《离散数学》课程实验报告

1-命题逻辑联接词、真值表、主范式

一、题目简介

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。

1. 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）。

2. 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）。

二、原理与方法

1. 逻辑联接词的运算

本实验要求利用 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。

2. 求任意一个命题公式的真值表

要求利用 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）。

三、解题思路

①A 题：首先是对各个输入量的处理，要确定输入的为 0 或 1，否则则为出错，接下来就是运算处理，在 C 语言中本身支持的有与、或、非这三种，如果不是与、或、非的可以通过转化变为与、或、非的形式，具体流程为：

1.先后输入 p 和 q 的值，并进行错误判断及处理。若无错误，则继续步骤 2。

2.按照公式进行运算，并按照格式输出合取、析取、条件和双向条件的结果。

3.判断是否继续运算，若是则返回步骤 1 的操作；若否则结束运算程序。

②B、C 题：首先是输入一个合理的式子，然后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，然后用函数运算，输出结果，并根据结果归类给范式，最后输出范式。主要有 3 个函数：

a.主运算函数，按照运算符号的优先级按顺序进行运算；

b.分级运算函数，这个函数是通过判断括号是否为最内级括号，从而对最内级内容执行运算函数，若非最终结果，则再向外一级检查括号。一级级向外运算，最后得出最终结果；

c.真值表递加函数，通过二进制的加法原理递进产生。

1.利用系统头文件 map，设置好可能用到的运算符优先级的大小。

2.输入式子，并进行错误处理。

注意：特别检查左右括号是否成对、是否存在&、|、^互相任意相连的情况以及表达式的最后一位是否是非运算符。

3.遍历式子，计算不同命题变项的个数。

4.计算各命题变项分别取 0、1 时的组合并存储。从 0 遍历到 2^n-1，经过模 2 除 2 求其二进制，并存储即可。

5.计算命题公式各行(0 或 1)的运算结果。

（1）设置两个栈，num 中间结果、opter 存储联结词。

（2）从左至右扫描合式公式：遇到命题变项时将其压入 num；遇到联结词时，将其与 opter 栈顶元素进行优先级比较。

（3）遇到括号时，左括号‘（’直接压入 opter；遇右括号‘）’将 opter 中左括号前面的元素依次弹出压入 num，并将左括号 pop 出栈。

（4）到达表达式最右端用“#”标记。

（5）将 opter 中剩余元素依次弹出压入 num 中，最后将 num 中元素依次弹出得到最终的后缀表达式。

6.根据运算结果输出主析取范式和主合取范式。若二进制组合经过转换对应出十进制 i 的运算结果为 1，则有主析取范式的极小项 m_i；若对二进制组合经过转换对应出十进制 i 的运算结果为 0，则有主合取范式的极大项 M_i。

四、所用数据结构和核心算法

1.栈 Stack

栈是限定仅在表尾进行插入或者删除的线性表。对于栈来说，表尾端称为栈顶（top），表头端称为栈低（bottom）。不含元素的空表称为空栈。作为一种数据结构，栈是一种只能在一端进行插入和删除操作的特殊线性表，所以栈又被称为后进先出的线性表（简称 LIFO:Last in, First out.结构）。它按照后进先出的原则存储数据，先进入的数据被压入栈底，最后的数据在栈顶，需要读数据的时候从栈顶开始弹出数据(最后一个数据被第一个读出来)。栈具有记忆作用，对栈的插入与删除操作中，不需要改变栈底指针。

本项目使用了 stack<char> opter 字符型的栈，用于存储表达式中读取的字符，以及 stack<int> num 整型栈，用于存储表达式中各命题变项的运算结果。

2.关联式容器 map

关联式容器容器在存储元素值的同时，还会为各元素额外配备一个值（又称为键），其本质是一个 C++基础数据类型或者自定义类型的元素。它的功能是在使用关联式容器的过程中，如果已知目标元素的键的值，则直接通过该键就可以找到目标元素，而无需再通过遍历整个容器的方式。使用关联式容器存储的元素，都是一个一个的“键值对”（ <key,value> ），并且元素默认会根据各元素的键值的大小做升序排序。

