同济大学

TONGJI UNIVERSITY

离散数学课程设计

项目名称	命题逻辑联接词、真值表、主范式
学　　院	计算机科学与技术学院
专　　业	软件工程
学生姓名	杨瑞晨
学　　号	2351050
指导教师	唐剑锋
日　　期	2024 年 12 月 1 日

目 录

1 项目分析

1.1 项目背景

在命题逻辑领域，正确运用联接词和计算公式的真值是至关重要的。简而言之，我们首先着眼于两个命题 p 和 q，探讨在特定取值下，如何通过不同的联接词将它们组合，并确定这些组合公式的真值。此外，对于包含更多变量和逻辑联接词的复杂公式，我们同样关注其成真赋值、成假赋值、主析取范式和主合取范式。因此，本项目旨在实现上述两个功能。

1.2 项目要求

（1）**逻辑联接词的运算**：

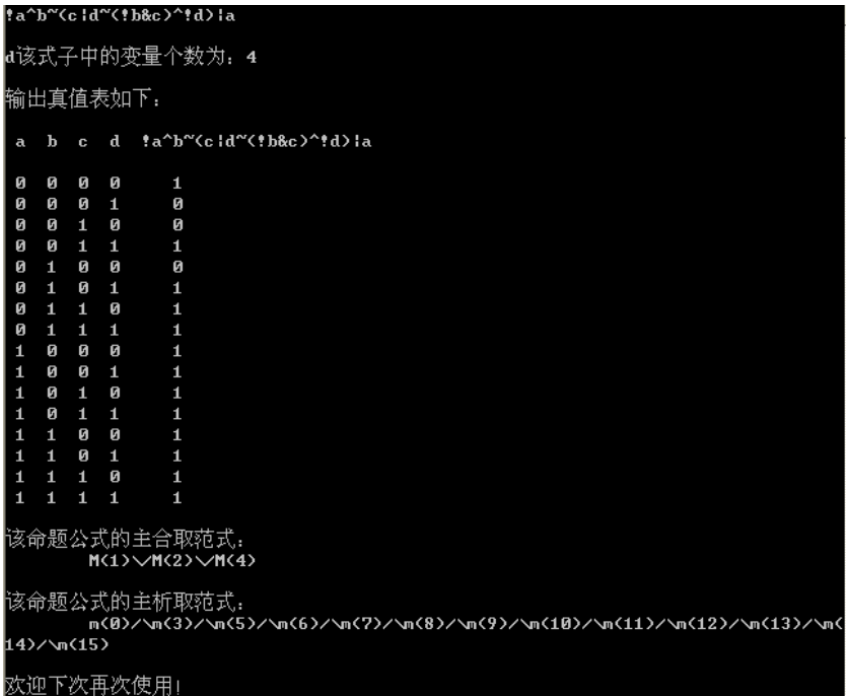
要求实现二元合取、析取、条件和双向条件表达式的计算。(A)

（2）**求任意一个命题公式的真值表和主析取范式、主合取范式**：

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。(B)

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。(C)

1.3 项目示例



1.4 项目环境

使用 C++ 语言实现，开发环境为 Linux 下的 gcc 编译器。

2 项目设计

2.1 数据结构应用

本项目中，A 题直接进行逻辑运算，无需额外数据结构。B 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，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)

2.1.1 std::map

std::map 是 C++ 标准模板库（STL）中的一个关联容器，它能够存储由键（key）和值（value）组成的键值对（key-value pairs）。std::map 中的元素是按照键的顺序自动排序的，通常实现为红黑树，这使得它能够提供快速的键值查找、插入和删除操作。

以下是 std::map 的一些主要特性：

（1）键值对存储：每个元素包含一个键和一个与之关联的值。

（2）自动排序：元素根据键的顺序自动排序，通常是按照键的升序排列。

（3）唯一键：每个键在 std::map 中都是唯一的。

（4）高效操作：提供对数时间复杂度的查找、插入和删除操作。

（5）迭代器支持：可以使用迭代器访问 std::map 中的元素。

（6）键值访问：可以通过键快速访问对应的值。

2.1.2 std::stack

std::stack 是 C++ 标准模板库（STL）中的一个容器适配器，它提供了对后进先出（LIFO）数据结构的支持。容器适配器不拥有其元素，而是“包装”了另一个容器，如 std::vector、std::deque 或 std::list，以提供特定的接口。

