****

**Tongji University**

**离散数学实验报告**

**001-**命题逻辑联接词、真值表、主范式

专 业： 软 件 工 程

指导教师： 唐 剑 锋

学 号： 2 1 5 3 0 6 1

姓 名： 谢 嘉 麒

* 1. **命题逻辑联接词、真值表、主范式**
  2. **实验简介**
     1. **对于合取、析取、条件和双向条件的分析**

本题要求从键盘输入两个命题变元 P 和 Q 的真值，求它们的合取、析取、条件和双向条件的真值。（A）。

* + 1. **对于命题真值表和主范式的分析**

本题要求任意一个命题公式的真值表（B），并根据真值表求主范式（C）。

* 1. **实验目的**

本次实验主要学习命题逻辑中的逻辑联结词，真值表，并学会提取主范式等操作。在加深对于离散数学概念的理解的同时，进一步解决实际问题，并将其转化为编程思想，实现算法及程序设计和应用。最后通过实验报告进行反馈。

* 1. **实验要求与分析**
     1. **逻辑联接词的运算**

本题目要求用C++语言实现程序化的合取，析取，条件和双向条件的运算。经过笔者分析，可以采用不同函数分别实现这几种逻辑联接词，并利用其中的相似性和互通性进行代码简化和代码复用，总体实现对于逻辑联接词的代码化。

* + 1. **求任意一个命题公式的真值表**

本实验运用C/C++语言实现输入任意一个命题公式，运行程序得到其真值表。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。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。算法逻辑如下：

（1）将二进制加法模拟器赋初值0。

（2）计算模拟器中所对应的一组真值指派下合式公式的真值。

（3）输出真值表中对应于模拟器所给出的一组真值指派及这组真值指派所对应 的一行真值。

（4）产生下一个二进制数值，若该数值等于 2 n -1，则结束，否则转（2）。

1. **整体设计思路**
   1. **算法设计原理**
      1. **逻辑联接词的运算**
2. **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。

**真值分析：**只有当两个命题变项 P = TRUE, Q = TRUE 时方可 P∧Q =TRUE, 而 P、 Q 只要有一方为 FALSE 则 P∧Q = FALSE。

1. **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。

**真值分析：**只有当两个命题变项 P = FALSE, Q = FALSE 时方可 P∨Q =FALSE, 而 P、 Q 只要有一为 TRUE 则 P∨Q = TRUE。

1. **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。

**真值分析：**只有当两个命题变项 P = TRUE, Q = FALSE 时方可 P→Q =FALSE, 其余均为 TRUE。

1. **双条件：**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。

**真值分析：**当两个命题变项 P = TRUE, Q =TRUE 时方可 P←→Q =TRUE, 其余均为 FALSE。

* + 1. **求任意一个命题公式的真值表**

1. **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。
2. **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。
3. **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。
   1. **程序逻辑设计**
      1. **逻辑联接词的运算**

总体上通过类方式进行实现。

1. 首先，需要考虑进行输入输出的错误处理。根据命题概念，当输入内容不为0/1时，需要给出输入错误提示并跳转重新输入。本过程可以通过while循环结合cin.函数清空缓冲区进行实现，保证命题的真假输入正确。
2. 其次，考虑命题变元之间的运算处理。因C++语言库中包括“！”“||”“&&”等运算符，其功能与逻辑联接词等价，故可在实现合取、析取时直接调用。而对于条件、双条件的实现，可以转化为P→Q=（！P）||Q及P←→Q=（（！P）||Q）∧（（！Q）||P）的形式进行运算。
3. 最后，实现对于结果的输出，以及提供端口允许用户判断是否继续运行。
   * 1. **求任意一个命题公式的真值表**

总体上运用栈的方式实现。

1. 首先，输入一个合理的式子。
2. 其次，从式子中查找出变量的个数，开辟一个二进制函数，用来生成真值表。
3. 最后，用函数运算，输出结果，并根据结果归类给范式，最后输出范式。
4. 对于函数部分，主要是3个函数，一个为真值表递加函数，通过二进制的加法原 理递进产生，一个为分级运算函数，这个函数是通过判断括号，选出最内级括号 的内容执行运算函数，这样一级级向外运算，最后得出最终结果，剩下一个为主 运算函数，按照运算符号的优先级按顺序进行运算。
5. **本实验所用的数据结构**
   1. **逻辑联接词的运算**

本程序主要使用类进行封装和实现。在头文件中定义ProLogic类，并在相应.cpp文件中实现类成员函数。

其中，ProLogic类定义如下：

class ProLogic {

public:

//构造函数

ProLogic(int a = -1, int b = -1) :P(a), Q(b) {};

//合取

bool Conjunction();

//析取

bool Extraction();

//条件

bool Condition();

//双向条件

bool Bicondition();

//菜单与输入

void MenuInput();

