**命题逻辑联接词、真值表、主范式**

**1、题目简介**

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。

题目要求：

问题A：本实验要求利用C/C＋＋语言，实现二元合取、析取、条件和双向条件表达式的计算。充分利用联接词和逻辑运算符之间的相似性来实现程序功能。

问题BC：本实验要求利用C/C＋＋语言，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。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2n-1，则结束，否则转（2）。

**2、解题思路**

（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 = T, Q = T时方可P∧Q =T, 而P、Q只要有一方为F则P∧Q = F。P∧Q可用来表示日常用语P与Q, 或P并且Q。

（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 = F, Q = F时方可P∨Q =F, 而P、Q只要有一为T则P∨Q = T。P∨Q可用来表示日常用语P或者Q。

（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 = T, Q = F时方可P→Q =F, 其余均为T。

（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。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 = T, Q =T时方可P←→Q =T, 其余均为F。

（5）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。

（6）主范式：

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。

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。

**3、数据结构设计**

这个作业的核心是表达式的计算，所以需要用到栈这个数据结构，由于我解题的算法是将表达式转化为逆波兰式，然后通过逆波兰式计算结果，所以需要一个二叉树存储逆波兰式，同时由于一个运算符既有符号以及优先级所以我还定义了一个结构体来表示运算符和运算数，而且由于是对式子中的字母一一赋值所以有设置了一个字母结构体，存储字母的值以及符号。

具体代码如下：

struct Op

{

char ch;

int piorty = -1;

};

struct letter

{

char ch;

int value;

};

struct Node

{

Node\* Lnode = NULL;

Node\* Rnode = NULL;

Op\* op;

};

class Mystack

{

public:

bool pop();

bool push(Node\* op);

Node\* gettop() { return top == -1 ? NULL : ops[top]; }

void putout() {

for (int i = 0; i <= top; i++)

cout << char(((ops[i]->op->ch<=1) ? (ops[i]->op->ch+'0'): ops[i]->op->ch));

cout << "#" << endl;

}

private:

Node\* ops[20];

int top = -1;

};

class Mytree

{

public:

void init();

void calculate();

void getop(Op\*& op, int& i);

char findnum(char ch);

int getoutcome();

void setletters(int i);

void calculatenums(Mystack& stack1, Mystack& stack2);

void putoutexp();

Node\* getroot() { return root; }

private:

Node\* root;

char exp[100];

letter letters[20];//存储字符

int letternum = 0;

int xq[2000];//存储合取范式

int x = 0;

int hq[2000];//存储析取范式

int h = 0;

};

**4、核心算法**

算法总体思路：

1、读取用户输入的表达式

2、依次给变量赋值

3、赋值之后开始遍历表达式

使用getop()函数读取一个字符，此时需要两个栈分别用于存放运算数和运算符，如果读取到的是运算数就将运算数放到栈中，如果读取到运算符时，将这个运算符与栈顶的运算符比较，如果栈为空或者优先级比栈顶元素高，运算符进栈，反之则，清栈直到满足第一点的条件。

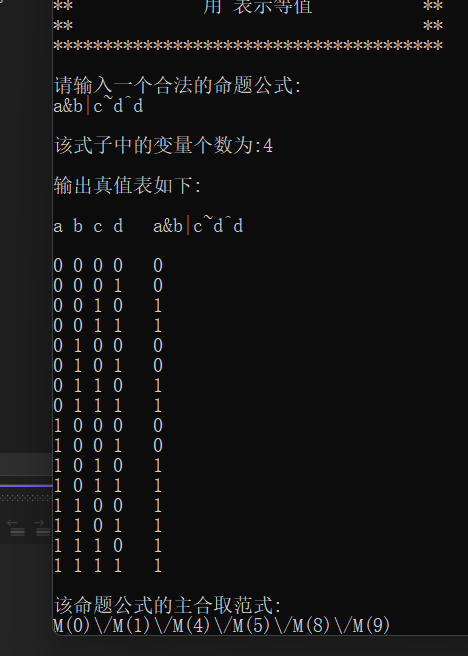
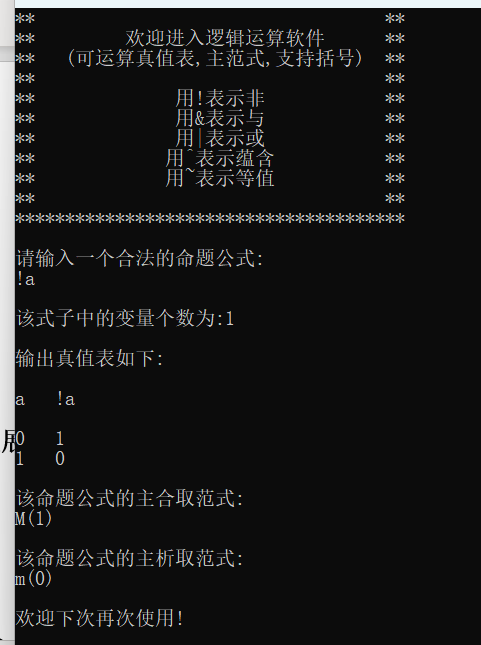
清栈过程：取运算符栈顶元素，和栈顶的两个运算数，执行运算符的计算然后将结果存入运算数栈中。