以下是 std::stack 的一些主要特性：

（1）LIFO 顺序：元素按照后进先出的顺序进行添加和移除。

（2）动态大小：std::stack 的大小可以动态变化。

（3）只读访问：std::stack 只提供了对栈顶元素的访问，不支持随机访问。

（4）容器适配器：它使用另一个容器来存储元素，通常默认使用 std::deque。

2.1.3 std::string

std::string 是 C++ 标准库中的一个类，用于表示字符串。std::string 类提供了一系列成员函数，用于处理字符串的操作，如连接、查找、替换、插入和删除等。

以下是 std::string 的一些主要特性：

（1）动态大小：std::string 的长度可以根据需要动态变化。

（2）字符编码：通常使用 UTF-8 或其他编码方式存储字符。

（3）内存管理：自动管理内存分配和释放，无需手动操作。

（4）操作符重载：支持多种操作符的重载，如 +（连接）、==（比较）等。

（5）迭代器支持：提供迭代器，可以遍历字符串中的每个字符。

2.2 算法设计

2.2.1 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 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。

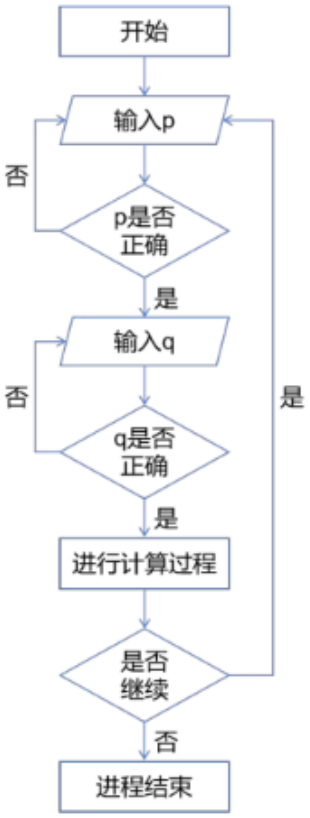
2.2.2 性能评估

第一题由于变量没有数量上的变化，时间固定为 $O(1)$。

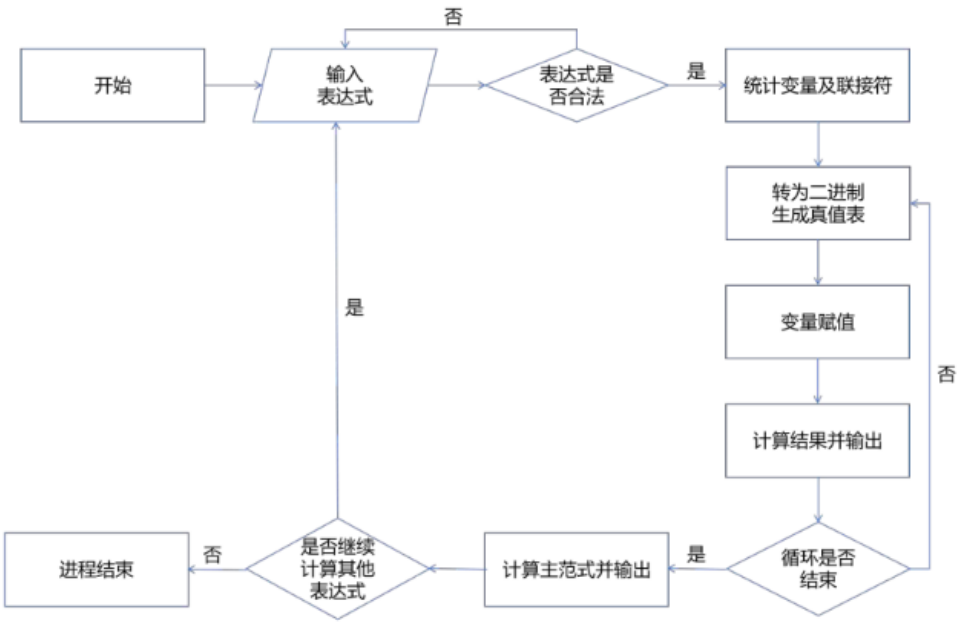
第二题较为复杂，因为并不了解 map 的访问等操作的内部实现，设 map 的各类操作时间为 $O(1)$，同时设有 k 个命题变项，表达式长度为 n ，由于所有的命题取值有 2^k 种，因此需进行 2^k 次循环，每次需遍历一遍表达式，需要 $O(n)$，故总的时间复杂度为 $O(n * 2^k)$。

