

**教学上机实验报告**

**课程名称：** 编译原理

**任课教师：** 张建春老师

**学生学号：**  361909020429

**学生姓名：**  马银宝

**专业班级：** 计科合1904班

**2021 ～ 2022学年 第 一 学期**

|  |
| --- |
| **河南理工大学**  **教学上机实验报告评价分值标准** |
| |  |  |  |  |  |  |  |  |  | | --- | --- | --- | --- | --- | --- | --- | --- | --- | | 序号 | 评价指标 | 分值 | 评价等级及参考分值 | | | | | 评价分 | | 优 | 良 | 中 | 合格 | 差 | | 1 | 实验报告内容完整充实 | 10 | 10 | 8 | 7 | 6 | 3 |  | | 2 | 实验内容书写规范、字迹工整认真 | 10 | 10 | 8 | 7 | 6 | 3 |  | | 3 | 实验过程叙述详细、概念正确，语言表达准确，结构严谨，调理清楚，逻辑性强，自己努力完成，没有抄袭。 | 30 | 30 | 26 | 23 | 20 | 10 |  | | 4 | 对实验过程中存在的问题分析详细透彻、深刻、全面、规范、，结合实验内容，有自己的个人见解和想法，并能结合该实验提出相关问题，给出解决方法。 | 30 | 30 | 26 | 23 | 20 | 10 |  | | 5 | 实验结果、分析和结论正确无误 | 20 | 20 | 17 | 15 | 13 | 6 |  | | 总得分 | | | | | | | |  |   签名（签章）：  日期： 年 月 日 |
|  |

