**黑龙江大学**

**实 验 报 告**

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| **课程名称** | 软件构造 | | | | | | |
| **实验项目名称** | 词法分析程序的构造 | | | | | | |
| **实验时间**  **（日期及节次）** | 4月11日第5-6节 | | | | | | |
| **专业** | 软件工程 | | **学生所在学院** | | | 软件学院 | |
| **年级** | 2022级 | | **学号** | | | 20225958 | |
| **姓名** | 李嘉富 | | **指导教师** | | | 徐为 | |
| **实验室名称** | 4-108 | | | | | | |
| **实验成绩** | **预习情况** | **操作技术** | | **实验报告** | **附加：综合创新能力** | | **实验**  **综合成绩** |
|  |  | |  |  | |  |
| **教师签字** |  | | | | | | |

**黑龙江大学教务处**

1. 实验目的

1、了解词法分析的任务。

2、掌握构造识别各类单词自动机的方法及过程。

3、掌握符号表的建立及单词的分类方法。

4、掌握词法分析程序的基本原理与构造方法。

二、实验环境

编程环境：Visual Studio 2022

运行环境: Visual Studio 2022

三、实验内容

实验以自动机理论为基础，通过各种算法，采用了字符串以及数组等数据结构。主要思想为关键字和id通过数组查找的形式，进行赋值，凭此识别，而界符和运算符则通过switch识别，进行定点分类。

Target文件（原代码）如下：

Int main()

{

int A = 555;

int Asdb = 1;

char B = 71;

if(A>B){

5DC = 333;

B = A+Asdb;

}

Return 0;

}@//@为结束符号

按下运行按钮之后，得出结果Target和id

四、实验结果（测试用例、实验结果）

Target：（存储已经经过词法分析的数据）

(1,1,A)

(1,2,Asdb)

(1,3,B)

(1,4,DC)

(5,( )

(5,) )

(3,main)

(4,+)

(3,int)

(2,0)

(2,1)

(2,333)

(2,555)

(2,5)

(3,char)

(2,71)

(5,{)

(5,;)

(4,=)

(3,if)

(5,))

(4,>)

(3,return)

id：（存储第二类即id数据）

A

Asdb

B

DC

五、实验总结

在这次实验中，我学会了从文件中绕过空格读取的能力和如何使文件中的数据只存在一种。通过实验，我进一步了解到了有穷自动机（DFA）运行的基本逻辑。虽然经我的拙劣模仿能够运行，但远远不能按照矩阵的形式操作，希望我在以后的实验中能够学习更多，争取完成老师的所有任务。

附录（源代码）

总源代码：

package com.meteor.test;  
  
import java.io.FileInputStream;  
import java.io.FileOutputStream;  
import java.io.IOException;  
import java.nio.charset.StandardCharsets;  
import java.util.ArrayList;  
import java.util.HashMap;  
  