2.2.3 流程图表示

A 题：


开始
输入p
否
p是否正确
是
输入q
否
q是否正确
是
进行计算过程
是否继续
是
否
进程结束

B 题:


否
开始
输入表达式
表达式是否合法
是
统计变量及联接符
转为二进制生成真值表
变量赋值
计算结果并输出
循环是否结束
否
是
计算主范式并输出
是否继续计算其他表达式
否
进程结束
是

2.2.4 代码实现

A 题：

```cpp
#include <iostream>
#include <vector>
#include <climits>

using namespace std;

//显示欢迎消息
void welcome()
{
    cout << "****************************************" << endl;  //标语
    cout << "**                                    **" << endl;;
    cout << "**        欢迎进入逻辑运算程序        **" << endl;
    cout << "**                                    **" << endl;
    cout << "****************************************\n" << endl;

}

//输入处理
int getInput(char ch)
{
    while (true) {
        int input;
        cout << " 请输入" << ch << " 的值 (1 或 0)，以回车结束：";
        cin >> input;
        if ((input != 1 && input != 0) || cin.fail()) {
            cout << endl << ch << " 的值输入有误，请重新输入！" << endl;
            cin.clear(); //清空缓冲区
            cin.ignore(INT_MAX, '\n');
            continue;
        }
        else {
            cin.ignore(INT_MAX, '\n');
            return input;
        }
    }
}
```

```
37
38  //继续计算？
39  bool continueCalc()
40  {
41      bool flag;
42      char input;
43      cout << endl << " 是否继续运算？(y/n) ";
44      while (true) {
45          cin >> input;
46          if ((input != 'y' && input != 'n') || cin.fail()) {
47              cout << " 错误！请输入 y/n" << endl;
48              cin.clear(); //清空缓冲区
49              cin.ignore(INT_MAX, '\n');
50              continue;
51          }
52          else {
53              if (input == 'y') {
54                  flag = true;
55              }
56              else {
57                  cout << " 欢迎下次再次使用！ " << endl;
58                  flag = false;
59              }
60              cin.ignore(INT_MAX, '\n');
61              return flag;
62          }
63      }
64  }
65
66  int main() {
67      welcome();
68
69      while (true) {
70          int P = getInput('P');
71          int Q = getInput('Q');
72
73          vector<int> calc;
74          calc.push_back(P && Q);                    //与运算
75          calc.push_back(P || Q);                    //或运算
```

```
        calc.push_back(!P || Q);                    //蕴含运算，与与或非等价
        calc.push_back((!P || Q) && (!Q || P));     //等值运算

        cout << endl;
        cout << " 合取：\n\tp /\\ q = " << calc[0] << endl;
        cout << " 析取：\n\tp \\/ q = " << calc[1] << endl;
        cout << " 条件：\n\tp -> q = " << calc[2] << endl;
        cout << " 双条件：\n\tp <-> q = " << calc[3] << endl;

        if (!continueCalc()) {
            break;
        }
    }
    return 0;
}
```