|  |
| --- |
| **河南理工大学教学上机实验报告** |
| 上机时间 2021 年 11月 18 日 |
| **实验题目：**  实验二-词法分析器的设计 |
| **实验目的和要求：**  目的**：**  理解词法分析在编译程序中的作用； 加深对有穷自动机模型的理解；掌握词法分析程序的实现方法和技术  要求**：**  （1）待分析的简单的语法  关键字： if else for while main return int  运算符和界符： = + - \* / < <= > >= <> = ; ( ) #  其他单词是标识符 id 和整型常数 num，通过以下正规式定义：  id=l(l|d)\*  num=dd\*  空格、注释：在词法分析中要去掉。  （2）设计各种单词符号对应的种别编码  该程序实现词法分析，从文件 data.txt 中读取一段小程序，分解出一个个的单词，其中有关键词，有界符、运算符等等，代码还需实现去掉空格、回车、注释等等情况，最后的输出结果是以单词二元组（单词种别码，单词自身的值）  的形式输出。  主要的函数有：  char m\_getch() 从输入缓冲区读取一个字符到 ch 中  void getbc( ) 去掉空白字符  void concat( ) 拼接单词  int letter( ) 判断输入字符是否是字母  int digit( ) 判断输入字符是否是数字  int reserve( ) 检索关键字表格，判断单词是否为关键字  void retract( ) 回退一个字符  WORD \* scanner( ) 词法扫描程序，返回值是二元组  data.txt 文件中存放的源代码为：  int main(){  int i, sum=0;//循环开始  for(i=1; i<=100; i++){  sum += i;  }  printf("%d\n",sum);  return 0;  }  ++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++  +++++++++++++++++++++++++++++++++++++++++++++++++++++++  （3）编写词法分析器，可对输入的源 |
| **实验过程：**   1. 首先我定义了各个标识符、关键字等字符或者字符串的种别码如下   */\*\*  \* 1 关键字  \* 2 标识符  \* 3 算数运算符  \* 4 逻辑运算符  \* 5 关系运算符  \* 6 比较运算符  \* 7 赋值运算符  \* 8 自增运算符  \* 9 自减运算符  \* 10 数据类型符号  \* 11 边界符  \* 12 转义字符  \* 13 常数  \* \*/*   1. 然后建立一个java文件，并命名为SecondExperiment 2. 其次编写子程序四个个子函数*sReserve(String Reserve) 、sLetter(char lsymbol)、Is\_Digit(char symbol) 、IsCompareVoSymbol（char lsymbo）*分别返回布尔值来判断是否为关键字、字母、数字和比较运算符或者赋值运算符，是返回‘真’，否则返回‘假’ 3. 接下来编写识别过程函数*AnalysisProcess(String PathName)*，判断关键字、运算符以及标识符等等，其中*AnalysisProcess(String PathName)，*函数调用（3）中的子函数完成整个过程的识别，最后编写主程序入口*main(String[] args)* 4. 具体的程序代码如下   import com.mysql.jdbc.Connection;  import java.io.\*; import java.sql.DriverManager; import java.sql.ResultSet; import java.sql.SQLException; import java.sql.Statement; import java.util.Scanner;  */\*\*  \* 1 关键字  \* 2 标识符  \* 3 算数运算符  \* 4 逻辑运算符  \* 5 关系运算符  \* 6 比较运算符  \* 7 赋值运算符  \* 8 自增运算符  \* 9 自减运算符  \* 10 数据类型符号  \* 11 边界符  \* 12 转义字符  \* 13 常数  \* \*/* public class SecondExperiment {   */\*\*  \* IsReserve(String Reserve)  \* 从数据库中查询关键字并判断是所传参数是否为关键字  \*/* public static boolean IsReserve(String Reserve) {  Connection con = null;  ResultSet resultSet = null;  Statement s = null;  try {  Class.*forName*("com.mysql.jdbc.Driver");  con = (Connection) DriverManager.*getConnection*("jdbc:mysql://localhost:3306/java", "root" +  "", "123456");  String sql = "select \* from reserve";  s = con.createStatement();  resultSet = s.executeQuery(sql);  while (resultSet.next()) {  String name = resultSet.getNString("reserveName");  if (name.equals(Reserve)) {  return true;  }  }  } catch (SQLException throwables) {  throwables.printStackTrace();  } catch (ClassNotFoundException e) {  e.printStackTrace();  } finally {  try {  con.close();  } catch (SQLException e) {  e.printStackTrace();  }  try {  s.close();  } catch (SQLException e) {  e.printStackTrace();  }  try {  resultSet.close();  } catch (SQLException e) {  e.printStackTrace();  }  }  return false;  }   */\*\*  \* IsLetter(char x)  \* 判断是否是字母函数  \*/* public static boolean IsLetter(char symbol) {  if ((symbol >= 'a' && symbol < 'z') || (symbol > 'A' && symbol < 'Z') || symbol == '\_') {  return true;  } else {  return false;  }  }   */\*\*  \* IsDigit(char symbol)  \* 判断是否是数字函数  \*/* public static boolean IsDigit(char symbol) {  if ((symbol >= '0' && symbol <= '9') || symbol == '.') {  return true;  } else {  return false;  }  }   */\*\*  \* IsDataTypeSymbol(char symbol)  \* 判断是哪种数据类型符号  \*/* public static boolean IsDataTypeSymbol(char symbol) {  if (symbol == '%' || symbol == 'd' || symbol == 'c' || symbol == 'l' || symbol == 'f' || symbol == 's') {  return true;  } else {  return false;  }  }   */\*\*  \* IsLogicalSymbol  \* 判断参数字符是否为逻辑运算符  \*\*/* public static boolean IsLogicalSymbol(char symbol) {  if (symbol == '&' || symbol == '|' | symbol == '!'){  