

**实验报告**

**（2019 / 2020学年第一学期）**

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| 课程名称 | 离散数学 | | | | | |
| 实验名称 | 利用真值表法求取主析取范式以及主合取范式 | | | | | |
| 实验时间 | 2020 | 年 | 10 | 月 | 28 | 日 |
| 指导单位 | 计算机学院计算机科学与技术系 | | | | | |
| 指导教师 |  | | | | | |

|  |  |  |  |
| --- | --- | --- | --- |
| 学生姓名 |  | 班级学号 |  |
| 学院(系) | 计算机学院 | 专业 | 信息安全 |

**实验报告**

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **实验名称** | 利用真值表法求取主析取范式以及主合取范式 | | | **指导教师** | 柯昌博 |
| **实验类型** | **验证** | **实验学时** | **4** | **实验时间** | 2020.10.28 |
| 1. **实验目的和要求**   内容：  编程实现用真值表法求取含n个变元的合式公式所对应的主析取范式和主合取范式。  要求：  能够列出合式公式的真值表并给出相应主析取和主合取范式。 | | | | | |
| 二、**实验环境(实验设备)**  硬件：微型计算机  软件：Windows 操作系统、Microsoft Visual C++6.0、C++ | | | | | |
| **三、实验原理及内容**  **主要数据结构和算法**  1.栈  2. 逆波兰式，也叫后缀表达式（将运算符写在操作数之后）  3.DFS（用来枚举n个命题变元的2^n种指派）  4.STL自带数据结构<map>,<set>等  **原理**  为什么要将看似简单的中序表达式转换为复杂的逆波兰式？原因就在于这个简单是相对人类的思维结构来说的，对计算机而言中序表达式是非常复杂的结构。相对的，逆波兰式在计算机看来却是比较简单易懂的结构。因为计算机普遍采用的内存结构是栈式结构，它执行先进后出的顺序。平常我们写的数学表达式a+b，就是一种中缀表达式，写成后缀表达式就是ab+。再举一个复杂的例子，中缀表达式(a+b)\*c-(a+b)/e的逆波兰式是ab+c\*ab+e/-。  **（1）中序表达式转后序表达式具体流程**  step1：初始化一个栈和一个后缀表达式字符串  step2：从左到右依次对中缀表达式中的每个字符进行以下处理，直到表达式结束  如果字符是‘（’，将其入栈  如果字符是数字，添加到后缀表达式的字符串中  如果字符是运算符，先将栈顶优先级高于该运算符的运算符出栈，添加到后缀表达式中，再将该运算符入栈。当‘（’在栈中时，优先级最低  如果字符是‘）’，将栈顶元素出栈，添加到后缀表达式中，直到出栈的是‘（’  step3：如果表达式结束，但栈中还有元素，将所有元素出栈，添加到后缀表达式中  **（2）后缀表达式的计算**  后缀表达式没有括号，运算符的顺序即为实际运算顺序，在求值过程中，当遇到运算符时，只要取得前两个操作数就可以立即进行计算。当操作数出现时，不能立即求值，需要先保存等待运算符。对于等待中的操作数而言，后出现的先运算，所以需要一个栈辅助操作。  后缀表达式的运算过程如下：  step1：设置一个栈  step2：从左到右对后缀表达式中的字符进行以下处理：  - 如果字符是数字，现将其转化为数字，然后入栈  - 如果字符是运算符，出栈两个值进行计算。计算结果入栈  - 重复以上步骤，直到后缀表达式扫描结束，栈中最后一个元素就是表达式的结果。  **（3）使用DFS枚举n个命题变元的2^n种指派**  **完整代码与注释**  #include <iostream>  #include <string>  #include<cstring>  #include <set>  #include <map>  using namespace std;  string org; //原式，后面将其改造成后缀表达式  string pcnf; //主合取范式  string pdnf; //主析取范式  int a, b, res; //a、b保存出栈的两个元素的值，res保存计算结果  int choose;  class SeqStack  {  public:  SeqStack(int mSize);  ~SeqStack();  bool Push(char x);  bool Pop();  char Top();  private:  int top;  char \*st;  int maxtop;  };    SeqStack::SeqStack(int mSize)  {  maxtop = mSize - 1;  st = new char[mSize];  top = -1;  }    SeqStack::~SeqStack()  {  delete[]st;  }    bool SeqStack::Push(char x)  {  if(top == maxtop)  return false;  st[++top] = x;  return true;  }    bool SeqStack::Pop()  {  if(top == -1)  return false;  top--;  return true;  }    char SeqStack::Top()  {  return st[top];  }    void And(); //合取  void Or(); //析取  void Not(); //否定  void If(); //条件  void Iif(); //双条件  bool CanIn(char out); //判断能否进栈  void Suffix(); //求后缀表达式  void Calculate(); //计算公式的值  void Print(); //输出真值表和范式  void Calculate1();    SeqStack stack(200);    map<char,int>map1; //（命题变元，真假）  //dfs枚举t个2^t次方种 01组合  void dfs(map<char,int>::iterator n)//n=begin;  {  if(n==map1.end())  {Calculate1();//计算  return;  }  else  {  n->second=1;  n++;  dfs(n);  n--;  n->second=0;  n++;  dfs(n);  }  }  int main()  {  cout << "! 否定" << endl << "| 析取" << endl <<"& 合取" << endl << "-> 条件" << endl << "<-> 双条件" << endl << endl;  cout<<"请输入命题" <<endl;  char str[100];//初始读入  char ch[100];//简化初始式  cin >> str;  int cnt = 0;  for(int i = 0; i < strlen(str);)  {  if(str[i] == '-')  {  ch[cnt++] = '>';  i += 2;  }  else if(str[i] == '<')  {  ch[cnt++] = '~';  i += 3;  }  else  {  ch[cnt++] = str[i];  if((65<=str[i]&&str[i]<=90)||(97<=str[i]&&str[i]<=122)){map1.insert(pair<char, int>(str[i],1));}  i++;    }  }  ch[cnt++] = '\0';  org = ch;  Suffix(); //转后缀  map<char,int>::iterator iter = map1.begin();    for (iter = map1.begin();iter != map1.end(); iter++){  cout << iter->first<<"\t";//打印各最小命题变元  }  cout<<str; //打印输入式  cout<<endl;    iter = map1.begin();  dfs(iter) ;      if(pdnf.length() != 0)  pdnf.erase(pdnf.length() - 2); //去掉式子末尾的多余连接符  if(pcnf.length() != 0)  pcnf.erase(pcnf.length() - 2);    cout << "主析取范式：" << pdnf << endl << endl;  cout << "主合取范式：" << pcnf << endl << endl;    return 0;  }  void And() //合取  {  res = a \* b;  stack.Push(res);  }    void Or() //析取  {  res = a + b;  res = res > 1 ? 