B 题：

```
#include <iostream>
#include <string>
#include <map>
#include <stack>
#include <cmath>
#include <limits> // for std::numeric_limits
using namespace std;

typedef map<char, int> Map_ci; // 定义字符到整数的映射，用于表示运算符及其优先级
typedef map<int, char> Map_ic; // 定义整数到字符的映射，用于命题变量及其索引
typedef map<int, int> Map_ii;  // 定义整数到整数的映射，用于存储命题变量的二进制值

Map_ci priority; // 全局变量，存储运算符的优先级

Map_ic getProposition(string formula);
int findProposition(Map_ic, char p);
int pow2(int n);
Map_ii toBinary(int n_proposition, int index);
int calculate(string formula, Map_ic pSet, Map_ii value);
void check(stack<int> &value, stack<char> &opter);
bool isOperator(const char ch);
void printMenu();

// 输出菜单
void printMenu()
{
    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";
```

```
32      cout << "**              用! 表示非             **\n";
33      cout << "**              用 & 表示与            **\n";
34      cout << "**              用 | 表示或            **\n";
35      cout << "**             用 ^ 表示蕴含           **\n";
36      cout << "**             用 ~ 表示等值           **\n";
37      cout << "**                                    **\n";
38      cout << "****************************************\n\n";
39  }
40
41  // 判断字符是否为运算符
42  bool isOperator(const char ch)
43  {
44      return ch == '!' || ch == '&' || ch == '|' || ch == '^' || ch == '~' || ch == '(' || ch ==
        ↪  ')';
45  }
46
47  bool isValidFormula(const string &str)
48  {
49      //游标对象
50      string::const_iterator it = str.begin();
51      //记录括号情况的栈
52      stack<char> brackets;
53      //操作符（单目、双目）个数，数字个数
54      int op1Num = 0, op2Num = 0, numNum = 0;
55
56      while (it != str.end()) {
57          //若扫描到字符表示命题，则进行计数
58          if (((*it) >= 'a' && (*it) <= 'z') || ((*it) >= 'A' && (*it) <= 'Z'))
59              //一个数字扫描完成，进行计数
60              numNum++;
61          //若遇到左括号，压入栈中
62          if (*it == '(')
63              brackets.push(*it);
64          else if (*it == ')')
65          {
66              /* 若遇到右括号，栈不为空则弹出一个元素；
67                栈为空则说明右括号多余，非法 */
68              if (!brackets.empty())
69                  brackets.pop();
70
71              else {
72                  cout << " 括号不匹配" << endl;
73                  return false;
74              }
75          }
76          else if (isOperator(*it))
77          {
78              //若遇到操作符，进行计数
79              if (*it == '!')
80                  op1Num++;
81              else
82                  op2Num++;
83          }
84
85          if (it != str.end())
86              it++;
87          else
88              break;
89      }
```

```
90
91      if (!brackets.empty())
92      {
93          //若扫描完成栈仍不为空，则左括号多余，非法
94          cout << " 括号不匹配" << endl;
95          return false;
96      }
97
98      if (numNum != op2Num + 1)
99      {
100         //若双目运算符个数不等于数字个数 +1，则不匹配，非法
101         cout << " 操作符与操作数数量不匹配" << endl;
102         return false;
103     }
104     //所有标准均符合，则表达式合法
105     return true;
106 }
107
108 bool askToContinue()
109 {
110     char ch;
111     while (true)
112     {
113         cout << " 是否继续运算 (Y/N)？" << endl;
114         cin >> ch;
115         if (ch == 'Y' || ch == 'y')
116         {
117             cin.clear();
118             cin.ignore(numeric_limits<streamsize>::max(), '\n');
119             system("cls");
120             return true;
121         }
122         else if (ch == 'N' || ch == 'n') {
123             return false;
124         }
125         else
126         {
127             cout << " 输入错误，请重新输入" << endl;
128             cin.clear();
129             cin.ignore(numeric_limits<streamsize>::max(), '\n');
130         }
131     }
132 }
133
134 int main()
135 {
136     priority['('] = 6;
137     priority[')'] = 6;
138     priority['!'] = 5;
139     priority['&'] = 4;
140     priority['|'] = 3;
141     priority['^'] = 2;
