入的布尔表达式在语法上是正确的，为后续的计算提供了可靠的基础。

2、表达式解析和计算功能的实现

表达式的解析和计算通过一系列递归函数实现，主要包括 CalculateExpression、ParseExpression、ParseTerm 和 ParseFactor。这种方法允许处理复杂的嵌套表达式和不同优先级的运算符。

|  |
| --- |
| 附件5、表达式解析和计算功能函数部分片段 |
| bool CalculateExpression(const string& input, const map<char, bool>& values) {  size\_t pos = 0; // 初始化位置为0  return ParseExpression(input, values, pos); // 从表达式开始解析  }  bool ParseExpression(const string& expr, const map<char, bool>& values, size\_t& pos) {  bool left = ParseTerm(expr, values, pos); // 解析左侧项  while (pos < expr.length()) { // 当还有字符待处理时  char op = expr[pos]; // 获取当前字符  if (op != '|' && op != '~' && op != '^') break; // 如果不是或、等价、蕴含运算符，则退出循环  pos++; // 移动到下一个字符  bool right = ParseTerm(expr, values, pos); // 解析右侧项  // 根据运算符计算结果  switch (op) {  case '|': // 或运算  left = (left || right);  break;  case '~': // 等价运算  left = (left == right);  break;  case '^': // 蕴含运算  left = (!left || right);  break;  }  }  return left; // 返回最终结果  } |

ParseTerm 和 ParseFactor 函数类似，分别处理与运算和非运算、括号、变量

这种实现方式能够准确计算复杂布尔表达式的结果，为真值表的生成提供了核心功能。

3、真值表生成功能的实现

真值表的生成通过 CalculateAndPrint 函数实现。该函数首先获取表达式中的所有变量，然后生成所有可能的变量值组合，对每种组合计算表达式的结果。

关键实现代码片段：

|  |
| --- |
| 附件6、CalculateAndPrint函数 |
| void CalculateAndPrint(const string& input) {  set<char> variables = GetVariables(input); // 获取所有变量  int num\_variables = static\_cast<int>(variables.size()); // 获取变量数量  PrintTableHeader(variables); // 打印表头  vector<bool> results; // 存储所有可能输入的结果  for (int i = 0; i < (1 << num\_variables); ++i) { // 遍历所有可能的输入组合  map<char, bool> variable\_values; // 存储当前组合的变量值  int index = 0;  for (char var : variables) { // 为每个变量赋值  variable\_values[var] = (i & (1 << index)) != 0;  ++index;  }  bool result = CalculateExpression(input, variable\_values); // 计算结果  results.push\_back(result); // 存储结果  PrintTableRow(variable\_values, result); // 打印当前行  }  cout << "主析取范式: " << endl;  cout << GetCanonicalForm(results, true) << endl; // 打印主析取范式  cout << "主合取范式: " << endl;  cout << GetCanonicalForm(results, false) << endl; // 打印主合取范式  } |

