实验报告

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **年级、专业、班级** | | **2018级物联网02班** | | **姓名** | **20184261 刘玉莽** |
| **实验题目** | 词法分析程序的设计与实现 | | | | |
| **实验时间** | **2021/4/17** | | **实验地点** | **DS1421** | |
| **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； □语法、语义正确； □报告规范；  其他：  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：   设计并实现C语言的词法分析程序，要求实现以下功能：   1. 最低要求： 2. 可以识别出用C语言编写的源程序中的每个单词符号，并以记号的形式输出每个单词符号。（基本数据类型至少包含整数、字符、布尔类型；至少包含赋值语句、变量申明语句、算数表达式语句、布尔表达式语句、循环语句、分支语句、函数定义及调用） 3. 可以识别并且跳过源程序中的注释。 4. 可以统计源程序中的语句行数、各类单词的个数，以及字符总数，并输出统计结果。 5. 检查源程序中存在的词法错误，并报告错误所在的位置。 6. 可选要求： 7. 能识别指数形式的浮点数。 8. 能识别8进制/16进制数。 9. 能识别数组类型及结构体。 10. 对源程序中出现的错误进行适当的恢复，使得词法分析可以继续进行，对源程序进行一次扫描，即可检查并报告源程序中存在的所有词法错误及错误位置。 11. 实验要求： 12. 自由选用程序设计语言（java，python，c/c++）作为实现语言，手工编写词法分析程序。 13. 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. C语言子集   数据类型: **int**, 无符号整数, 取值范围0-9999  int a;  int a,b;  int a = 1;  算术运算符: **+**,**-**  a = b + 1;  a = b + c;  赋值运算符: **=**  a = 1;  关系运算符: **==** ,**>**,**<,<>,>=,<=**  a = (b==c);  a = (b>c);  a = (b<c);  逻辑运算符: **&&**, **||, !**  a = (b&&c);  a = (b||c);  a = (!b);  条件语句: **if**  if(a==b)  {  };  循环语句: **while**  while(a==b)  {  };  输入,输出: **get**,**put**  get(a);  put(a);  语句结束符: **;**  **加分项：**  条件语句 **if else**  if(a==b)  {  };  else  {  }  及其他C语言部分，例如数组等; | | | | | |
| 三、实验过程或算法 | | | | | |
| 1. **实现内容**   最低要求全部实现，包括   1. 可以识别出用C语言编写的源程序中的每个单词符号，并以记号的形式输出每个单词符号。（基本数据类型至少包含整数、字符、布尔类型；至少包含赋值语句、变量申明语句、算数表达式语句、布尔表达式语句、循环语句、分支语句、函数定义及调用） 2. 可以识别并且跳过源程序中的注释。 3. 可以统计源程序中的语句行数、各类单词的个数，以及字符总数，并输出统计结果。 4. 检查源程序中存在的词法错误，并报告错误所在的位置。   可选要求均实现：   1. 能识别指数形式的浮点数。 2. 能识别8进制/16进制数。 3. 能识别数组类型及结构体。 4. 对源程序中出现的错误进行适当的恢复，使得词法分析可以继续进行，对源程序进行一次扫描，即可检查并报告源程序中存在的所有词法错误及错误位置。   **分析过程**     1. **语言说明：设计的词法分析器包含C语言哪些子集，相应的文法描述**   词法分析器对应的c语言是一个包括宏定义，注释，函数调用，基本数据类型（char、void、int、float），基本运算（算数运算、赋值运算，关系运算、逻辑运算、位运算），for，while循环的子集。  对应文法  <程序> -> <预处理>|<函数>  <函数> -> <类型说明><函数名称><复合语句>  <类型说明> -> void|char|int|float  <复合语句> -> <”{}”>|<”{语句列表}”>  <语句列表> -> <语句>|<语句列表><语句>  <语句> -> < 表达式语句> |< 条件语句> |< 循环语句> |< 跳转语句>  表达式语句 -> <表达式>”;“  <表达式> -> <赋值表达式>  |<  表达式> ‘,‘< 赋值表达式>  <条件语句> -> <IF(表达式)语句> |< IF‘(‘ 表达式 )语句 >< 条件语句>  <循环语句>→ <WHILE ( 表达式) 语句> |< FOR (表达式语句+表达式语句 )语句|FOR ( 表达式语句 表达式语句 表达式)语句>   1. 词法编码表说明  |  |  | | --- | --- | | token值 | token属性 | | "int", "main", "char", "bool", "void",    "if", "else", "while", "return", "for", | 关键字 | | 变量声明 | 标识符 | | 十进制数 | 十进制数 | | 0x4f | 十六进制数 | | 0432 | 八进制数 | | ”str“ | 字符串 | | ‘c’ | 字符 | | ++ | 自增运算符 | | + | 加号 | | -- | 自减运算符 | | - | 减号 | | \* | 乘号 | | / | 除法 | | // | 注释 | | % | 求模 | | == | 相等运算符 | | = | 赋值运算符 | | != | 不相等运算符 | | ! | 逻辑非 | | >= | 大于等于运算符 | | >> | 右移 | | > | 大于运算符 | | <= | 小于等于运算符 | | << | 左移 | | < | 小于等于运算符 | | && | 逻辑与 | | & | 按位与 | | || | 逻辑或 | | | | 按位或 | | ^ | 异或 | | ~ | 取反 | | ';' ,'{' ,'}', '(' ,')'  ,',' ,':' | 符号 | | struct | 结构体 | | [xx] | 数组 |  1. 错误处理   如果在词法分析阶段发现了词法错误，程序采用的方式是报告出错位置，并且忽视这个错误，继续向下分析。   1. 程序说明：   lexer(char \*src)函数，是词法分析的主要函数，每次读入源程序的一行代码进  分析。  while (!(feof(pFile)))              {                  memset(src, 0, sizeof(src));                  fgets(src, 499, pFile);                  lexer(src);  // every time analyze one line              }              printf("总行数:%d\t总的字符数:%d\t标识符与关键字总个数:%d\n", lines, characters, words);  对于标识符以及结构体的识别分析  else if(isalpha(token) || token == '\_'){              undifined = 0;              // parse identifier              words++;              int flag\_key;              flag\_key = 0;  // keyword in default              while (isalnum(\*src) || \*src == '\_')              {                  src++;              }              for(int i = 0; i < KEYWORDNUM; i++){                  if(memcmp(lastPos, (int)(keywords)+10\*i, src-lastPos) == 0 && (strlen(keywords[i]) == src-lastPos)){  // keywords+10\*i is wrong                      if(i == 0){                          printf("第%d行，定义了结构体", lines);                      }                      else{                          printf("%s\t <---> 关键字", keywords[i]);                      }                      flag\_key = 1;                      break;                  }              }              if(!flag\_key){                  memcpy(str, lastPos, src-lastPos);                  printf("%s\t <---> 标识符", str);              }          }  对于十进制，八进制，十六进制以及指数形式的浮点数的识别。  