142     priority['~'] = 1;
143     priority['#'] = 0; // 结束符
144     // 运算符优先级
145
146     while (true)
147     {
148         printMenu();
```

```
149
150         string formula;
151         while (true)
152         {
153             cout << " 请输入合法的命题公式: " << endl;
154             cin >> formula;
155             if (isValidFormula(formula))
156                 break;
157             else
158             {
159                 cout << " 输入的公式不合法，请重新输入! " << endl;
160                 cin.clear();
161                 cin.ignore(numeric_limits<streamsize>::max(), '\n');
162             }
163         }
164
165         Map_ic proposition_set = getProposition(formula); // 获取公式中的命题变项（即命题字母），
            ↪ 并返回它们的集合
166         cout << " 该式子中的变量个数为: " << proposition_set.size() << endl;
167
168         cout << " 输出真值表如下: " << endl;
169         for (unsigned int i = 0; i < proposition_set.size(); i++)
170         {
171             cout << proposition_set[i] << "\t";
172         }
173         cout << formula << endl;
174
175         int *m = new int[pow2(proposition_set.size())];   // 动态分配内存用于保存每行真值表计算
            ↪ 的结果，依次存放命题公式的各行 (1 或 0) 的运算结果的值
176
177         for (int i = 0; i < pow2(proposition_set.size()); i++)
178         {
179             // 将当前行数转换为命题变量的二进制组合
180             Map_ii bina_set = toBinary(proposition_set.size(), i);
181             // 输出当前行的二进制组合（即命题变项的真假值）
182             for (unsigned int j = 0; j < bina_set.size(); j++)
183             {
184                 cout << bina_set[j] << "\t";
185             }
186             // 计算当前行下，公式在对应命题变量取值下的结果
187             int result = calculate(formula, proposition_set, bina_set);
188             *(m + i) = result;
189             cout << result << endl;
190         }
191
192         // 输出该逻辑公式的主析取范式 (Disjunctive Normal Form)
193         int n_m = 0;
194         cout << " 该命题公式的主析取范式: " << endl;
195         for (int i = 0; i < pow2(proposition_set.size()); i++)
196         {
197             if (*(m + i) == 1)
198             {
199                 if (n_m == 0)
200                 {
201                     cout << "m<" << i << ">";
202                 }
203                 else
204                 {
205                     cout << " \\/ m<" << i << "> ";
```

```
206                 }
207                 n_m++;
208             }
209         }
210         if (n_m == 0)
211         { // 如果没有一个组合使公式为真，输出 0
212             cout << "0";
213         }
214         cout << endl;
215 
216         // 输出该逻辑公式的主合取范式（Conjunctive Normal Form）
217         int n_M = 0;
218         cout << " 该命题公式的主合取范式: " << endl;
219         for (int i = 0; i < pow2(proposition_set.size()); i++)
220         {
221             if (*(m + i) == 0)
222             {
223                 if (n_M == 0)
224                 {
225                     cout << "M<" << i << ">";
226                 }
227                 else
228                 {
229                     cout << " /\\ M<" << i << "> ";
230                 }
231                 n_M++;
232             }
233         }
234         if (n_M == 0)
235         {
236             cout << "0";
237         }
238         cout << endl
239              << endl;
240 
241         delete[] m;
242 
243         if (!askToContinue())
244             break;
245     }
246     return 0;
247 }
248 
249 int findProposition(Map_ic pSet, char p) // 返回-1，表示该命题变项尚未被遍历过，可计数；否则说明
    ↪ 该命题变项已被遍历过，则不重复计数。另外，还可以返回指定命题变项的下标
250 {
251     // 遍历命题集合，查找命题变量 p 的下标
252     Map_ic::iterator it = pSet.begin();
253     while (it != pSet.end())
254     {
255         if (it->second == p)
256         {
257             return it->first; // 返回该命题变量的下标
258         }
259         it++;
260     }
261     return -1; // 未找到，返回 -1，表示该命题变量未被记录
262 }
263 
```

```
264 Map_ic getProposition(string formula) // 该函数返回所输入公式中的命题变项（不包括运算符）