本项目使用了 C++关联式容器之一——map，用以对应<符号-优先级>、<命题变项-是否被被遍历>、<十进制对应的二进制每位序号-值>。

五、相关代码

A:

```
/*2151133 孙韩雅*/
#include <stdio.h>
#include <iostream>
using namespace std;

int main() {
	bool p = false, q = false;
	char choice_continue;//用于存放是否继续运算的选择
	cout << "**************************************" <<
endl;
	cout << "**                                  **" <<
endl;
	cout << "**        欢迎进入逻辑运算程序      **" <<
endl;
	cout << "**                                  **" <<
endl;
	cout << "**************************************" <<
endl << endl;
PART1:
```

```
        cout << "请输入P的值（0或1）:";
        cin >> p;
        if (cin.fail()) {
                cout << "P的值输入有误,请重新输入!" << endl;
                cin.clear();
                cin.ignore(100, '\n');
                goto PART1;
        }
PART2:
        cout << "请输入Q的值（0或1）:";
        cin >> q;
        if (cin.fail()){
                cout << "Q的值输入有误,请重新输入!" << endl;
                cin.clear();
                cin.ignore(100, '\n');
                goto PART2;
        }

        cout << endl;
        cout << "合取: P/\\Q =" << (p && q )<< endl; //输出合取结果
        cout << "析取: P\\/Q =" << (p || q )<< endl; //输出析取结果
        cout << "条件: P->Q =" <<((!p) || q) << endl; //输出条件结果
        cout <<"双条件: P<->Q =" << (((!p) || q) && ((!q) || p))<< endl; //输出双条件结果

PART3:
        cout << endl << "是否继续运算?（y/n）";  //是否继续操作
        cin >> choice_continue;
        if (choice_continue == 'y' || choice_continue == 'n'|| choice_continue == 'Y'|| choice_continue == 'N'){
                if (choice_continue == 'y'|| choice_continue == 'Y')
                        goto PART1;  //返回P值的输入
                else
                        cout << "程序已退出!" << endl;
        }
        else{
                cout << "输入错误,请重新输入!" << endl;
                cin.clear();
                cin.ignore(100, '\n');
                goto PART3; //回到询问是否继续操作
        }
        return 0;
}
```

