实验报告

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **年级、专业、班级** | | **2019级计算机科学与技术卓越02班** | | | **姓名** | **李燕琴** |
| **实验题目** | 词法分析程序的设计与实现 | | | | | |
| **实验时间** | **2022-04-01** | | **实验地点** | **DS3402** | | |
| **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 | | | | | | |
| 1、实现内容 本实验主要根据给定文法设计并实现了简易的词法分析器，从源程序中识别出单词，记录其单词类别和单词值；通过读入testfile.txt文件并将结果输出到output.txt文件，并通过希冀平台踢狗的判题器，进行验证。 2、语言说明 实验中涉及的编程语言为SysY 语言，是 C 语言的一个子集。每个 SysY 程序的源码存储在一个扩展名为 sy 的文件中。该文件中有且仅有⼀个名为 main 的主函数定义，还可以包含若干全局变量声明、常量声明和其他函数定义。SysY 语言支持 int 类型和元素为 int 类型且按行优先存储的多维数组类型，其中 int 型整数为 32 位有符号数；const 修饰符用于声明常量。 3、词类编码 根据题目要求，词类编码表如表 1所示。  表 1 词类编码表   |  |  |  |  |  |  |  |  | | --- | --- | --- | --- | --- | --- | --- | --- | | 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 |  |  |  4、程序说明①getsym程序 程序的主要内容，FSM的主体，用于判断下一个可以规约的串，具体结构如图 1所示：    图 1 getsym规约主函数 ②judge程序 根据getsym得到的规约原始字符串srcStr，输出具体的类别码，具体结构如图 2所示。其中caseCode是为了方便进入特定的判断程序而自定义的code，主要含义如表 1所示。    图 2 judge输出结果判断  表 2 caseCode编码   |  |  | | --- | --- | | caseCode | 含义 | | specialStrCode | 词汇编码表中特定的单词或标识符 | | commonStrCode | 普通字符串 | | numberCode | 整形常量 | | charCode | 字符常量 |  ③标识符或关键字的识别 标识符或关键字以字母或下划线为首字母，其他部分可用字母、数字、下滑线。在处理时，首先判断第一个字母是否是字母或下滑线，之后再依次读取接下来的字符，直到该字符不是字母、数字或下划线，主要流程如图 3所示，源码实现部分如图 4所示。    图 3 标识符或关键字识别过程    图 4 标识符或关键字代码部分  处理specialStrCode时，主要通过如图 5所示的类别码映射表区分关键字和标识符，即在该映射表中的为关键字，否则为标识符。    图 5 特殊单词及其类别码映射 ④数字识别部分 数字识别，首字母必须为0…9的数字，且其余部分也需为数字。主要识别过程和代码如图 6、图 7所示。    图 6 数字识别过程    图 7 数字识别源代码 ⑤运算符识别 赋值符=，基本运算符主要由+、-、\*、/组成，由于是单字符运算符，故判断逻辑比较简单。比较运算符主要由>、>=、<、<=、!=、==六类，其中有单字符也有双字符，故需要多一步判断。单、双字符判断过程代码分别如图 8、图 9所示。运算符识别过程如图 10所示。    图 8 单字符运算符识别源代码    图 9 双字符运算符识别源代码    图 10 运算符的识别过程  其中标点符号，如’:’、’,’、’;’ 识别流程和单字符运算符一致。 ⑥字符串的识别 通过双引号引出字符串，匹配最近的另一个双引号，但这里需要检查双引号是否匹配，注意字符串中如果上一个字符为转移字符，下一个字符为双引号，则不匹配该双引号，计入字符串，再读取下一个字符。具FSM如图 11所示，代码如图 12所示。    图 11 字符串的识别    图 12 字符串识别源码 ⑦单引号匹配过程 单引号和双引号不同的是，单引号内部只需要一个字符，注意如果是转移字符’\\’，则需要单独再读取一个字符并计入srcStr中。具体识别流程如图 11图 12所示。    图 13 字符识别过程    图 14 字符识别源代码 ⑧括号匹配识别 本实验涉及的文法中，包括小括号、中括号、大括号等，需要两两匹配，故本实验我主要通过递归来构建匹配过程。匹配到小括号时，再次进入getsym()，直到遇到反向括号‘)‘，或程序终止时（遇到’\0’），返回getsym()函数，且如果是前者，正常输出类别码；否则，出现括号不匹配错误。主要过程如图 15图 16所示。    图 15 括号识别过程（以小括号为例）    图 16 括号匹配源代码 ⑨错误处理 在词法分析实验中主要处理如图 18所示的错误。其中，错误来源在之前的识别过程中提及，这里不做赘述。    图 17 本实验涉及的主要错误类型  在错误处理上，需要输出错误所在行数，以及错误类型。故在读取字符的时候，就需要根据读取的内容实时记录行数，代码如图 19所示。    图 18 错误处理相关程序 | | | | | | |
| 四、实验测试 1、要素测试 这里提供的测试程序，直接照用题目的测试程序，可能涵盖的要素不齐全，但因为已经过了希冀OJ的测试，故该程序是满足上文提到的所有要素的。 ①测试样例  1. coNst **int** cONst1 = 001, const2 = -100; 2. **const** **char** const3 = '\_'; 3. **int** change1; 4. **char** change3; 5. **int** gets1(**int** var1,**int** var2){ 6. change1 = var1 + var2; 7. **return** (change1); 8. } 9. **void** main(){ 10. printf("Hello World"); 11. printf(gets1(10, 20)); 12. }  ②文件输出  1. CONSTTK coNst 2. INTTK int 3. IDENFR cONst1 4. ASSIGN = 5. INTCON 001 6. COMMA , 7. IDENFR const2 8. ASSIGN = 9. MINU - 10. INTCON 100 11. SEMICN ; 12. CONSTTK const 13. CHARTK char 14. IDENFR const3 15. ASSIGN = 16. CHARCON \_ 17. SEMICN ; 18. INTTK int 19. IDENFR change1 20. SEMICN ; 21. CHARTK char 22. IDENFR change3 23. SEMICN ; 24. INTTK int 25. IDENFR gets1 26. LPARENT ( 27. INTTK int 28. IDENFR var1 29. COMMA , 30. INTTK int 31. IDENFR var2 32. RPARENT ) 33. LBRACE { 34. IDENFR change1 35. ASSIGN = 36. IDENFR var1 37. PLUS + 38. IDENFR var2 39. SEMICN ; 40. RETURNTK return 41. LPARENT ( 42. IDENFR change1 43. RPARENT ) 44. SEMICN ; 45. RBRACE } 46. VOIDTK void 47. MAINTK main 48. LPARENT ( 49. RPARENT ) 50. LBRACE { 51. PRINTFTK printf 52. LPARENT ( 53. STRCON Hello World 54. RPARENT ) 55. SEMICN ; 56. PRINTFTK printf 57. LPARENT ( 58. IDENFR gets1 59. LPARENT ( 60. INTCON 10 61. COMMA , 62. INTCON 20 63. RPARENT ) 64. RPARENT ) 65. SEMICN ; 66. RBRACE }  2、错误测试 因实验一涉及的错误过于简单，如下的测试样例只覆盖了部分错误，在后续的实验会一一完善程序。 ①测试样例  1. coNst **int** cONst1 = 001, const2 = -100; 2. **const** **char** const3 = '\_; 3. **int** change1; 4. **char** change3; 5. **int** gets1(**int** var1,**int** var2){ 6. change1 = var1 + var2; 7. **return** (change1; 8. } 9. **void** main(){ 10. printf("Hello World); 11. printf(gets1(10, 20)); 12. }  ②错误输出  ③文件输出  1. CONSTTK coNst 2. INTTK int 3. IDENFR cONst1 4. ASSIGN = 5. INTCON 001 6. COMMA , 7. IDENFR const2 8. ASSIGN = 9. MINU - 10. INTCON 100 11. SEMICN ; 12. CONSTTK const 13. CHARTK char 14. IDENFR const3 15. ASSIGN = 16. SEMICN ; 17. INTTK int 18. IDENFR change1 19. SEMICN ; 20. CHARTK char 21. IDENFR change3 22. SEMICN ; 23. INTTK int 24. IDENFR gets1 25. LPARENT ( 26. INTTK int 27. IDENFR var1 28. COMMA , 29. INTTK int 30. IDENFR var2 31. RPARENT ) 32. LBRACE { 33. IDENFR change1 34. ASSIGN = 35. IDENFR var1 36. PLUS + 37. IDENFR var2 38. SEMICN ; 39. RETURNTK return 40. LPARENT ( 41. IDENFR change1 42. SEMICN ; 43. RBRACE } 44. VOIDTK void 45. MAINTK main 46. LPARENT ( 47. RPARENT ) 48. LBRACE { 49. PRINTFTK printf 50. LPARENT ( 51. STRCON Hello World); 53. PRINTFTK printf 54. LPARENT ( 55. IDENFR gets1 56. LPARENT ( 57. INTCON 10 58. COMMA , 59. INTCON 20 60. RPARENT ) 61. RPARENT ) 62. RPARENT ) 63. SEMICN ; 64. RBRACE } | | | | | | |
| 五、实验总结 问题一：输入输出问题 **问题陈述：**  本实验使用C++编写，最开始使用ifostream时，使用的常见的读入语句 ifo<<char; 但是这种会自动忽略空格、换行符等，为单词识别和行数记录带来不变。  **解决办法：**  经学习了解到read的用法，即ifo.read((char\*)&ch,1)读入一个字符，其中的1表示读取一个字节，read执行时，会自动向后移动ifo读入指针，故写起来也很方便。 问题二：向后退一个字节问题 **问题陈述：**  我的第一版程序中的getsym()结束时，始终保持ch为下一个将要处理的字符，但因此也带来不便（冗余代码多）。故第二版中添加了回退的思想，即如果多读了一个字节判断，则回退即可，封装为函数roll\_back(int x)，x表示文件指针后退的字节个数。但是有一个奇怪的问题出现了，一旦调用roll\_back()函数后，再调用getch()函数，这个时候函数指针本来应该后移动一个，但是确移动了12个，导致输出一直不正确。  **解决办法：**  经过向大佬请教，才发现在对于某些编译器，默认以文件文本的方式读取，在移动指针时会受到编译器优化问题导致移动不准确，故设置读取方式时，需要设置为二进制，即ifp.open(inFile, ios::binary) 即可。 | | | | | | |