265 {
266     Map_ic proposition;       // 用于存储命题变项
267     int n_proposition = 0; // 记录当前命题变量的数量
268 
269     for (unsigned int i = 0; i < formula.length(); i++)
270     {
271         char c = formula[i];
272         if (isalpha(c))
273         {
274             // 遍历所有命题变项
275             int r = findProposition(proposition, c);
276             if (r == -1)
277             {
278                 // 说明该命题变项尚未被遍历过
279                 proposition[n_proposition] = c; // 将命题变量添加到集合中
280                 n_proposition++;               // 命题变量计数加 1
281             }
282         }
283         else if (!priority.count(c))
284         { // 如果当前字符不是命题变量且不在优先级表中
285             cerr << c << " is undefined!" << endl;
286             exit(2);
287         }
288     }
289     return proposition;
290 }
291 
292 Map_ii toBinary(int n_proposition, int index) // 该函数返回命题变项的二进制（1 或 0）取值
293 {
294     Map_ii result;
295     // 将整数 index 转换为二进制表示，并存储到 result 中
296     for (int i = 0; i < n_proposition; ++i)
297     {
298         int r = index % 2;
299         result[n_proposition - 1 - i] = r;
300         index = index / 2;
301     }
302     return result;
303 }
304 
305 int pow2(int n) // 该函数返回指定数字的二次方的值
306 {
307     if (n == 0)
308         return 1;
309     else
310         return 2 * pow2(n - 1);
311 }
312 
313 int calculate(string formula, Map_ic pSet, Map_ii value) // 该函数返回给定命题变项（值取 1 或
    ↪ 0，可含括号）组合的运算结果
314 {
315     stack<char> opter; // 运算符栈
316     stack<int> pvalue; // 操作数栈
317 
318     opter.push('#');          // 将 # 作为结束符入栈
319     formula = formula + "#"; // 在公式末尾添加结束符
320 
321     for (unsigned int i = 0; i < formula.length(); i++)
```

```
322     {
323         char c = formula[i];
324         if (isalpha(c))
325         {                                       // 如果是命题变项
326             int pos = findProposition(pSet, c); // 查找命题变项的下标
327             pvalue.push(value[pos]);            // 将对应的真值 (0 或 1) 入栈
328         }
329         else
330         {
331             // 此时遍历的是运算符
332             char tmp = opter.top(); // 获取栈顶运算符
333             // 如果当前栈顶运算符优先级高于当前运算符
334             if (priority[tmp] > priority[c])
335             {
336                 while (priority[tmp] > priority[c] && tmp != '(')
337                 {
338                     check(pvalue, opter); // 执行运算并更新栈
339                     tmp = opter.top();      // 更新栈顶运算符
340                     if (tmp == '#' && c == '#')
341                     {                           // 如果是结束符，返回最终结果
342                         return pvalue.top(); // 返回运算结果
343                     }
344                 }
345                 opter.push(c); // 否则，将当前运算符入栈
346             }
347             else
348                 opter.push(c);
349         }
350     }
351     return -1;
352 }
353
354 void check(stack<int> &value, stack<char> &opter) // 该函数返回两个命题变项 (取值 1 或 0) 的各
    ↪ 种运算结果 (0 或 1)
355 {
356     // 该函数用于执行栈顶两个命题变量的运算，并将结果存入栈中。
357     // value 栈存储命题变量的真假值，opter 栈存储运算符。
358
359     int p, q, result;        // 定义两个命题变量的值 p 和 q，以及存储结果的 result
360     char opt = opter.top(); // 获取栈顶的运算符
361
362     switch (opt)
363     {
364     case '&':                // 与运算符处理
365         p = value.top();    // 取出栈顶命题变量 p 的值
366         value.pop();        // 将 p 从栈中弹出
367         q = value.top();    // 取出下一个栈顶命题变量 q 的值
368         value.pop();        // 将 q 从栈中弹出
369         result = p && q;    // 计算 p 与 q 的逻辑与运算
370         value.push(result); // 将计算结果压入栈中
371         opter.pop();        // 将与运算符从运算符栈中弹出
372         break;
373
374     case '|':                // 或运算符处理
375         p = value.top();    // 取出栈顶命题变量 p 的值
376         value.pop();        // 将 p 从栈中弹出
377         q = value.top();    // 取出下一个栈顶命题变量 q 的值
378         value.pop();        // 将 q 从栈中弹出
379         result = p || q;    // 计算 p 与 q 的逻辑或运算
```