B&C:

```
/*2151133 孙韩雅*/
#include <iostream>
#include <string>
#include <map>//利用map容器反映的符号向优先级的映射
#include <stack>//栈的头文件
using namespace std;

typedef map<char, int> M_ci;//为map<char, int>起别名为Map_ci
typedef map<int, char> M_ic;//为map<int, char>起别名为Map_ic
typedef map<int, int> M_ii;//为map<int, int>起别名为Map_ii

M_ci priority;//首先定义一个运算符优先级map,类型是map<char, int>
bool check_fomula (string a) {//检查输入公式的正确性
        /*检查左右括号是否成对、运算符非法相连*/
        int left = 0, right = 0;
        for (unsigned int i = 0; i < a.length(); i++) {//对式子进行遍历
                if (left < right) //右括号数不能多于左括号数
                        return false;
                if (a[i] == '(') {
                        left++;
                        if (a[i + 1] =='&' || a[i + 1] =='|' || a[i + 1] == '^')
                                //不能存在‘(&’ ‘(|’ ‘(^’的情况
                                return false;
                }
                else if (a[i] == ')') {
                        right++;
                        if (a[i - 1] =='&' || a[i - 1] =='|' || a[i - 1] == '^')
                                //不能存在‘&)’ ‘|)’ ‘^)’的情况
                                return false;
                }
        }
        if (left != right) //左右括号是否成对
                return false;
        if (a[a.length() - 1] =='&' || a[a.length() - 1] =='|' || a[a.length() - 1] =='!' || a[a.length() - 1] =='~' || a[a.length() - 1] == '^')
```

```
            //检查表达式的最后一位
            return false;
        return true;//若上述都没问题，则返回真值
}
int find_proposition(M_ic pSet, char p){//计算不同命题变项的个数
        M_ic::iterator it = pSet.begin();//map 迭代器 it 指向 map 的开头
        while (it != pSet.end()) {
            if (it->second == p) //判断 it 指向的 vaule 是否为 p
                return it->first;//若为 p，则返回指定命题变项的下标
            it++;//迭代器 it 右移
        }
        return -1;//遍历完，没有找到，则返回-1
}
M_ic get_proposition(string formula){ //该函数返回所输入公式中的命题变项(不包括运算符)
        M_ic proposition;
        int n_proposition = 0;//记录不同命题变项的个数
        for (unsigned int i = 0; i < formula.length(); i++){//逐个字符遍历式子
            char c = formula[i];//将遍历到的字符存在字符型变量 c 中
            if ((c >='a' && c <='z') || (c >='A' && c <='Z')) {//判断是否是英文大小写字母
                if (find_proposition(proposition, c) == -1) //该命题变项尚未在式子中出现过
                    proposition[n_proposition++] = c;
            }
            else if (!priority.count(c)) { //c 在 priority 中不存在，即不是运算符
                cout << c << " is undefined!" << endl;//即既不是字母，又不是运算符。则输出该字符未定义的提示
                exit(2);//退出程序
            }
        }
        return proposition;
}
M_ii translate(int n_proposition, int value){//该函数返回命题变项的二进制取值
        M_ii result;
        for (int i = 0; i < n_proposition; i++) {
            result[n_proposition - 1 - i] = value % 2;
            value = value / 2;
        }
        return result;
}
int power(int n) {//返回 2^n 的值
        if (n == 0)
            return 1;
        else
            return 2 * power(n - 1);//递归 n-1 次
}
void stack_pop(int& p, int& q,stack<bool>value) {//为了后续函数遍历，将出栈操作放入一个函数
        p = value.top();//取整数栈栈顶元素
        value.pop();//出栈
        q = value.top();//取整数栈栈顶元素
        value.pop();//出栈
}
void check(stack<bool>& value, stack<char>& opter) {//返回两个命题变项分别取 0 或 1 时的各种组合运算结果
        int p, q, result;
        char opt = opter.top();//字符栈的栈顶运算符
        switch (opt) {
        case '&':
            stack_pop(p, q, value);
            result = p && q;//与运算
            value.push(result);//将结果进栈
            opter.pop();//符号栈栈顶元素出栈
            break;
        case '|':
            stack_pop(p, q, value);
            result = p || q;//或运算
            value.push(result);//将结果进栈
            opter.pop();//符号栈栈顶元素出栈
            break;
        case '!':
            p = value.top();//取整数栈栈顶元素
            value.pop();//出栈
            result = !p;//非运算
            value.push(result);//将结果进栈
            opter.pop();//符号栈栈顶元素出栈
            break;
        case '^':
            stack_pop(p, q, value);
```

```
            result = !p || q;//蕴含运算，等价为~p∪q
            value.push(result);//将结果进栈
            opter.pop();//符号栈栈顶元素出栈
            break;
        case '~':
            stack_pop(p, q, value);
            result = (!p || q) && (p || !q);//等值运算，p蕴含q且q蕴含p。
            value.push(result);//将结果进栈
            opter.pop();//符号栈栈顶元素出栈
            break;
        case ')':
            opter.pop();
            while (opter.top() != '(')
                check(value, opter);//递归
            if (opter.top() == '(')
                opter.pop();//出栈
            break;
        default:
            break;
        }
}
int calculate(string formula, M_ic pSet, M_ii value) { //返回给定整体命题变项组合的运算结果
    stack<char> opter;//字符栈
    stack<bool> num;//整数栈
    opter.push('#');//压入栈底元素标识
    formula = formula + "#";//公式结尾标识符
    for (unsigned int i = 0; i < formula.length(); i++) {
        char c = formula[i];//将遍历到的字符存在c中
        if ((c >='a' && c <='z') || (c >='A' && c <='Z')) //若是英文大小写字母
            num.push(value[find_proposition(pSet, c)]);
        else {
            char tmp = opter.top();
            if (priority[tmp] > priority[c]) {//若栈顶元素优先级大于遍历到的符号的优先级
                while (priority[tmp] > priority[c] && tmp != '(') {
                    check(num, opter); //返回两个命题变项(取值0或1)的各种运算结果
                    tmp = opter.top();
                    if (tmp == '#' && c == '#') //若为#，即栈底或式子结尾
                        return num.top();//返回整数栈的栈顶元素
                }
                opter.push(c);//符号入栈
            }
            else
                opter.push(c);//符号入栈
        }
    }
    return -1;
}

int main() {
    //运用map<char, int>的类型，将符号和优先级对应起来
    priority['('] = 6;
    priority[')'] = 6;
    priority['!'] = 5;
    priority['&'] = 4;
    priority['|'] = 3;
    priority['^'] = 2;
    priority['~'] = 1;
    priority['#'] = 0;
    //菜单显示
    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";
    cout << "**              用!表示非           **\n";
    cout << "**              用&表示与           **\n";
    cout << "**              用|表示或           **\n";
    cout << "**             用^表示蕴含          **\n";
    cout << "**             用~表示等值          **\n";
    cout << "**                                  **\n";
    cout << "**************************************\n\n";
    cout << "Please enter a legitimate proposition formula:" << endl;
    string formula;
    while (1) {
        cin >> formula;
        if (check_fomula(formula)) //检查式子是否合法
            break;
        else
```

```
                cout <<"表达式有误，请重新输入！" << endl;
        }
        M_ic proposition_set = get_proposition(formula);//获取包含所输入公式中的命题变项的 map(不包括运算符)
        //输出该式子中的变量个数
        cout <<"该式子中的变量个数为:" <<proposition_set.size() << endl << endl;
        cout << "输出真值表如下：" << endl;
        /*打印表头*/
        for (unsigned int i = 0; i < proposition_set.size(); i++)
                cout << proposition_set[i] << "       ";
        cout << formula << endl;//输出公式

        int* p=NULL;
        p = (int*)malloc(sizeof(int) * power(proposition_set.size()));   //该数组依次存放命题公式的各行(0 或 1)的运算结果的值
        for (int i = 0; i < power(proposition_set.size()); i++){//循环 2 的命题个数的次方次，因为有这么多种二进制组合方式
                M_ii bina_set = translate(proposition_set.size(), i);//返回命题变项的二进制取值
                for (unsigned int j = 0; j < bina_set.size(); j++)
                        cout << bina_set[j] << endl;//输出一组命题变项 0、1 的组合
                int result = calculate(formula, proposition_set, bina_set); //返回给定命题变项组合的运算结果
                *(p + i) = result;//存放命题公式的各行的运算结果的值
                cout << result << endl;
        }
        int m = 0, M = 0;//记录范式中极小项极大项的个数
        cout << "该命题公式的主析取范式：" << endl;
        for (int i = 0; i < power(proposition_set.size()); i++)
{
                if (*(p + i) == 1) {
                        if (m == 0)
                                cout << "m<" << i << ">";//特判第一个极小项，输出时前面不用有\/符号
                        else
                                cout << " \\/ m<" << i << "> ";
                        m++;//极小项个数加一
                }
        }
        if (m == 0)
                cout << "0";//没有极小项
        cout << endl;
        cout << "该命题公式的主合取范式：" << endl;
        for (int i = 0; i < power(proposition_set.size()); i++)
{
                if (*(p + i) == 0) {//当对应的运算结果为 0 时
                        if (M == 0) //特判第一个极大项，输出时前面不用有/\符号
                                cout << "M<" << i << ">";
                        else
                                cout << " /\\ M<" << i << "> ";
                        M++;//极大项个数加一
                }
        }
        if (M == 0)
                cout << "0";//没有极大项
        cout << endl;
        free(p);//释放内存
        return 0;
}
```