1 : res;  stack.Push(res);  }    void Not() //否定  {  a = stack.Top();  stack.Pop();  res = a == 1 ? 0 : 1;  stack.Push(res);  }    void If() //条件,b->a  {  res = (b == 1 && a == 0) ? 0 : 1;  stack.Push(res);  }    void Iif() //双条件  {  res = (b == a) ? 1 : 0;  stack.Push(res);  }    bool CanIn(char out) //先计算优先级，然后判断能否进栈  {  char in = stack.Top();  int i, o; //分别表示栈内外运算符的优先级  switch(in)  {  case '#':i = 0; break;  case '(':i = 1; break;  case '~':i = 3; break;  case '>':i = 5; break;  case '|':i = 7; break;  case '&':i = 9; break;  case '!':i = 11; break;  case ')':i = 12; break;  }  switch(out)  {  case '#':o = 0; break;  case '(':o = 12; break;  case '~':o = 2; break;  case '>':o = 4; break;  case '|':o = 6; break;  case '&':o = 8; break;  case '!':o = 10; break;  case ')':o = 1; break;  }    if(i < o) //如果栈外的优先级比栈内的高，就可以进栈，因为离栈顶越近，就越先出栈  return true;  else  return false;  }    void Suffix() //转换为后缀表达式  {  string tmp = ""; //保存后缀表达式  stack.Push('#'); //栈底  for(int i = 0; (unsigned)i < org.length(); i++)  {  //if(org[i] == 'P' || org[i] == 'Q' || org[i] == 'R' || org[i] == 'S' || org[i] == 'T' || org[i] == 'U') //如果是P、Q、R 就保存到字符串tmp中  if((65<=org[i]&&org[i]<=90)||(97<=org[i]&&org[i]<=122))  {  tmp = tmp + org[i];  continue;  }  if(CanIn(org[i]))  stack.Push(org[i]);  else if(org[i] == ')')  {  while(stack.Top() != '(')  {  tmp = tmp + stack.Top();  stack.Pop();  }  stack.Pop();  }  else  {  do  {  tmp = tmp + stack.Top();  stack.Pop();  } while(!CanIn(org[i]));  stack.Push(org[i]);  }  }  while(stack.Top() != '#')  {  tmp = tmp + stack.Top();  stack.Pop();  }  stack.Pop(); // '#' 出栈  org = tmp;  }  void Calculate1()  {    for(int i = 0; (unsigned)i < org.length(); i++)//org是后缀表达式  {if((65<=org[i]&&org[i]<=90)||(97<=org[i]&&org[i]<=122))//后缀表达式中遇到命题变元 ，入栈其01值  { stack.Push(map1[org[i]]);  continue;  }    if(org[i] != '!') // 后缀表达式中的单元连接词!  {  a = stack.Top();  stack.Pop();  b = stack.Top();  stack.Pop();  }  switch(org[i])//后缀表达式中遇到双元连接词 ，栈顶变元出栈 ，并计算，更新res真假  {  case '~':Iif(); break;  case '>':If(); break;  case '|':Or(); break;  case '&':And(); break;  case '!':Not(); break;  }  }  map<char,int>::iterator iter = map1.begin();    if(res == 1) //res是整个命题的真假  {  pdnf = pdnf + "(";//pdnf为主析取范式  for (iter = map1.begin();iter != map1.end(); iter++){  string t1(1,iter->first);char t3[3]={'!',iter->first,'\0'}; string t2(t3);//把char类型的iter->first,!iter->first变成string，蛋疼的c++字符串  pdnf = pdnf + (iter->second == 1? t1 : t2) ;  if(iter != --map1.end()){pdnf=pdnf+"&" ;}  }  pdnf = pdnf + ")"+" | ";    }  else  {pcnf = pcnf + "(";//pncf为主合取范式    for (iter = map1.begin();iter != map1.end(); iter++){  string t1(1,iter->first); char t3[3]={'!',iter->first,'\0'}; string t2(t3);//把char类型的iter->first,!iter->first变成string，蛋疼的c++字符串  pcnf = pcnf + (iter->second == 0 ? t1 : t2);  if(iter != --map1.end()){pcnf=pcnf+"|" ;}  }  pcnf = pcnf + ")"+" & ";  }    for (iter = map1.begin();iter != map1.end(); iter++){//打印每种组合命题变元的01值  cout << iter->second << "\t";    }  cout<<res<<endl;  res=0; //非必须  }  **运行截图** | | | | | |