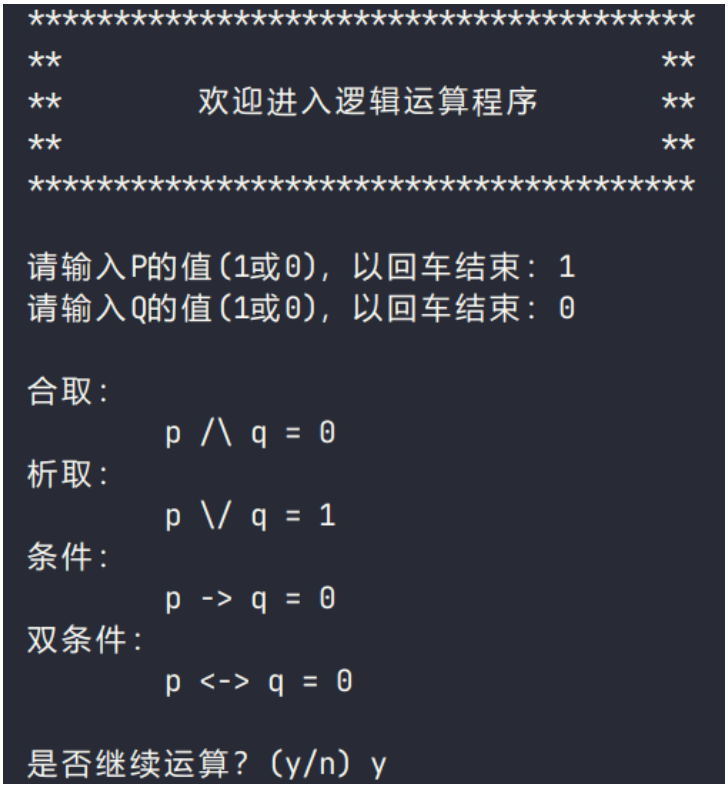
```
380         value.push(result); // 将计算结果压入栈中
381         opter.pop();        // 将或运算符从运算符栈中弹出
382         break;
383 
384     case '!':                // 非运算符处理
385         p = value.top();    // 取出栈顶命题变量 p 的值
386         value.pop();        // 将 p 从栈中弹出
387         result = !p;        // 计算 p 的逻辑非运算
388         value.push(result); // 将计算结果压入栈中
389         opter.pop();        // 将非运算符从运算符栈中弹出
390         break;
391 
392     case '^':                // 蕴含运算符处理 (p -> q)
393         q = value.top();    // 取出栈顶命题变量 q 的值
394         value.pop();        // 将 q 从栈中弹出
395         p = value.top();    // 取出下一个栈顶命题变量 p 的值
396         value.pop();        // 将 p 从栈中弹出
397         result = !p || q;    // 计算 p 蕴含 q 的逻辑运算 (!p 或 q)
398         value.push(result); // 将计算结果压入栈中
399         opter.pop();        // 将蕴含运算符从运算符栈中弹出
400         break;
401 
402     case '~':             // 等值运算符处理 (p <-> q)
403         p = value.top(); // 取出栈顶命题变量 p 的值
404         value.pop();     // 将 p 从栈中弹出
405         q = value.top(); // 取出下一个栈顶命题变量 q 的值
406         value.pop();     // 将 q 从栈中弹出
407         // 计算 p 和 q 的逻辑等值运算: (!p 或 q) 且 (p 或 !q)
408         result = (!p || q) && (p || !q);
409         value.push(result); // 将计算结果压入栈中
410         opter.pop();        // 将等值运算符从运算符栈中弹出
411         break;
412 
413     case '#': // 结束符处理
414         // 结束符不做处理，直接跳过
415         break;
416 
417     case '(': // 左括号处理
418         // 左括号不做处理，直接跳过
419         break;
420 
421     case ')':         // 右括号处理
422         opter.pop(); // 弹出右括号
423         // 一直执行运算直到遇到左括号
424         while (opter.top() != '(')
425         {
426             check(value, opter); // 递归调用 check 函数，计算括号内的表达式
427         }
428         if (opter.top() == '(')
429         {                 // 如果遇到左括号
430             opter.pop(); // 弹出左括号
431         }
432         break;
433 
434     default: // 默认处理，处理其他情况
435         break;
436     }
437 }
```