六、体会与心得

这是离散的第一个项目，也是我认为最简单易懂的章节知识，但是放在代码实现的过程中还是会有磕磕绊绊。但是回头看看，这个项目的实现不仅帮助我很好的掌握命题逻辑连接词、真值表和主范式，还在代码方面给了我很大的提升技巧。

A 题相对而言比较简单容易实现，而 B、C 两道题在每个命题变项的获取和分析方面还是有一些难度的。我与同学们一起研读了示例代码，在网上查阅代码中所用到的数据结构和 C++功能，比如说关联式容器 map，在对应变量和键值时有很大的用处，不仅是在离散数学，我在数据结构的项目编写中也学会了使用 map，受益匪浅。

另外，该项目代码要达到完善，还是有许多细节值得注意的。就从输入公式的检查来说，用户会有许多不经意间导致的错误输入，这时我们就没有进行程序执行的必要。那么为了省时间，我们就得对输入的公式做一个前期的检查，也得排查许多方面，比如括号不成对、符号非法连接、非法符号结尾等等。以及各种内存的释放，到 main 函数执行完毕后，还是可

以进行一些清理，增加程序的清洁度。

当然，本次的编写不同于自己从0开始编代码，首先教会我在拿到一个代码后，要学会阅读并注释，这样才能帮助自己更好地理解它的逻辑，从而进行更深入的完善与健壮。