同济大学

离散数学说明文档

--命题逻辑联接词、真值表、主范式

姓　　名：胡正华

学　　号：2353741

任课教师：李　冰

1. 题目简介

1.1 题目要求

A 题：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。

B题：求任意一个命题公式的真值表。

C题：根据B中所得的命题公式的真值表求主合取范式和主析取范式。

其中，!表示非，|表示或，^表示蕴含，~表示等值。

1.2 题目样例

A题：

B、C题：

2. 解题思路

2.1 A题

依据离散数学的知识和C++的语法可知，求合取表达应使用&&，求析取表达应使用||,求条件表达应使用(!a)||b（蕴含等值式），求双条件表达应使用((!a) || b) && ((!b) || a)（等价等值式）。

2.2 B、C题

依据离散数学知识可知，求真值表即打印命题公式在所有可能的赋值下的真值的列表，含有n个命题变项的公式则有2^n个赋值。而求主范式则需要根据真值表来求。当求出该命题公式的真值表后，找出所有原公式的成真赋值，则所有成真赋值对应的十进制数为所有极小项m的角码，从而求得主析取范式；反之，找出所有原公式的成假赋值，则所有成假赋值对应的十进制数为所有极大项M的角码，从而求得主合取范式。

在设计程序时，需设计的函数有：获取公式中的命题、查找命题位置、求2的n次方、将数值转换为二进制、计算逻辑公式值和执行操作符对应的逻辑运算。

3. 代码实现

3.1 A题

用一个大小为4一维数组存放四个表达式的真值。核心代码如下：

```
1
2         // 计算逻辑运算结果
3         int a[4];
4         a[0] = i && j;  // 合取（AND）
5         a[1] = i || j;  // 析取（OR）
6         a[2] = (!i) || j;  // 条件（蕴含）
7         a[3] = ((!i) || j) && ((!j) || i);  // 双条件（等值）
8
9         // 输出结果
10        cout << "\n合取:\n      P/\\Q=" << a[0] << endl;
11        cout << "析取:\n      P\\/Q=" << a[1] << endl;
12        cout << "条件:\n      P→Q=" << a[2] << endl;
13        cout << "双条件:\n      P←→Q=" << a[3] << endl;
14
15        // 询问是否继续
16        cout << "\n是否继续运算?（y/n）: ";
17        cin >> s;
18        if (s == 'n') {
19            cout << "欢迎下次再次使用! Press any key to continue";
20            break;  // 退出程序
21        }
22        else if (s ≠ 'y') {
23            continue;
24        }
```

3.2 B、C题

3.2.1 定义逻辑运算符的优先级

```cpp
typedef map<char, int> Map_ci;
Map_ci priority;
priority['('] = 6;
priority[')'] = 6;
priority['!'] = 5;
priority['&'] = 4;
priority['|'] = 3;
priority['^'] = 2;
priority['~'] = 1;
priority['#'] = 0;
```

3.2.2 计算公式在所有赋值下的真值并存储

计算2的n次方。该函数用于计算有多少种赋值情况：

```cpp
int pow2(int n)
{
    if (n == 0)
        return 1;
    else
        return 2 * pow2(n - 1);
}
```

设计一个函数，其功能是查找命题p并返回其索引，实现方法为使用迭代器来从头遍历：

```cpp
int findProposition(Map_ic pSet, char p)
{
    Map_ic::iterator it = pSet.begin();
    while (it != pSet.end())
    {
        if (it->second == p)
        {
            return it->first;
        }
        it++;
    }
    return -1;
}
```

设计一个函数，其功能为从公式中提取命题变项并加入到proposition中。这个函数用来打印表头：

```
Map_ic getProposition(string formula)
{
    Map_ic proposition;
    int n_proposition = 0;
    for (unsigned int i = 0; i < formula.length(); i++)
    {
        char c = formula[i];
        if ((c ⩾ 'a' && c ⩽ 'z') || (c ⩾ 'A' && c ⩽ 'Z'))
        {
            int r = findProposition(proposition, c);

            if (r == -1)
            {
                proposition[n_proposition] = c;
                n_proposition++;
            }
        }
        else if (!priority.count(c))
        {
            cout << c << " is undefined!" << endl;
            exit(2);
        }
    }
    return proposition;
}
```