3 项目测试

3.1 A 题

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。

3.1.1 正常测试



3.1.2 健壮性测试

```
是否继续运算？（y/n）y
请输入P的值(1或0)，以回车结束：3

P的值输入有误，请重新输入！
请输入P的值(1或0)，以回车结束：q

P的值输入有误，请重新输入！
请输入P的值(1或0)，以回车结束：0
请输入Q的值(1或0)，以回车结束：u4399r34jf

Q的值输入有误，请重新输入！
请输入Q的值(1或0)，以回车结束：1

合取：
        p /\ q = 0
析取：
        p \/ q = 1
条件：
        p -> q = 1
双条件：
        p <-> q = 0

是否继续运算？（y/n）8
错误！请输入 y/n
n
欢迎下次再次使用！
```

3.2 B 题

B，C 题目由于本身关系比较密切，所以将两个做在了一起。这个程序达到了题目要求的各个功能，可以运算与，或，非，蕴含，等值条件组成的表达式，并且支持括号运算。同时，程序还支持了错误输入的检测，如果输入的表达式不符合规范，程序会进行错误提示，并要求重新输入。

3.2.1 **正常测试**

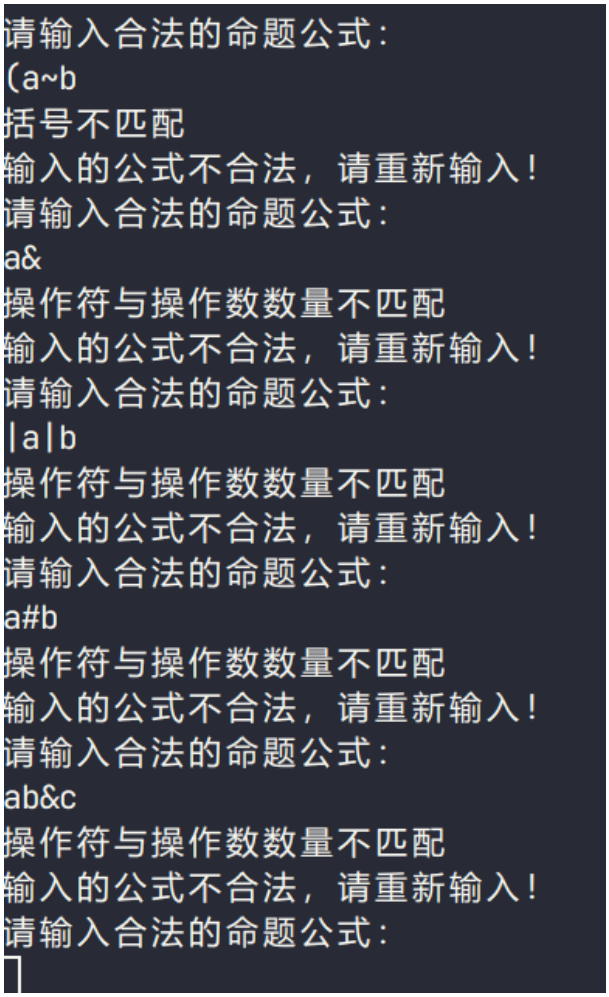
```
**************************************
**                                  **
**          欢迎进入逻辑运算软件        **
**   (可运算真值表,主范式,支持括号)   **
**                                  **
**              用!表示非            **
**              用&表示与            **
**              用|表示或            **
**              用^表示蕴含          **
**              用~表示等值          **
**                                  **
**************************************

请输入合法的命题公式:
!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
该命题公式的主析取范式:
m<0> \/ m<1>  \/ m<3>
该命题公式的主合取范式:
M<2>

是否继续运算(Y/N)?
```

```
请输入合法的命题公式:
(a~b)|c&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
0       0       0       0       1
0       0       0       1       1
0       0       1       0       1
0       0       1       1       1
0       1       0       0       0
0       1       0       1       0
0       1       1       0       0
0       1       1       1       1
1       0       0       0       0
1       0       0       1       0
1       0       1       0       0
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
该命题公式的主析取范式:
m<0> \/ m<1>  \/ m<2>  \/ m<3>  \/ m<7>  \/ m<11>  \/ m<12>  \/ m<13>  \/ m<14>  \/ m<15>
该命题公式的主合取范式:
M<4> /\ M<5>  /\ M<6>  /\ M<8>  /\ M<9>  /\ M<10>
```

3.2.2 健壮性测试


请输入合法的命题公式:
(a~b
括号不匹配
输入的公式不合法，请重新输入!
请输入合法的命题公式:
a&
操作符与操作数数量不匹配
输入的公式不合法，请重新输入!
请输入合法的命题公式:
|a|b
操作符与操作数数量不匹配
输入的公式不合法，请重新输入!
请输入合法的命题公式:
a#b
操作符与操作数数量不匹配
输入的公式不合法，请重新输入!
请输入合法的命题公式:
ab&c
操作符与操作数数量不匹配
输入的公式不合法，请重新输入!
请输入合法的命题公式:

4 心得体会

在完成此项目的过程中，我深刻体会到了离散数学与编程实践相结合的重要性。这个项目不仅让我复习了命题逻辑的基本概念，如命题逻辑联结词、真值表、主析取范式和主合取范式，而且还锻炼了我的编程技能，尤其是在处理复杂逻辑和数据结构方面。

通过编写这个逻辑运算软件，我学会了如何定义和实现自定义的数据结构，比如使用 map 来存储运算符的优先级和命题变量的索引。这些数据结构对于程序的逻辑清晰和高效运行至关重要。我也体会到了 STL 容器的便利性，它们大大简化了代码的编写和调试过程。

在实现逻辑运算的过程中，我深入了解了栈这一数据结构的重要性。利用栈来处理运算符的优先级和括号匹配问题，让我更加熟悉了“后进先出”的概念，并学会了如何在实际问题中应用这一概念。

代码中的递归和迭代逻辑让我意识到了算法设计的重要性。我学会了如何通过逻辑推理和逐步调试来优化算法的性能，尤其是在处理大型数据集时。此外，我也学会了如何通过用户输入来动态构建和计算真值表，这不仅提高了程序的灵活性，也加深了我对逻辑运算的理解。

在用户交互方面，我学会了如何设计友好的用户界面和提示信息，这对于提高用户体验至关重要。通过不断地测试和改进，我学会了如何编写健壮的代码来处理非法输入和异常情况。

总的来说，这个项目是一个宝贵的学习经历，它不仅提高了我的编程能力，也加深了我对离散数学的理解。我相信这些知识和技能将在我未来的学习和工作中发挥重要作用。