public class Test *{* // 定义状态转换矩阵，第一列为状态，第一行为字符（可以使用稀疏数组优化空间）  
 public static final String*[][] stateMatrix* = new String*[][] {  
 {*"" ,"i" ,"n" ,"t" , "f","m","a","c","h","r" , "e","u" , "+" , "-", "\*", "/", "=",">" ,"," ,";" ,"(" ,")" ,"{" ,"}" *}*,  
 *{*"0" ,"1" , "" , "" , "" ,"8", "","4", "","13", "" , "" , "19","19","19","19","19","19","20","20","20","20","20","20"*}*,  
 *{*"1" ,"" ,"2" , "" ,"12","" , "","" , "", "" , "" , "" , "" , "" , "" , "" , "" , "" , "" , "" , "" , "" , "" , "" *}*,  
 *{*"8" ,"" ,"" , "" , "" ,"" ,"9","" , "", "" , "" , "" , "" , "" , "" , "" , "" , "" , "" , "" , "" , "" , "" , "" *}*,  
 *{*"4" ,"" ,"" , "" , "" ,"" ,"" ,"" ,"5", "" , "" , "" , "" , "" , "" , "" , "" , "" , "" , "" , "" , "" , "" , "" *}*,  
 *{*"13","" ,"" , "" , "" ,"" ,"" ,"" ,"" , "" ,"14", "" , "" , "" , "" , "" , "" , "" , "" , "" , "" , "" , "" , "" *}*,  
 *{*"19","" ,"" , "" , "" ,"" ,"" ,"" ,"" , "" , "" , "" , "" , "" , "" , "" , "" , "" , "" , "" , "" , "" , "" , "" *}*,  
 *{*"20","" ,"" , "" , "" ,"" ,"" ,"" ,"" , "" , "" , "" , "" , "" , "" , "" , "" , "" , "" , "" , "" , "" , "" , "" *}*,  
 *{*"2" ,"" ,"" ,"3" , "" ,"" ,"" ,"" ,"" , "" , "" , "" , "" , "" , "" , "" , "" , "" , "" , "" , "" , "" , "" , "" *}*,  
 *{*"12","" , "" , "" , "" ,"" ,"" ,"" ,"" , "" , "" , "" , "" , "" , "" , "" , "" , "" , "" , "" , "" , "" , "" , "" *}*,  
 *{*"9" ,"10", "" , "" , "" ,"" ,"" ,"" ,"" , "" , "" , "" , "" , "" , "" , "" , "" , "" , "" , "" , "" , "" , "" , "" *}*,  
 *{*"5" , "" , "" , "" , "" ,"" ,"6","" ,"" , "" , "" , "" , "" , "" , "" , "" , "" , "" , "" , "" , "" , "" , "" , "" *}*,  
 *{*"14", "" , "" ,"15", "" ,"" ,"" ,"" ,"" , "" , "" , "" , "" , "" , "" , "" , "" , "" , "" , "" , "" , "" , "" , "" *}*,  
 *{*"3" , "" , "" , "" , "" ,"" ,"" ,"" ,"" , "" , "" , "" , "" , "" , "" , "" , "" , "" , "" , "" , "" , "" , "" , "" *}*,  
 *{*"10", "" ,"11", "" , "" ,"" ,"" ,"" ,"" , "" , "" , "" , "" , "" , "" , "" , "" , "" , "" , "" , "" , "" , "" , "" *}*,  
 *{*"6" , "" , "" , "" , "" ,"" ,"" ,"" ,"" ,"7" , "" , "" , "" , "" , "" , "" , "" , "" , "" , "" , "" , "" , "" , "" *}*,  
 *{*"15", "" , "" , "" , "" ,"" ,"" ,"" ,"" , "" , "" ,"16", "" , "" , "" , "" , "" , "" , "" , "" , "" , "" , "" , "" *}*,  
 *{*"11", "" , "" , "" , "" ,"" ,"" ,"" ,"" , "" , "" , "" , "" , "" , "" , "" , "" , "" , "" , "" , "" , "" , "" , "" *}*,  
 *{*"7" , "" , "" , "" , "" ,"" ,"" ,"" ,"" , "" , "" , "" , "" , "" , "" , "" , "" , "" , "" , "" , "" , "" , "" , "" *}*,  
 *{*"16", "" , "" , "" , "" ,"" ,"" ,"" ,"" ,"17", "" , "" , "" , "" , "" , "" , "" , "" , "" , "" , "" , "" , "" , "" *}*,  
 *{*"17", "" ,"18", "" , "" ,"" ,"" ,"" ,"" , "" , "" , "" , "" , "" , "" , "" , "" , "" , "" , "" , "" , "" , "" , "" *}*,  
 *{*"18", "" , "" , "" , "" ,"" ,"" ,"" ,"" , "" , "" , "" , "" , "" , "" , "" , "" , "" , "" , "" , "" , "" , "" , "" *}*,  
 *}*;  
 // 初态  
 public static final String *initialState* = "0";  
 public static String *currentState* = "0";  
 public static HashMap*<*String, String*> map* = new HashMap*<>()*;  
 // 标识符位置  
 public static int *index* = 1;  
  
