**实验报告**

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **年级、专业、班级** | | 2019级计算机科学与技术01班 | | **姓名** | 谢双骏 |
| **实验题目** | 词法分析程序的设计与实现 | | | | |
| **实验时间** | 2022.4.2 | | **实验地点** | DS3401 | |
| **实验成绩** |  | | **实验性质** | **□验证性 □设计性 ■综合性** | |
| **教师评价：**  □算法/实验过程正确；□源程序/实验内容提交 □程序结构/实验步骤合理；  □实验结果正确； □报告规范；  实验代码得分（满分7分）：  实验报告得分（满分3分）：  **评价教师签名：** | | | | | |
| **一、实验目的**  🡪了解编译器中词法分析环节的作用和意义  🡪掌握词法分析基础，实现词法分析器 | | | | | |
| **二、实验项目内容**   1. **实验要求：**   请根据给定的文法设计并实现词法分析程序，从源程序中识别出单词，记录其单词类别和单词值，输入输出及处理要求如下：   1. 数据结构和与语法分析程序的接口请自行定义；类别码需按下表格式统一定义； 2. 为了方便进行自动评测，输入的被编译源文件统一命名为testfile.txt（注意不要写错文件名）；输出的结果文件统一命名为output.txt（注意不要写错文件名），结果文件中每行按如下方式组织： 3. 单词类别码 单词的字符/字符串形式(中间仅用一个空格间隔) 4. 单词的类别码请统一按如下形式定义：  |  |  |  |  |  |  |  |  | | --- | --- | --- | --- | --- | --- | --- | --- | | **单词名称** | **类别码** | **单词名称** | **类别码** | **单词名称** | **类别码** | **单词名称** | **类别码** | | 标识符 | IDENFR | else | ELSETK | - | MINU | = | ASSIGN | | 整形常量 | INTCON | switch | SWITCHTK | \* | MULT | ; | SEMICN | | 字符常量 | CHARCON | case | CASETK | / | DIV | , | COMMA | | 字符串 | STRCON | default | DEFAULTTK | < | LSS | ( | LPARENT | | const | CONSTTK | while | WHILETK | <= | LEQ | ) | RPARENT | | int | INTTK | for | FORTK | > | GRE | [ | LBRACK | | char | CHARTK | scanf | SCANFTK | >= | GEQ | ] | RBRACK | | void | VOIDTK | printf | PRINTFTK | == | EQL | { | LBRACE | | main | MAINTK | return | RETURNTK | != | NEQ | } | RBRACE | | if | IFTK | + | PLUS | ： | COLON |  |  |   【输入形式】testfile.txt中的符合文法要求的测试程序。 【输出形式】要求将词法分析结果输出至output.txt中。  【特别提醒】   * 读取的字符串要原样保留着便于输出，特别是数字，这里输出的并不是真正的单词值，其实是读入的字符串，单词值需另行记录。 * 本次作业只考核对正确程序的处理，但需要为今后可能出现的错误情况预留接口。 * 在今后的错误处理作业中，需要输出错误的行号，在词法分析的时候注意记录该信息。 * 单词的类别和单词值以及其他关注的信息，在词法分析阶段获取后，后续的分析阶段会使用，请注意记录；当前要求的输出只是为了便于评测，完成编译器中无需出现这些信息，请设计为方便打开/关闭这些输出的方案。   【文法定义】请到“2020年文法定义(用于查看文法，不计入成绩）”作业中查看文法   1. **实验要求：** 2. 【开发语言及环境】用C/C++实现，平台支持C++11标准，源代码文件必须使用UTF-8编码，才能够输出评测平台能够识别的中文(如果不确定源文件的编码，推荐使用notepad++查看) 3. 提交实验报告及源代码。实验报告需严格遵循学校文档规范，内容包含对应文法、词类编码表、词法分析测试用例。 | | | | | |
| 1. **实验过程或算法** 2. **程序实现内容**   本程序是一个C++语言词法分析程序，可以从上至下，从左至右，依次逐个对源程序的字符进行扫描，按照给定语言的词法规则识别出一个个单词符号，把识别出来的标识符存入符号表打印出来。本程序可以识别要求给出的各种类型的记号，包括给定关键词和运算符，以及标识符等等，本程序也可实现对可能出现的错误格式的处理。  本程序的设计思想主要基于状态转移图，每读进一个字符就检测进入什么状态，根据状态转移图决定状态之间的转移以及输出的时机，最后将各个字符连成串后加入token序列，并标记类别。  本程序完成了实验的要求，包括识别出单词，并记录其单词类别和单词值、跳过空白、检查错误、文件类别输入的读取，以及文件类输出的规范处理。   1. **语言说明** 2. **语言分类**   在本程序中，将语言分成一下几种记号：   1. 标识符：以字母或’ \_ ’开头，后紧跟字母、数字或’ \_ ’组成的符号串 2. 关键词：标识符集合的子集，如int，float，if，else，while，for等在语言中有特殊含义的词 3. 无符号数：是由0-9数字组成的符号串，在实验的语言中可以为0开头 4. 算术运算符：+,-,\*,/ 5. 逻辑运算符： <｜<=｜>｜>=｜!