这种实现方式不仅生成了完整的真值表，还为主范式的计算提供了必要的数据。

4、核心算法实现

程序的核心算法主要涉及布尔表达式的解析和计算、真值表的生成以及主范式的构建。

1.布尔表达式解析

布尔表达式的解析采用递归下降法，按照运算符的优先级逐层处理：

1. ParseExpression 处理最低优先级的运算（或、等价、蕴含）。

2. ParseTerm 处理与运算。

3. ParseFactor 处理最高优先级的运算（非、括号、变量）。

这种方法能够有效处理复杂的嵌套表达式和不同优先级的运算符。

2. 真值表生成

真值表生成算法的步骤如下：

1. 提取表达式中的所有变量。

2. 生成所有可能的变量值组合（2^n，其中n是变量数量）。

3. 对每种组合计算表达式的结果。

4. 格式化输出每一行的结果。

这种方法确保了真值表的完整性和准确性。

3. 主范式构建

主范式的构建基于真值表的结果，实现步骤如下：

1. 对于主析取范式，选择真值表中结果为真的所有行。

2. 对于主合取范式，选择真值表中结果为假的所有行。

3. 使用标准格式（M<i> 或 m<i>）表示每个项。

4. 用适当的连接符（∨ 或 ∧）连接所有项。

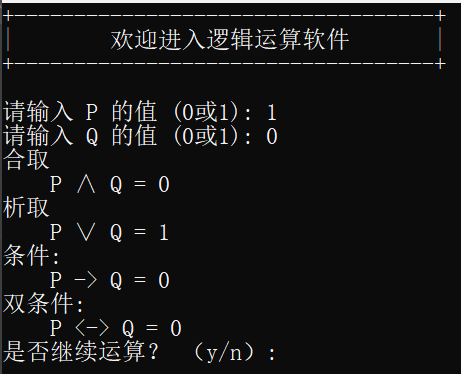
这种实现方式准确地生成了标准形式的主析取范式和主合取范式。

通过这些核心算法的实现，程序能够有效地处理复杂的布尔表达式，生成完整的真值表，并构建标准的主范式，从而达到了实验的预期目标。

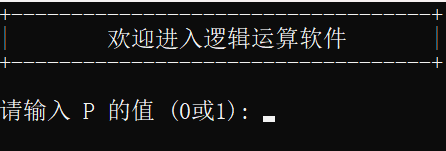
# 实验结果

## 7.1 命题逻辑联接词结果

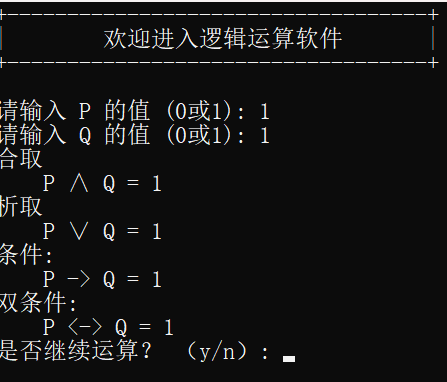
测试：当P的逻辑值为1，Q的逻辑值为0时，P∧Q为假，P∨Q为真，P→Q为假，P↔Q为假。



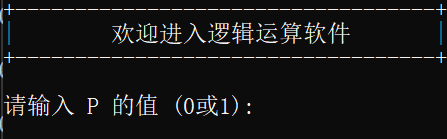
由于使用了\_getch()函数，当p、q输入值不为1、0时，不会显示输入，有光标。