return true;  }  else {  return false;  }  }   */\*\*  \* IsCompareVoSymbol(char symbol)  \* 判断是哪种比较运算符  \*/* public static boolean IsCompareVoSymbol(char symbol) {  if (symbol == '<' || symbol == '=' || symbol == '>') {  return true;  } else {  return false;  }  }   */\*\*  \* AnalysisProcess(String PathName)  \* 分析过程函数  \*/* public static void AnalysisProcess(String PathName) {  char ch;  String line\_string = "";  File file = new File(PathName);  if (file.exists()) {  try (Scanner input = new Scanner(file)) {  while (input.hasNextLine()) {  line\_string = input.nextLine();  for (int p = 0; p < line\_string.length(); p++) {  ch = line\_string.charAt(p);  if (*IsLetter*(ch)) {  String strToken = "";  while (*IsDigit*(ch) || *IsLetter*(ch)) {  strToken = strToken + ch;  ch = line\_string.charAt(++p);  }  --p;  if (*IsReserve*(strToken)) {  System.*out*.println("(1\t,\t" + strToken+"\t)");  } else {  System.*out*.println("(2\t,\t" + strToken+"\t)");  }  } else if (Character.*isDigit*(ch)) {  String strToken\_Number = "";  while (*IsDigit*(ch)) {  strToken\_Number = strToken\_Number + ch;  p = p + 1;  ch = line\_string.charAt(p);  }  --p;  System.*out*.println("(13\t,\t" + strToken\_Number+"\t)");  } else if (*IsCompareVoSymbol*(ch)) {  String strTokenCompare = "";  while (*IsCompareVoSymbol*(ch)) {  strTokenCompare = strTokenCompare + ch;  p = p + 1;  ch = line\_string.charAt(p);  }  --p;  switch (strTokenCompare) {  case "=":  System.*out*.println("(7\t,\t" + strTokenCompare+"\t)");  break;  case "<":  System.*out*.println("(6\t,\t" + strTokenCompare+"\t)");  break;  case ">":  System.*out*.println("(6\t,\t" + strTokenCompare+"\t)");  break;  case ">=":  System.*out*.println("(6\t,\t" + strTokenCompare+"\t)");  break;  case "<=":  System.*out*.println("(6\t,\t" + strTokenCompare+"\t)");  break;  default:  System.*out*.println("(13\t,\t其他不能识别的字符或者字符串\t)");  break;  }  } else if (ch == ';') {  System.*out*.println("(11\t,\t" + ch+"\t)");  } else if (ch == ',') {  System.*out*.println("(11\t,\t" + ch+"\t)");  } else if (ch == '(') {  System.*out*.println("(11\t,\t" + ch+"\t)");  } else if (ch == ')') {  System.*out*.println("(11\t,\t" + ch+"\t)");  } else if (ch == '{') {  System.*out*.println("(11\t,\t" + ch+"\t)");  } else if (ch == '}') {  System.*out*.println("(11\t,\t" + ch+"\t)");  } else if (ch == '"') {  System.*out*.println("(11\t,\t" + ch+"\t)");  } else if (ch == '\*') {  System.*out*.println("(3\t,\t" + ch+"\t)");  } else if (ch == '\\') {  p = p + 1;  System.*out*.println("(12\t,\t" + line\_string.charAt(p)+"\t)");  } else if(*IsLogicalSymbol*(ch)) {  String strTokenLogical = "";  while(*IsLogicalSymbol*(ch)){  strTokenLogical=strTokenLogical+ch;  p=p+1;  ch=line\_string.charAt(p);  }  --p;  if("!".equals(strTokenLogical)){  System.*out*.println("(4\t,\t"+strTokenLogical+"\t)");  }  else if("||".equals(strTokenLogical)){  System.*out*.println("(4\t,\t"+strTokenLogical+"\t)");  }else if("&&".equals(strTokenLogical)){  System.*out*.println("(4\t,\t"+strTokenLogical+"\t)");  }  else {  }  }  else if (ch == '+') {  String strToken\_AutoIncrease = "";  while (ch == '+') {  strToken\_AutoIncrease = strToken\_AutoIncrease + ch;  p = p + 1;  try {  ch = line\_string.charAt(p);  } catch (Exception exception) {  break;  }  }  --p;  if (strToken\_AutoIncrease.length() == 1) {  System.*out*.println("(3\t,\t" + strToken\_AutoIncrease+"\t)");  break;  } else if (strToken\_AutoIncrease.length() == 2) {  System.*out*.println("(8\t,\t" + strToken\_AutoIncrease+"\t)");  } else {  }  } else if (ch == '-') {  String strTokenAutoReduce = "";  while (ch == '-') {  strTokenAutoReduce = strTokenAutoReduce + ch;  p = p + 1;  try {  ch = line\_string.charAt(p);  } catch (Exception exception) {  break;  }  }  --p;  if ("-".equals(strTokenAutoReduce)) {  System.