=｜== 6. 分界符： ( ) { } [ ] 7. 标点符号：, : ; 8. 字符常量：'＜算数运算符＞'｜'＜字母＞'｜'＜数字＞’ 9. 字符串："｛十进制编码为32,33,35-126的ASCII字符｝" 10. **状态转换图**   在这里状态0是初始状态，若此时读入的字符是字母或者下划线则进入状态1，即标识符识别过程；若进入的字符是数字则进入数字的识别过程。”<“和”>”和”=”和”!”分别进入状态5,6,7，以下是具体说明：  在开始状态首先读进一个字符。若读入的字符为空则继续读，直到读进非空字符为止，然后根据所读的字符转至相应的处理过程。   1. 在标识符状态，识别字符后继续读取，直到下一个字符不是（数字||字母||下划线）为止，再把字符组合成串，在关键词表中查询标识符是关键词还是用户自定义表示符，并输出相应的类别码，跳到初始状态 2. 在无符号整数状态，识别字符后继续读取，直到下一个字符不是数字为止，再把字符组合成串，并输出整形常量类别码，跳到初始状态 3. 在>,<,=,!状态下，识别字符后继续读取，判断后一个输入是不是等号，如果是，那么把字符组合成串，并输出大于等于类别码；如果不是，那么输入指针回退，跳到初始状态 4. 在算术运算符和分界符以及标点符号状态下，输出对应的类别码，跳到初始状态 5. 在‘状态下，识别字符后继续读取，直到读取到’，将中间读取的字符记录，输出字符常量类别码，跳到初始状态 6. 在“状态下，识别字符后继续读取，直到读取到”，将中间读取的字符组合起来记录，并输出字符常量类别码，跳到初始状态   IMG_1083(20220402-214012)   1. **词法编码表**  |  |  |  |  |  |  | | --- | --- | --- | --- | --- | --- | | 单词名称 | 类别码 | 单词名称 | 类别码 | 单词名称 | 类别码 | | 标识符 | IDENFR | default | DEFAULTTK | >= | GEQ | | 整形常量 | INTCON | while | WHILETK | == | EQL | | 字符常量 | CHARCON | for | FORTK | != | NEQ | | 字符串 | STRCON | scanf | SCANFTK | : | COLON | | const | CONSTTK | printf | PRINTFTK | = | ASSIGN | | int | INTTK | return | RETURNTK | ; | SEMICN | | char | CHARTK | + | PLUS | , | COMMA | | void | VOIDTK | - | MINU | ( | LPARENT | | main | MAINTK | \* | MULT | ) | RPARENT | | if | IFTK | / | DIV | [ | LBRACK | | else | ELSETK | < | LSS | ] | RBRACK | | switch | SWITCHTK | <= | LEQ | { | LBRACE | | case | CASETK | > | GRE | } | RBRACE |   从词法编码表中我们可以看到，最前面是标识符、整形常量、字符常量、字符串，之后是关键字，接下来是运算符和标点符号，最后是分界符。   1. **符号表**  |  |  |  |  | | --- | --- | --- | --- | | **符号序号** | **符号名** | **符号类型** | **引用行** |   我们以测试程序为例，绘制符号表部分：   1. int num1,num2,op,ans; 2. get(num1,num2,op); 3. if(op==0) 4. { 5. ans = num1 + num2; 6. }; 7. if(op==1) 8. { 9. ans = num1 - num2; 10. }; 11. if(op==2) 12. { 13. ans = num1 & num2; 14. }; 15. if(op==3) 16. { 17. ans = num1 | num2; 18. }; 19. put(ans);  |  |  |  |  | | --- | --- | --- | --- | | **符号序号** | **符号名** | **符号类型** | **引用行** | | 1 | Num1 | int | 1,2,5,9,13,17 | | 2 | Num2 | Int | 1,2,5,9,13,17 | | 3 | op | Int | 1,2,3,7,11,15 |  1. **错误处理说明**   常见错误处理主要有：   1. 当第一个读入的字符不属于词法分析程序任何一种情况时，如第一次就读入了￥，@等这样未定义的符号 2. 后续读入的字符不符合状态转换图   在上述情况出现时，则会跳到else程序段，即输出非法字符   1. **程序说明** 2. 在开头我们首先声明一些必要的变量 3. MAXN是最大的序列长度 4. Token是记录的类别码 5. Val是单词的值 6. Row是单词的行 7. Col是单词的列 8. p和cnmd是迭代的index 9. Line和cur在执行的过程中记录行和列   const int MAXN=2000;  //词法分析token序列  string token[MAXN];  string val[MAXN];  int row[MAXN];  int col[MAXN];  int p = 0;  int cnmd =0;  int line=1;  int cur=0;   1. 判断是不是数字   //看是否为数字  bool number(char A){      if(A>='0'&&A<='9')          return true;      return false;  }   1. 