打印表头的代码如下：

```
string formula;
cin >> formula;
Map_ic proposition_set = getProposition(formula);
cout << "该式子中的变量个数为: " << proposition_set.size() << endl << "输出真值表如下: " << endl;
for (unsigned int i = 0; i < proposition_set.size(); i++)
{
    cout << proposition_set[i] << "\t";
}
cout << formula << endl;
```

设计将十进制整数转换为二进制的函数，并返回命题变项的二进制取值。这里存储的是其中一种赋值方式，相当于按位来存储这个二进制数：

```
typedef map<int, int> Map_ii;
Map_ii toBinary(int n_proposition, int index)
{
    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;
}
```

设计一个计算两个命题变项在一次运算后的结果函数，用switch-case的方式枚举各种运算符下的运算结果（由于代码较长，故此处直接复制）：

```
void check(stack<int>& value, stack<char>& opter)
{
    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;
    }
}
```

核心代码：计算公式函数：

```
int calculate(string formula, Map_ic pSet, Map_ii value)
{
    stack<char> opter;
    stack<int> pvalue;
    opter.push('#');
    formula = formula + "#";
    for (unsigned 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
        {
            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;
}
```

关键语句解释：

1. opter.push('#')：将一个特殊字符#入栈，作为运算符栈的初始符号，用于标记公式的结束。

2. formula = formula + "#"：通过在公式末尾加上#确保最后一个运算符能够处理。
3. if ((c >= 'a' && c <= 'z') || (c >= 'A' && c <= 'Z'))
   {
       pvalue.push(value[findProposition(pSet, c)]);
   }

如果当前字符c是命题变量（即字母），那么通过 findProposition(pSet, c)查找它在pSet中的索引，并根据映射value获取该命题变量的值（0或1）。然后将该值压入 pvalue 栈中。

3.2.3 打印真值表

```
int* m;
m = (int*)malloc(sizeof(int) * pow2(proposition_set.size()));
for (int i = 0; i < pow2(proposition_set.size()); i++)
{
    Map_ii bina_set = toBinary(proposition_set.size(), i);
    for (unsigned int j = 0; j < bina_set.size(); j++)
    {
        cout << bina_set[j] << "\t";
    }
    int result = calculate(formula, proposition_set, bina_set);
    *(m + i) = result;
    cout << result << endl;
}
```

3.2.4 求主合取范式和主析取范式

```
int n_m = 0, n_M = 0;
cout << "该命题公式的主合取范式：" << endl;
for (int i = 0; i < pow2(proposition_set.size()); i++)
{
    if (*(m + i) == 0)
    {
        if (n_M == 0)
        {
            cout << "M<" << i << ">";
        }
        else
        {
            cout << "\\/M<" << i << ">";
        }
        n_M++;
    }
}
if (n_M == 0)
{
    cout << "0";
}
cout << endl;
cout << "该命题公式的主析取范式：" << endl;
for (int i = 0; i < pow2(proposition_set.size()); i++)
{
    if (*(m + i) == 1)
    {
        if (n_m == 0)
        {
            cout << "m<" << i << ">";
        }
        else
        {
            cout << "/\\m<" << i << ">";
        }
        n_m++;
    }
}
if (n_m == 0)
{
    cout << "0";
}
```

4. 心得体会

这次实验工程量较大，不仅提升了我的编程能力，也加深了我对逻辑运算和数据结构的理解。

这次作业让我深入学习并实践了栈和映射的数据结构：

栈在处理逻辑公式解析和优先级问题时表现得非常高效。例如，利用两个栈分别存储操作数和运算符，使得复杂的嵌套公式能够按照正确的优先级被解析。

映射（map）在存储命题变量及其值时非常直观，比如将变量与其对应的值关联起来，便于快速查找和操作。

这次作业让我深刻体会到，面对复杂任务时，冷静分析、分解问题是关键。编程不仅是技术的体现，更是思维逻辑的艺术。