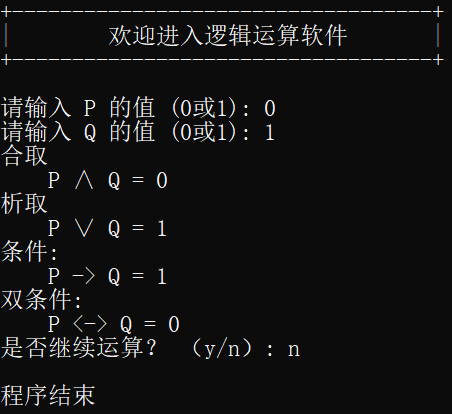
同样，当选则输入值不为y、n时，不会显示输入，有光标。



如果输入y，程序会清屏并进行下一次运算。

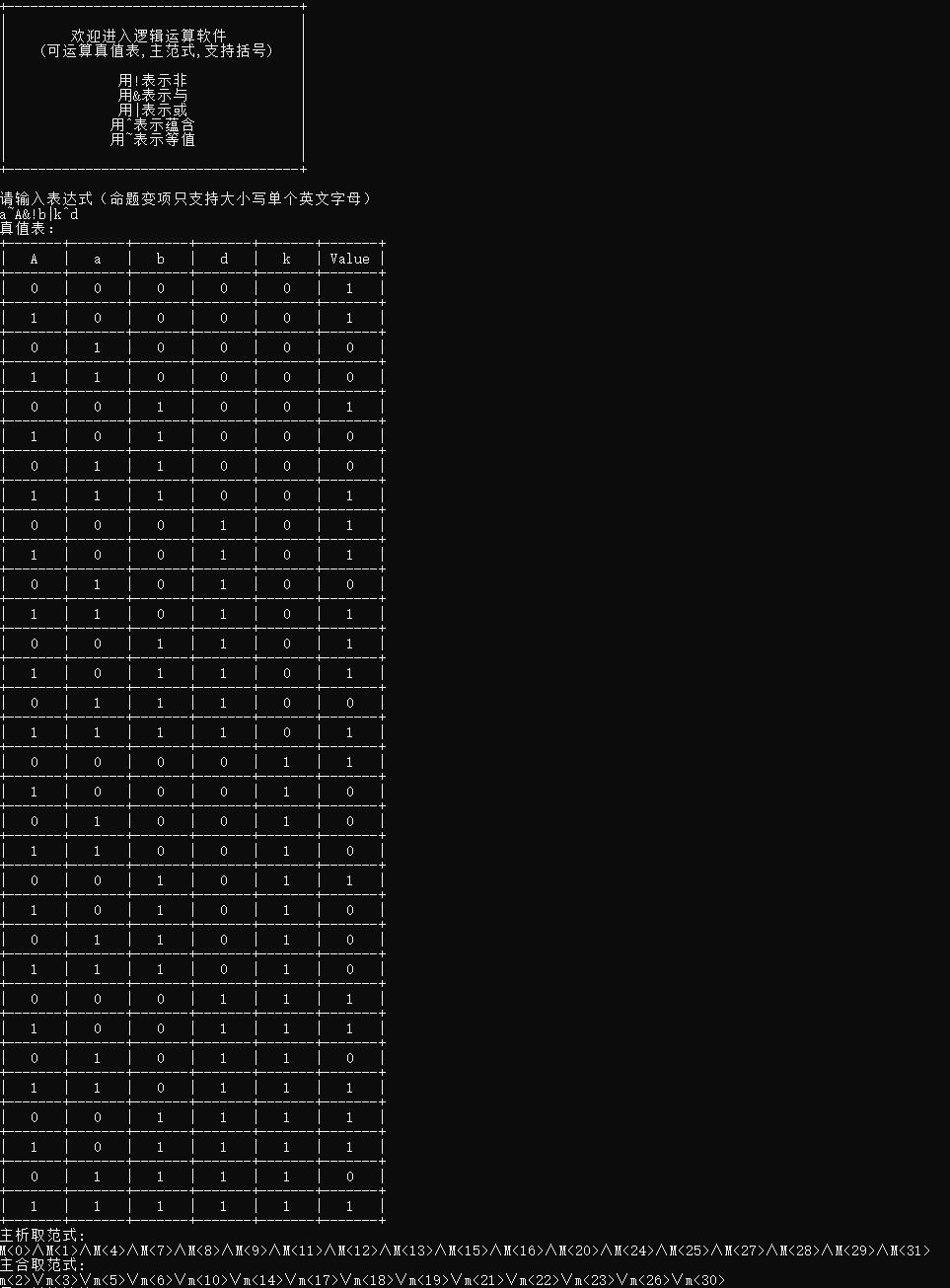


如果输入n，程序会退出，并输出提示。

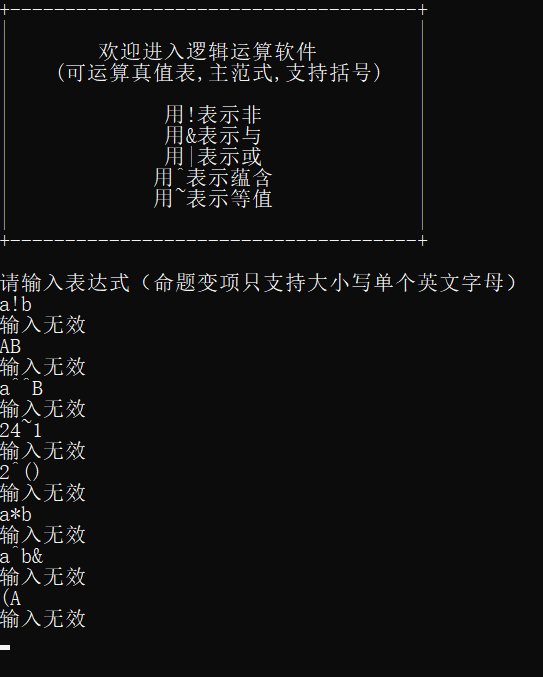


## 7.2 真值表与主范式实验结果

测试1：输入正确公式，得到正确结果



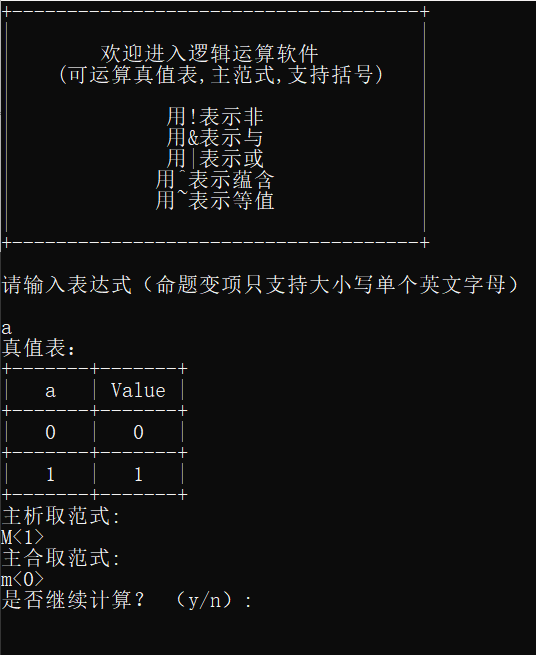
测试2：输入错误输入，检查错误

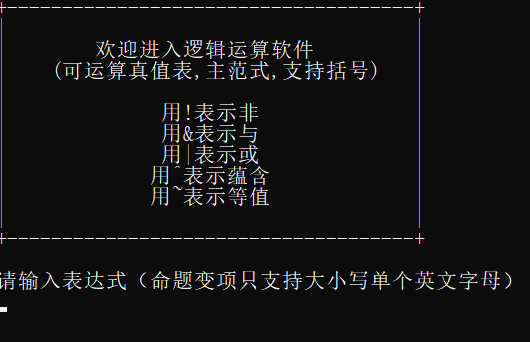


这里分别检查了使用！当作连接运算符、不合法变项、多个运算符连接、空括号、不合法运算符、运算符使用不恰当的情况。

测试3：输入继续计算选项

由于此处函数与上一模块使用相同，故逻辑、显示相同。