判断是不是字母和下划线   //看是否为字母或下划线  bool letter(char A){      if((A=='\_')||(A>='A'&&A<='Z')||(A>='a'&&A<='z'))          return true;      return false;  }   1. 看是否是保留字   如果识别的字符串是保留字，那么就记录下来，返回true，（由于篇幅原因只展示const，其他保留字代码相似）；否则，返回false；  //看是否是保留字  bool reserved\_word(string s){      if(s=="const")      {          token[p++] = "CONSTTK";          val[cnmd++] = s;          row[p-1]=line;          col[p-1]=cur;          return true;      }      ...      return false;  }   1. 词法分析的主程序，输入一个文件指针，我们使用fgetc函数每次读进一个字符，并使用fseek在特定的状态下回退：   void lexical\_analysis(){}  //标识符、整数、字符、字符串、保留字、运算符、分界符      FILE \*f;      char CHAR;      f=fopen("testfile.txt","r");   1. 如果读取到了空格和制表符跳过   //读取文件并处理  while((CHAR=fgetc(f))!=EOF){          cur++;          //除了空格和Tab全读取          while(CHAR!=' '&&int(CHAR)!=9){   1. 读取到了换行符就line+1，cur清0   else if(CHAR=='\n'){                  line++;                  cur=0;              }   1. 非法字符进入错误处理   else{                  //错误处理接口                  cout<<"\""<<CHAR<<"\""<<" in line "<<line<<" , column "<<cur<<endl;                  throw "illegal input";              }   1. 其他字符按照分析程序进行分析即可 2. 如果第一个读取的是字母或者下划线，那么一直读取字符直到下一个字符不是字母/下划线/数字为止，然后调用reserved\_word函数判断是不是保留字，是则token序列加入对应类别码和单词值，不是则token序列加入标识符类别码和单词值，记录行列数，行列标记更新，因为读取到了标识符后的字符，所以需要fseek函数回退   //以字母或开头,要么是保留字，要么是标识符              if(letter(CHAR)){                  string s="";                  int k=0;                  do{                      s+=CHAR;                  }                  while((CHAR=fgetc(f))!=EOF&&CHAR!=' '&&(letter(CHAR)||number(CHAR)));                  fseek(f,-1,SEEK\_CUR);                  if(!(reserved\_word(s)))//判断标识符是保留字还是标识符                  {                      token[p++] = "IDENFR";                      val[cnmd++] = s;                      row[p-1]=line;                      col[p-1]=cur;                  }                  cur+=s.length();              }   1. 第一个读取的字符是数字，那么一直读取字符直到下一个字符不是数字为止，然后token序列加入整数类别码和单词值，记录行列数，行列标记更新，因为读取到了整数后的字符，所以需要fseek函数回退   //整数              else if(number(CHAR)){                  string s="";                  int i=0;                  do{                      s+=CHAR;                    }                  while((CHAR=fgetc(f))!=EOF&&CHAR!=' '&&number(CHAR));                  fseek(f,-1,SEEK\_CUR);                  token[p++] = "INTCON";                  val[cnmd++] = s;                  row[p-1]=line;                  col[p-1]=cur;                  cur+=s.length();              }   1. 