实验报告

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **年级、专业、班级** | | **2019级计算机科学与技术卓越01班** | | **姓名** | **魏永森** |
| **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** | **DS3 402** | |
| **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：  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)  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】（1）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。                      （2）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。                      （3）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。                      （4）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：   1、【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)  2、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。 | | | | | |
| 三、实验过程或算法  程序由main.cpp和table.h组成，table.h中包括头文件的导入和全局变量的声明。main.cpp则是词法分析器的逻辑主体。  首先定义全局变量，table是词法编码表，使用map实现；signal\_char是一些可以单独识别的特殊字符集合；signal\_comparison是单个的比较运算符集合；double\_comparison是双目比较运算符集合;key\_word是关键字集合。在table.h文件中进行定义。    之后定义main.cpp中的一些局部变量。buffer是存储全部输入的字符串；forward是向前指针；c是当前正在读的字符，也就是buffer[forward]；token是读入的字符串，默认等于c；name和category是两个全局变量，用来记录识别出的单词名称和类别码。  程序主体是利用while循环和if-else结构实现，逐个读取buffer中的字符c，并添加到token中，每次循环都将使得forward+1，token重置。框架如下：  while(forward < buffer\_size){         token += buffer[forward];  //更新token为当前字符c  // 1.判断特殊字符   +=\*/:=;,()[]{}  // 2.判断比较运算符    先判断 <,>,=,! 再判断 <=,>=,==,!=  // 3.判断整形常量  // 4.判断字符常量  // 5.判断字符串  // 6.判断关键字 或 标识符         token = ""; ++forward; // 重置token, forward指针前进一位  }  **循环体内识别步骤**  1.特殊字符的识别  // 特殊字符  +=\*/:=;,()[]{}  if(signal\_char.find(token)!=signal\_char.npos){      name.emplace\_back(token);      category.emplace\_back(table[token]);  }  目前token中只含有c，直接在signal\_char = “+=\*/:=;,()[]{}” 中寻找是否有含有token即可。如果能找到，说明当前字符就是特殊字符，就可以向要输出的两个数组中添加内容，单词名称name直接添加token，词类编码category添加根据token在词类编码表table中索引到的table[token]。  2.比较运算符的识别  // 比较运算符    先判断 <,>,=,! 再判断 <=,>=,==,!=  if(signal\_comparison.find(token)!=signal\_comparison.npos){      token += buffer[++forward];      if(find(double\_cocmparison.begin(),double\_cocmparison.end(),token) != double\_cocmparison.end()){          name.emplace\_back(token);          category.emplace\_back(table[token]);      }else{          token.erase(1,1);  // 删除第二个字符          name.emplace\_back(token);          category.emplace\_back(table[token]);          --forward; // 回退一位      }  }  首先，token中只含有c一个字符，直接在signal\_comparison集合中寻找目前token是不是一个单目比较运算符。如果是的话，就向token中继续添加一个字符，在判断目前token是不是在双目运算符集合double\_comparison中。  之后进行分支选择，如果现在token是一个双目运算符，那么直接向name，category中添加token和table[token]即可，无需对forward进行操作；但是如果token不是一个双目运算符，那么就需要将token中的第二位字符删除，因为第一个字符可以确保是运算符，可以在table中进行索引，再想name和category中添加token，table[token]，这时最后forward要回退一位，因为第二位既然不是包含在运算符中，那么就需要外层while再对其进行判断是否符合其他情况。  3. 