*out*.println("(3\t,\t" + strTokenAutoReduce+"\t)");  } else if ("--".equals(strTokenAutoReduce)) {  System.*out*.println("(9\t,\t" + strTokenAutoReduce+"\t");  } else {  }  } else if (*IsDataTypeSymbol*(ch)) {  String strTokenDataSymblo = "";  while (*IsDataTypeSymbol*(ch)) {  strTokenDataSymblo = strTokenDataSymblo + ch;  p = p + 1;  ch = line\_string.charAt(p);  }  p--;  switch (strTokenDataSymblo) {  case "%d":  System.*out*.println("(10\t,\t" + strTokenDataSymblo+"\t)");  break;  case "%lf":  System.*out*.println("(10\t,\t" + strTokenDataSymblo+"\t");  break;  case "%f":  System.*out*.println("(10\t,\t" + strTokenDataSymblo+"\t)");  break;  case "%c":  System.*out*.println("10\t,\t" + strTokenDataSymblo+"\t)");  break;  case "%s":  System.*out*.println("10\t,\t" + strTokenDataSymblo+"\t)");  break;  default:  break;  }  } else if (ch == '/') {  String strTokenExplain = "";  while (ch == '/') {  strTokenExplain = ch + strTokenExplain;  p = p + 1;  ch = line\_string.charAt(p);  }  p = line\_string.length();  } else if (Character.*isWhitespace*(ch)) {  } else {  }  }  }  } catch (FileNotFoundException e) {  e.printStackTrace();  }  } else {  System.*out*.println("输入的文件路径错误或者不存在，请重新输入！！！！！！");  }  }   */\*\*  \* C:\Users\ml\Desktop\data.txt  \* main(String[] args)  \* 主程序入口  \*/* public static void main(String[] args) {   while (true) {  String flag = "yes";  Scanner input = new Scanner(System.*in*);  System.*out*.println("请输入分析文件的绝对路径(退出请输入'yes')!");  String pathname = input.nextLine();  if (pathname.equals(flag)) {  System.*out*.println("即将退出程序，欢迎下次使用！！！！！！");  break;  }  SecondExperiment.*AnalysisProcess*(pathname);  }   } } |
| **实验结果：**  **C:\Users\ml\Desktop\data.txt**  **(1 , int )**  **(1 , main )**  **(11 , ( )**  **(11 , ) )**  **(11 , { )**  **(1 , int )**  **(2 , i )**  **(11 , , )**  **(2 , sum )**  **(7 , = )**  **(13 , 0.5 )**  **(11 , ; )**  **(1 , for )**  **(11 , ( )**  **(2 , i )**  **(7 , = )**  **(13 , 1 )**  **(11 , ; )**  **(2 , i )**  **(6 , <= )**  **(13 , 100 )**  **(11 , ; )**  **(2 , i )**  **(8 , ++ )**  **(11 , ) )**  **(11 , { )**  **(2 , sum )**  **(3 , + )**  **(11 , } )**  **(1 , printf )**  **(11 , ( )**  **(11 , " )**  **(10 , %d )**  **(12 , n )**  **(11 , " )**  **(11 , , )**  **(2 , sum )**  **(11 , ) )**  **(11 , ; )**  **(1 , return )**  **(13 , 0 )**  **(11 , ; )**  **(11 , } )**  **请输入分析文件的绝对路径(退出请输入'yes')!** |
| **实验分析：**  本次最大的收获是自己通过java语言完成了词法分析器的过程，也理解了词法分析的整个过程是如何运行的。下面我将根据具体典型过程具体分析。  识别关键字和自定义的标识符：这部分中我将标识符和关键字的识别放在了一起。在识别过程函数*AnalysisProcess(String PathName)*中先调用子函数*sLetter(char lsymbol)*判断识别的是否为“字母”，如果是进入if语句。在if语句中嵌套while循环用于判断接下来要识别的字符是否满足以上条件，如果满足条件则反复循环并最终识别整个字符串。While循环结束使用--p让下标回退，如果不回退，则退出循环的时候又会在最外层的for循环中做p++操作，这样就会将下一个识别字符的首字母没有识别上。最好来使用if语句调用子函数*sReserve(String Reserve)* 来判断刚才字符串是否为关键字，是打印输出关键字和其所对应的种别码，不是打印的是标识符和其所对应的种别码。  其他的识别都和上面的识别大差不差，再来说一下注释块的识别。首先还是使用if语句判断是否为‘/’，是就进入内层循环。内层循环中和上面的一样，还是使用while循环，条件和if中的一样。如下图    但是最关键的是想要跳过注释使用语句，直接将识别字符串的最后一位的下标也就是识别字符串的长度赋值给p使其直接跳过注释后面的字符串。  本实验的特别之处在于我在子函数*sReserve(String Reserve)* 中首先将“关键字”保存在了数据库中，然后通过查询数据库直接返回“关键字”来判断传参字符串是否为“关键字”，是返回“真”，不是返回“假”。这样比较方便操作，否则c语言的关键字很多，使用字符数组保存会使得查询判断很不友好，代码也不简洁。    综上所述，本实验的关键之处是在写代码的过程中一定要理清下标p和识别字符之间的关系，只有这样本实验才能顺利完成。  **出错分析：**  当然，本次实验也遇到了麻烦，首先一个典型的例子就是没有理清下标p和其所识别字符的之间的关系及时地做出回退操作导致有的标识符或者关键字等的首字母没有被识别出来。是我当时没有理解为什么要回退。本来老师课上讲了但是而我没有听懂，下课又问了老师以为会了，然后写代码的时候又出了问题。最后又抱着写的代码才知道自己出错在了哪里。实践出真知！  其次一个是在老师给定要识别的文件中的最后两行是连续的“++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++”。这个我放在了自增运算符“++”和“+”中一起识别了。代码比较简单，但是遇到了下面的问题：      很明显程序在识别出c语言文件的最后一个字符“}”的时候出错了，原因在于“}”后面有连续的“++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++”，所以报出了超出字符串索引的错误。但是java中的String没有长度限制啊？于是我在网上找到了答案：说即使String类型的数据没有长度限制但是String类型的数据太长会超出JVM虚拟机的内存空间，所以会报错。找到原因所在，所以我找到如下解决方案。异常处理：  。最后完美解决异常并且打印出了有异常的字符串。 |
| **实验成绩：**  日期： 年 月 日 |