判断是否为运算符，是的话直接token序列加入对应类别码和单词值，记录行列数，行列标记更新   //运算符              else if(CHAR=='+'){                  token[p++] = "PLUS";                  val[cnmd++] += CHAR;                  row[p-1]=line;                  col[p-1]=cur;              }              else if(CHAR=='-'){                  token[p++] = "MINU";                  val[cnmd++] += CHAR;                  row[p-1]=line;                  col[p-1]=cur;              }              else if(CHAR=='\*'){                  token[p++] = "MULT";                  val[cnmd++] += CHAR;                  row[p-1]=line;                  col[p-1]=cur;              }              //不用处理注释              else if(CHAR=='/'){                  token[p++] = "DIV";                  val[cnmd++] += CHAR;                  row[p-1]=line;                  col[p-1]=cur;              }   1. 读取到关系运算符，要注意看后边是否有等号，并以此决定是单个字符加入token序列还是带上后边的等号加入token序列，然后加入单词值，记录行列数，行列标记更新（如果后续的字符不是等号的话，需要回退）   else if(CHAR=='='){                  //后一个符号也是等于                  if((CHAR=fgetc(f))!=EOF&&CHAR=='=')                  {                      token[p++] = "EQL";                      val[cnmd++] = "==";                      row[p-1]=line;                      col[p-1]=cur;                      cur+=1;                  }                  //直接标记等号，然后回退                  else{                      token[p++] = "ASSIGN";                      val[cnmd++] = "=";                      row[p-1]=line;                      col[p-1]=cur;                      fseek(f,-1,SEEK\_CUR);                      break;                  }              }              else if(CHAR=='<'){                  //小于等于                  if((CHAR=fgetc(f))!=EOF&&CHAR=='='){                      token[p++] = "LEQ";                      val[cnmd++] = "<=";                      row[p-1]=line;                      col[p-1]=cur;                      cur+=1;                  }                  //小于，回退，因为向后读取了一个字符                  else {                      token[p++] = "LSS";                      val[cnmd++] = "<";                      row[p-1]=line;                      col[p-1]=cur;                      fseek(f,-1,SEEK\_CUR);                  }              }              else if(CHAR=='>'){                  //大于等于                  if((CHAR=fgetc(f))!=EOF&&CHAR=='='){                      token[p++] = "GEQ";                      val[cnmd++] = ">=";                      row[p-1]=line;                      col[p-1]=cur;                      cur+=1;                  }                  //大于，回退，因为向后读取了一个字符                  else {                      token[p++] = "GRE";                      val[cnmd++] = ">";                      row[p-1]=line;                      col[p-1]=cur;                      fseek(f,-1,SEEK\_CUR);                  }              }              else if(CHAR=='!'){                  //不等于                  if((CHAR=fgetc(f))!