**实验报告**

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **四、实验小结**（包括问题和解决方法、心得体会、意见与建议等）   1. 实验中遇到的主要问题及解决方法   关于命题表达式的处理，参考了网络上的逆波兰算法。即先将中序转后序，再依据各运算符优先级，计算后序表达式的值。  （二）实验心得  该实验的难点在于如何处理输入的命题字符串，需要一些数据结构与算法的知识，可以自行在网络上查找相关资料。完成后此程序理论上可以对任意命题公式求主析取范式与主合取范式并打印真值表，大大提升了相关作业的效率。  （三）意见与建议（没有可省略） | | | | | |
| 1. **支撑毕业要求指标点**   支撑毕业要求的指标点为：   * 1-4掌握计算机科学与技术领域的专业知识，能将专业知识用于分析和解决计算机领域复杂工程问题。   √   * 2-1能够应用数学、自然科学和工程科学的基本知识，识别和分析计算机领域复杂工程问题的特征。 | | | | | |
| **六、指导教师评语 (含学生能力达成度的评价)** | | | | | |
| **成绩** |  | **批阅人** |  | **日期** |  |

如果不太想写太多字，“指导教师评语”也可以设计为如下的各选择项用打勾形式（仅仅作为一个简单示例，请各课程负责人根据课程和实验情况以及支撑的指标点来自行设定选择项，同一门课程的不同实验评分细则项允许存在不同）：

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **评分细则** | **评分项** | **优秀** | **良好** | **中等** | **合格** | **不合格** |
| **遵守实验室规章制度** |  |  |  |  |  |
| **学习态度** |  |  |  |  |  |
| **算法思想准备情况** |  |  |  |  |  |
| **程序设计能力** |  |  |  |  |  |
| **解决问题能力** |  |  |  |  |  |
| **课题功能实现情况** |  |  |  |  |  |
| **算法设计合理性** |  |  |  |  |  |
| **算法效能评价** |  |  |  |  |  |
| **回答问题准确度** |  |  |  |  |  |
| **报告书写认真程度** |  |  |  |  |  |
| **内容详实程度** |  |  |  |  |  |
| **文字表达熟练程度** |  |  |  |  |  |
| **其它评价意见** |  | | | | |
| **本次实验能力达成评价（总成绩）** |  |  |  |  |  |