else if(isdigit(token)){  // consider positive number              undifined = 0;              // parse number, include dec, oct(045), hex(0x45)              int float\_num = 0;              int val = 0, dec\_oct;              if(token != '0'){  // dec                  val = token - '0';                    if(\*src == '.'){                      // here, we are able to recognize floating point numbers in exponential form                      // and the form must be standard.                      double frac = 0.1, val\_float;                      double base, exp;                        float\_num = 1;                      base = token-'0';                      src++;                      while (\*src != 'e' && \*src != 'E')                      {                          base += (\*src - '0') \* frac;                          frac /= 10;                          src++;                      }                      src++;                      if(\*src == '-'){                          src++;                          exp = -(\*src++ - '0');                          while (isdigit(\*src))                          {                              exp = exp \* 10 - (\*src - '0');                              src++;                          }                      }                      else{                          exp = \*src++ - '0';                          while (isdigit(\*src))                          {                              exp = exp \* 10 + (\*src - '0');                              src++;                          }                      }                      val\_float = base \* pow(10, exp);                      printf("第%d行，浮点数，值是%lf", lines, val\_float);                  }                  else{  // decimal integer                      while (isdigit(\*src))                      {                          val = val \* 10 + \*src - '0';                          src++;                      }                      memcpy(str, lastPos, src-lastPos);                      printf("%s\t <---> 十进制数，值为%d", str, val);                  }              }              else if(\*src == 'x'){  // hex                  // 'A'\t <---> 0x41                  // 'a'\t <---> 0x61                  val = (\*(++src) & 15) + (\*src >= 'A' ? 9 : 0);                  src++;                  while (isdigit(\*src) || ((\*src)>='A' && (\*src)<='F') || ((\*src)>='a' && (\*src)<='f'))                  {                      val = val\*16 + (\*(src) & 15) + (\*src >= 'A' ? 9 : 0);                      src++;                  }                  memcpy(str, lastPos, src-lastPos);                  printf("%s\t <---> 十六进制数，值为%d", str, val);              }              else{  // oct                  val = token - '0';                  while (isdigit(\*src))                  {                      val = val \* 8 + \*src - '0';                      src++;                  }                  memcpy(str, lastPos, src-lastPos);                  printf("%s\t <---> 八进制数，值为%d", str, val);              }          }  对于数组的识别  // recognize array          int cap = 0;  // capacity          switch(token){          case('['):              cap = \*src++ - '0';              while (isdigit(\*src))              {                  cap = cap \* 10 + \*src - '0';                  src++;              }              if(\*src == ']'){                  undifined = 0;                  src++;                  printf("第%d行，定义了数组，容量为%d", lines, cap);              }          default:              break;          }  对于词法错误的处理  if(undifined){              memcpy(str, lastPos, src-lastPos);              printf("第%d行，存在词法错误,%s未定义", lines, str);          } | | | | | |
| 1. 实验测试   测试程序“hello.c”  #include <stdio.h>  #include <string.h>  struct stu{      char name[20];      int id;  };  int main(){      // find the minimum val      int a, b, c;      int arr[3];      double float\_num = 1.0e3;      @  // undefined      a = 03;      b = 0xf5;      if(a == b){          c = a;      }      else if(a < b){          c = a;      }      else{          c = b;      }      while (a)      {          a--;      }      return 0;  }  测试结果        对源程序的各类单词进行了分析，输出了分析结果 ，并且在程序的最后给出类被分析程序的总行数，总的字符数以及标识符与关键字的总个数。 | | | | | |
| 五、实验总结  实验过程中遇到的主要问题是当单词是数字的时候，因为包括的情况比较多，十进制，八进制，十六进制，十六进制以及指数形式的浮点数。为了简化分析，这个词法分析程序主要考虑非负数。对于十六进制数，为了简化操作，采用了下面的方式来得到他的值。    while (isdigit(\*src) || ((\*src)>='A' && (\*src)<='F') || ((\*src)>='a' && (\*src)<='f'))                  {                      val = val\*16 + (\*(src) & 15) + (\*src >= 'A' ? 9 : 0);                      src++;                  }  因为’A’的十六进制是0x41，’a’的十六进制是0x61，所以通过上述方式得到个位数之后简化操作。 | | | | | |