//输出结果

void Output();

private:

int P;

int Q;

};

主类定义完成后，在.cpp文件中实现合取（Conjunction）、析取（Extraction）、条件（Condition）和双条件（Bicondition）的函数，并且为了保证程序完整性，实现菜单及输入函数（MenuInput）、输出函数（Output）。

* 1. **求任意一个命题公式的真值表**

本程序主要采用栈的数据结构。

通过定义不同运算符的优先级来保证与抽象概念相符合，在此基础上运用栈的数据结构，根据“Last In First Out”的特性进行命题逻辑联接词的运算。具体解释如下：

1. 运算数/运算符分别进各自的栈。
2. 若进栈的运算符优先级高于或等于（右结合）栈顶运算符，则进栈。
3. 若进栈的运算符优先级低于或等于（左结合）栈顶运算符，则先将栈顶运算符计算完成，计算数为运算数栈的两个元素，运算符及运算数出栈，运算结果进栈。
4. 重复以上步骤直到运算符栈为空，运算数栈只剩余一个元素，否则认为语法错误。

运算符优先级定义如下：

priority['('] = 6;

priority[')'] = 6;

priority['!'] = 5;

priority['&'] = 4;

priority['|'] = 3;

priority['^'] = 2;

priority['~'] = 1;

priority['#'] = 0;

1. **核心算法及问题**
   1. **对于输入正确性的判断**

本次试验需要对于输入命题变元真假的正确性（输入是否为0/1）进行判断，其中涉及到C++语言输入流特点以及输入缓冲区的处理，尝试后笔者选择运用while循环结合cin.clear()和cin.ignore()两个函数进行处理。将不为0/1的输入判误，清理缓冲区后提示客户重新输入，使程序更加健壮。

同时在实验BC中，通过在check函数中判断此时读入的数据是否符合逻辑来进行错误处理。

以P的真假输入举例：

while (1) {

cout << endl << " 请输入P的值（0或1）,以回车结束:";

cin >> P;

if (!cin.good()||(P!=1&&P!=0)) {

cin.clear();

cin.ignore(2147483648, '\n');

cout << endl << "P的值输入有误,请重新输入!";

continue;

}

else {

break;

}

}

* 1. **对于逻辑联接词优先级的处理**

为了解决顺序运算会导致逻辑联接词优先级被忽视导致出错的问题，本次实验经过探索，选择了为逻辑联接词自定义优先级以满足栈运算的方式。通过这种方式进行运算优先程度的判断和命题的提取，体现真值表运算的先后。

1. **心得体会**
   1. **强化对于相关知识的运用**

本次实验加深了笔者对于命题逻辑联接词的理解，强化了运用逻辑解决问题的能力，同时进一步学习合取、析取、条件、双条件、真值表、主范式的提取等知识点，提高了熟练程度。

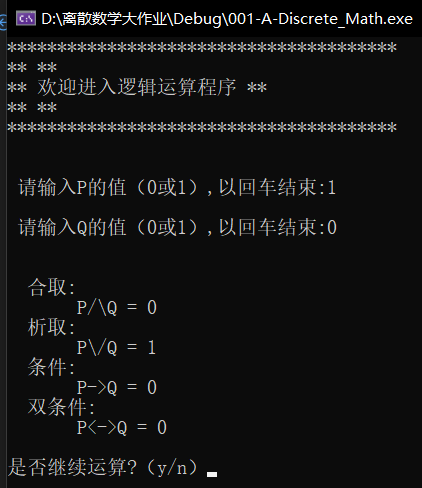
* 1. **加深对于命题逻辑联接词的优先级理解**

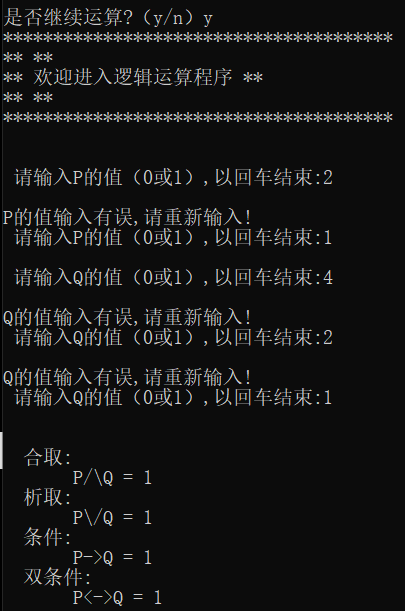
通过本次实验，笔者学会了处理优先级相关问题的方法，进一步理解了逻辑式运算中各命题逻辑联接词的优先级和运算特点，实现了用程序体现并完成相关逻辑运算，强化了概念，提高编程能力。