=EOF&&CHAR=='='){                      token[p++] = "NEQ";                      val[cnmd++] = "!=";                      row[p-1]=line;                      col[p-1]=cur;                      cur+=1;                  }                  //感叹号后只可能是等于              }   1. 如果是分界符或者标点符号的话，直接token序列加入对应类别码和单词值，记录行列数，行列标记更新，此处就拿一个逗号和左括号作为示例   else if(CHAR==','){                  token[p++] = "COMMA";                  val[cnmd++] += CHAR;                  row[p-1]=line;                  col[p-1]=cur;              }  else if(CHAR=='('){                  token[p++] = "LPARENT";                  val[cnmd++] += CHAR;                  row[p-1]=line;                  col[p-1]=cur;              }   1. 如果读取到了单影号的话，直接读取后续的字符直到读取到另一个单影号，中间的字符作为字符常量加入token序列，然后记录其单词值，记录行列数，行列标记更新   //如果是单影号，字符              else if(CHAR=='\''){                  string s = "";                  while((CHAR=fgetc(f))!=EOF&&CHAR!='\'')                  {                      s+=CHAR;                    }                  token[p++] = "CHARCON";                  val[cnmd++] = s;                  row[p-1]=line;                  col[p-1]=cur+1;                  cur+=s.length()+1;              }   1. 如果读取到了双影号的话，直接读取后续的字符直到读取到另一个双影号，中间的字符作为字符串，token序列加入字符串类别码，然后记录其单词值，行列数，行列标记更新   //如果是双赢号，字符串              else if(CHAR=='"'){                  string s = "";                  while((CHAR=fgetc(f))!=EOF&&CHAR!='"')                  {                      s+=CHAR;                  }                  token[p++] = "STRCON";                  val[cnmd++] = s;                  row[p-1]=line;                  col[p-1]=cur+1;                  cur+=s.length()+1;              }   1. 直到读取到EOF标识，即文件结束 | | | | | |
| 1. **实验总结** 2. **问题** 3. **识别双等号** 4. 详述：在识别双等号或者>=等情况的时候，因为要求，必须要读取后一个字符，所以如果后一个字符不是等号的话，在进行循环的时候，因为fget函数，就会跳过这个字符的识别，比如在分析i>j的时候，因为在分析>的时候已经读取了j了，所以j的分析就会被跳过。 5. 解决方法：在识别的时候，如果后续不是等号，就回退，使用fseek函数 6. **测试程序出现的一些特殊情况** 7. 详述：比如在测试文件中，PRINTF和printF也可以被识别成PRINTFTK类别码，因此在平台上无法通过。 8. 解决方法：把这几个特殊情况下的字符串也识别为保留字即可。 9. **错误识别** 10. 详述：最开始的时候，因为未考虑到错误识别，程序的设计实现并未严格按照状态机实现，因此在错误处理的时候，会出现一些跳转的问题。 11. 解决方法：更改程序，严格按照状态机实现，问题即可解决。 12. **实验收获**   这次实验难度不太大，主要是对于C++文件和字符的读取相关的函数我不太熟悉，然后对于一些字符读进来之后一些逻辑判断有点不清楚，但通过自己的检查与在网上的学习，逐步解决了很多问题。通过这次实验，我对于编译原理中词法分析的过程理解的更为透彻了。本质上词法分析就是一个状态机的设计，只要你设计出来了状态机，基本上不会有太大的问题，更有甚者，你可以直接把他看成一个字符串的处理程序，用普通的程序设计思想去做即可。 | | | | | |