# 实验小结

通过本次离散数学课程的实验，我深刻体会到了命题逻辑联接词、真值表与主范式在计算机科学中的重要性。实验过程中，我首先设计并实现了一个命题逻辑运算与真值表生成系统，这不仅加深了我对命题逻辑联接词的理解，也锻炼了我的编程实践能力。通过编程实践，我学会了如何将抽象的逻辑概念转化为具体的计算操作，这对于我理解和应用逻辑运算原理具有重要意义。

在实验中，我首先实现了逻辑联接词的运算，包括合取、析取、条件和双向条件。通过输入命题变量的值并计算结果，我直观地观察到了逻辑运算的过程和结果。接着，我编写程序生成了给定命题逻辑公式的真值表，这个过程帮助我理解了逻辑表达式在不同输入情况下的行为。最后，我学习了如何根据真值表的结果计算主析取范式和主合取范式，这对于简化逻辑表达式和优化逻辑运算具有重要的理论价值和实际应用。

此外，实验还培养了我从用户的角度思考问题，提升用户体验的意识。我设计了用户友好的交互界面，包括清晰的提示信息、简洁的操作流程和直观的输出结果，这些都极大地提升了用户的使用体验。

总的来说，这次实验不仅加深了我对离散数学中命题逻辑部分的理解，也提高了我的程序设计能力。我相信这些知识和技能将为我未来的学习和工作打下坚实的基础。