* 1. **优化对于实验报告的书写**

本次实验，笔者在撰写报告期间较为详细地学习了报告的书写方法，office软件的熟练运用，并且抓大取小，找到重点，不缺不漏，在文档撰写方面的能力有显著提高。

1. **程序测试**







1. **附件：核心代码展示**
   1. **逻辑联接词的运算**

bool ProLogic::Conjunction()

{

return P && Q;

}

bool ProLogic::Extraction()

{

return P || Q;

}

bool ProLogic::Condition()

{

return (!P) || Q;

}

bool ProLogic::Bicondition()

{

return ((!P) || Q) && ((!Q) || P);

}

void ProLogic::MenuInput()

{

cout << "\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*" << endl; //标语

cout << "\*\* \*\*" << endl;

cout << "\*\* 欢迎进入逻辑运算程序 \*\*" << endl;

cout << "\*\* \*\*" << endl;

cout << "\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*" << endl << endl;

while (1) {

cout << endl << " 请输入P的值（0或1）,以回车结束:";

cin >> P;

if (!cin.good()||(P!=1&&P!=0)) {

cin.clear();

cin.ignore(2147483648, '\n');

cout << endl << "P的值输入有误,请重新输入!";

continue;

}

else {

break;

}

}

while (1) {

cout << endl << " 请输入Q的值（0或1）,以回车结束:";

cin >> Q;

if (!cin.good() || (Q != 1 && Q != 0)) {

cin.clear();

cin.ignore(2147483648, '\n');

cout << endl << "Q的值输入有误,请重新输入!";

continue;

}

else {

break;

}

}

return;

}

void ProLogic::Output()

{

cout << Conjunction() << endl;

Extraction();

Condition();

Bicondition();

return;

}

* 1. **求任意一个命题公式的真值表**

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。同时，还可以返回指定命题变项的下标

{

Map\_ic::iterator it = pSet.begin();

while (it != pSet.end())

{

if (it->second == p)

{

return it->first;

}

it++;

}

return -1;

}

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)

{

Map\_ic proposition;

int n\_proposition = 0;

for (int i = 0; i < formula.length(); i++)

{

char c = formula[i];

if ((c >= 'a' && c <= 'z') || (c >= 'A' && c <= 'Z'))

{

//此时遍历所有命题变项

int r = findProposition(proposition, c);

//cout << r << endl;

if (r == -1)

{

//说明该命题变项尚未被遍历过

proposition[n\_proposition] = c;

n\_proposition++;

}

}

else if (!priority.count(c))

{

cout << c << " is undefined!" << endl;

exit(2);

}

}

return proposition;

}

Map\_ii toBinary(int n\_proposition, int index) //该函数返回命题变项的二进制(0或1)取值

{

Map\_ii result;

for (int i = 0; i < n\_proposition; i++)

{

int r = index % 2;

result[n\_proposition - 1 - i] = r;

index = index / 2;

}

return result;

}

int pow2(int n) //该函数返回指定数字的二次方的值

{

if (n == 0)

return 1;

else

return 2 \* pow2(n - 1);

}

int calculate(string formula, Map\_ic pSet, Map\_ii value) //该函数返回给定0或1组合的运算结果

{

stack<char> opter;

stack<int> pvalue;

opter.push('#');

formula = formula + "#";

for (int i = 0; i < formula.length(); i++)

{

char c = formula[i];

if ((c >= 'a' && c <= 'z') || (c >= 'A' && c <= 'Z'))

{

pvalue.push(value[findProposition(pSet, c)]);

}

else

{

//此时遍历的是运算符

char tmp = opter.top();

if (priority[tmp] > priority[c])

{

while (priority[tmp] > priority[c] && tmp != '(')

{

check(pvalue, opter);

tmp = opter.top();

if (tmp == '#' && c == '#')

{

return pvalue.top();

}

}

opter.push(c);

}

else

opter.push(c);

}

}

return -1;

}

void check(stack<int> &value, stack<char> &opter) //该函数返回两个命题变项(取值0或1)的各种运算结果(0或1)

{

int p, q, result;

char opt = opter.top();

switch (opt)

{

case '&':

p = value.top();

value.pop();

q = value.top();

value.pop();

result = p && q;

value.push(result);

opter.pop();

break;

case '|':

p = value.top();

value.pop();

q = value.top();

value.pop();

result = p || q;

value.push(result);

opter.pop();

break;

case '!':

p = value.top();

value.pop();

result = !p;

value.push(result);

opter.pop();

break;

case '^':

q = value.top();

value.pop();

p = value.top();

value.pop();

result = !p || q;

value.push(result);

opter.pop();

break;

case '~':

p = value.top();

value.pop();

q = value.top();

value.pop();

result = (!p || q) && (p || !q);

value.push(result);

opter.pop();

break;

case '#':

break;

case '(':

break;

case ')':

opter.pop();

while (opter.top() != '(')

{

check(value, opter);

}

if (opter.top() == '(')

{

opter.pop();

}

break;

default:

break;

}

}