判断是否是整形常量  // 判断整形常量  if(isdigit(buffer[forward])){      while(isdigit(buffer[++forward])){          token += buffer[forward];      }      --forward;      name.emplace\_back(token);      category.emplace\_back(table["整形常量"]);  }  判断当前字符c是否是一个数字，如果是的话进入下一个状态，持续向token中添加字符直至下一个字符不是数字。最后forward指针回退一位，向name和category中添加结果。  4. 判断是否是字符常量  // 判断字符常量  if(token=="\'"){      // cout << "开始字符判断..." ;      while(buffer[++forward] !='\''){          token+=buffer[forward];      }      token.erase(0,1);  // 去掉开头的  '      // cout << "当前token: " << token << "类型: " << table["字符常量"] <<  endl;      name.emplace\_back(token);      category.emplace\_back(table["字符常量"]);  }  如果遇到了字符c == **‘** , 那么就是一个字符常量的起始点，接着向token中不断添加字符直至遇到下一个单引号字符 **’** 。因为当前已经确定最后一位字符是单引号，所以不需要对forward进行回退一位的处理。  5. 字符串的判断  // 判断字符串  if(token=="\""){      // cout << "开始字符串判断...." ;      while(buffer[++forward] !='\"'){          token+=buffer[forward];      }      token.erase(0,1);  // 去掉开头的  "      // cout << "当前token: " << token << ".... "<< table["字符串"];      name.emplace\_back(token);      category.emplace\_back(table["字符串"]);      // cout << "已添加进容器" << endl;  }  字符串的判断和字符常量类似，只不过是以判断双引号为起始点和结尾。  6. 关键字和标识符的识别  // 判断关键字 或 标识符  if(isalpha(buffer[forward]) || buffer[forward]=='\_'){  // 可以是字母，数字或者下划线\_开头      while(isalnum(buffer[++forward]) || buffer[forward]=='\_'){          token += buffer[forward];      }      --forward;      string temp = token;    // 保留token原有格式，向name中填写      transform(token.begin(),token.end(),token.begin(),::tolower);   // token变为小写,关键字识别用小写      if(find(key\_word.begin(),key\_word.end(),token) != key\_word.end()){          name.emplace\_back(temp);          category.emplace\_back(table[token]);      }else{          name.emplace\_back(temp);          category.emplace\_back(table["标识符"]);      }  }  判断方法是全部当成用户自定义的标识符处理。只要遇到一个字母，数字或下划线\_就开始向token中不断地添加字符，直至遇到一个非字母，数字或下划线\_。接着使用一个暂时使用的字符串temp来记录当前的token，因为之后的tolower处理会使得token被改变。本次实验的关键字不区分大小写，比如coNst也是关键字，所以需要对token进行transform(::tolower)处理。  接着在关键字集合key\_word中寻找当前的token是否在其中，如果在，那么就是一个关键字，向name中添加未进行tolower处理的temp，但是category中需要根据经过tolower处理的token来索引词类编码table[token]；如果token不在key\_word中，那么说明token是关键字，向name和category中分别添加未经tolower处理的temp和table[“标识符”]。  7. 错误处理  如果遇到了当前字符c是在1~6种情况之外的字符，那么就不对其进行处理， forward前进一位，token重置，开始下一位字符的判断。 | | | | | |
| 四、实验测试  给出正常测试字符串 <====>=>>=+=123+\_-51\*2'123'[[[)))"asdf"  问号是结尾判断符。可以观察到能按字符输出正确的词法分析结果。    进行样例代码的测试，对比答案之后无误。    修改文件读取和将输出添加到output文件后，打包main.cpp和table.h并提交OJ平台进行测试，可以通过全部测试。 | | | | | |
| 五、实验总结  **遇到的问题：**  1. 在进行字符常量判断时，经常会程序崩溃。  原因是"\'"不能写成" \' "，空格也会加入判断，并且最后识别字符和字符串的时候，最后不forward回退一位，一位已经知道了结束字符，不用对其再一次进行判断。  2. 刚开始所有测试点都无法通过，输出是空。  问题出现在输入输出和编码。默认的cpp程序时GBK编码，没有注意需要改成utf-8的格式。平台默认读取testfile.txt文件并输出到output.txt文件中进行测试，更改后可以通过大部分测试。  3. 关键字不分大小写，标识符可以\_开头  没注意文法要求中的细节，在测试点出错之后进行修改。对token进行了tolower处理之后再去key\_word中寻找，修改完成后可以通过所有测试点。  **总结：**  这次实验使我加强了对词法分析的了解，掌握了状态机的使用。但是代码上其实还有很多细节未处理，比如换行，行数的记录。目前代码的复杂性还不高，使用了if-else结构，未使用状态机标配的switch-case结构，可能还需要根据之后的实验进行修改。 | | | | | |