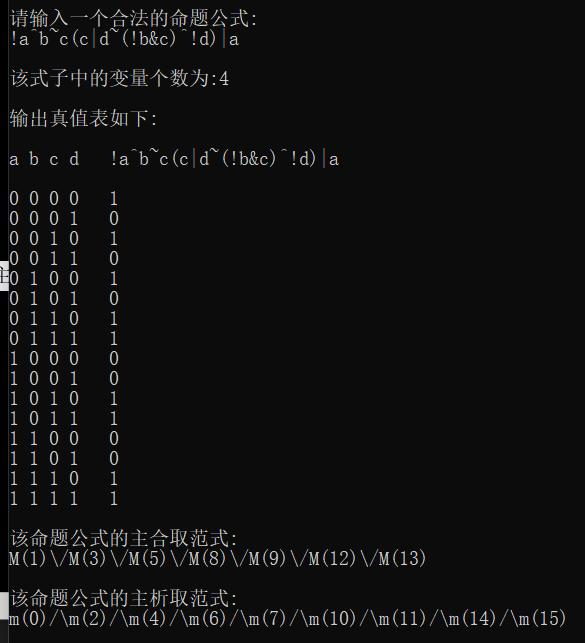
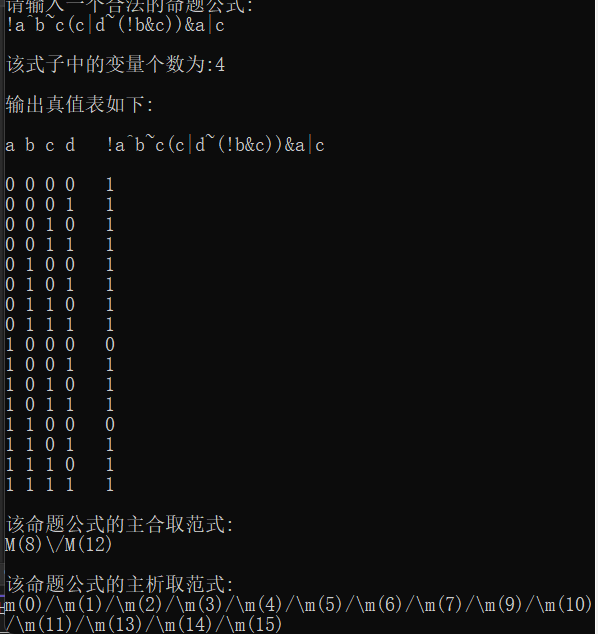
4、遍历完表达式之后，再次执行清栈，得出最后结果

5、若结果为0，则记录合取，结果为1则记录析取。

6、输出主析取范式主合取范式。

**结果展示：**

****

**5、心得体会**

理论与实际的桥梁：开始时，我对逻辑运算，真值表，主范式有了基本的理解，但在深入研究它们的具体应用时，我才真正意识到这些性质在计算机科学和其他领域的广泛应用。这提醒我，理论知识不仅仅是抽象的概念，当正确应用时，它们具有强大的实用价值。

细节决定成败：在算法实现阶段，我遇到了许多预料之外的挑战。某些理论在实际操作中并不容易实现，需要反复调试和优化，比如数据结构中学习到的栈这个数据结构可以很方便的处理括号这种特殊的运算方式，但是实际操作的时候由于题目的特殊性，我需要单独定义运算符的优先级，在调式过程中给我造成很大困扰，这教会了我，即使是最小的细节，也可能对项目的成功产生重大影响。

持续学习的必要性：尽管我之前已经有了一些离散数学的背景知识，但在项目进行中，我仍然遇到了许多我之前不熟悉的问题。我必须不断学习和研究，以确保项目的顺利进行。

总的来说，这个项目不仅加深了我的专业知识，还锻炼了我的团队合作和项目管理能力。我深深地意识到，无论是理论研究还是实际应用，都需要持续的努力和不断的学习。这将是我未来职业生涯中的宝贵经验。