 // 定义终态集合  
 public static final String*[] finalState* = *{*"3","7","11","12","18","19","20"*}*;  
  
 // 获取接收字符后的状态  
 public static String getNextCurrentState*(*String accept, String currentState*) {* // 如果存在状态集中，则获取状态值  
 if*(isExist(*accept*)) {* int currentStateRow = *getCurrentStateRow(*currentState*)*;  
 int currentStateCol = *getCurrentStateCol(*accept*)*;  
 if*(*currentStateRow != -1 && currentStateCol != -1*) {* return *stateMatrix[*currentStateRow*][*currentStateCol*]*;  
 *}  
 }* return null;  
 *}* // 判断是否存在状态集中  
 public static boolean isExist*(*String s*) {* for *(*int i = 1; i < *stateMatrix[*0*]*.length; i++*) {* if*(*s.equals*(stateMatrix[*0*][*i*])) {* return true;  
 *}  
 }* return false;  
 *}*// // 根据初态获取当前状态的值  
// public static String getCurrentState(String s) {  
// // 如果存在状态集中，则获取状态值  
// if(isExist(s)) {  
// for (int i = 0; i < stateMatrix[0].length; i++) {  
// if(s.equals(stateMatrix[0][i]) && !stateMatrix[0][i].equals("")) {  
// return stateMatrix[1][i];  
// }  
// }  
// }  
// return null;  
// }  
  
 // 获取当前状态的列  
 public static int getCurrentStateCol*(*String s*) {* // 如果存在状态集中，则获取状态值  
 if*(isExist(*s*)) {* for *(*int i = 0; i < *stateMatrix[*0*]*.length; i++*) {* if*(*s.equals*(stateMatrix[*0*][*i*])* && !*stateMatrix[*0*][*i*]*.equals*(*""*)) {* return i;  
 *}  
 }  
 }* return -1;  
 *}* // 获取当前状态的行  
 public static int getCurrentStateRow*(*String s*) {* // 如果存在状态集中，则获取状态值  
 for *(*int i = 1; i < *stateMatrix*.length; i++*) {* if*(*s.equals*(stateMatrix[*i*][*0*])) {* return i;  
 *}  
 }* return -1;  
 *}*// // 根据初态的值获取所在行的 所有字符  
// public static List<String> getNextStateValue(String s) {  
// ArrayList<String> list = new ArrayList<>();  
// int row = -1;  
// for (int i = 1; i < stateMatrix.length; i++) {  
// if(s.equals(stateMatrix[i][0])) {  
// row = i;  
// break;  
// }  
// }  
// for (int i = row; row != -1 && i < stateMatrix[row].length; i++) {  
// if(!stateMatrix[row][i].equals("")) {  
// list.add(stateMatrix[row][i]);  
// }  
// }  
// return list;  
// }  
  
 // 判断是否为终态  
 public static boolean isFinalState*(*String s*) {* for *(*int i = 0; i < *finalState*.length; i++*) {* if*(*s.equals*(finalState[*i*])) {* return true;  
 *}  
 }* return false;  
 *}*// public static void test() {  
// String currentState = getCurrentState("i");  
// List<String> n = getNextStateValue(currentState);  
// for (int i = 0; i < n.size(); i++) {  
// System.out.println(n.get(i));  
// }  
 //System.out.println("sbbbbbb");  
// }  
  
 public static void main*(*String*[]* args*)* throws Exception *{* //test();  
  
 // 是否是标识符  
 boolean flag = false;  
 // 是否是数字  
 boolean isShuZi = false;  
 boolean end = false;  
  
 // 创建一个文件字节输入流（从文件 -> 内存）  
 FileInputStream fileInputStream = new FileInputStream*(*"source.txt"*)*;  
 ArrayList*<*String*>* list = new ArrayList*<>()*;  
 int read = 0;  
 while*((*read = fileInputStream.read*())* != -1*) {* list.add*(((*char*)* read + ""*))*;  
 *}* // 创建一个文件字节输出流（从内存 -> 文件）  
 FileOutputStream outputStream = new FileOutputStream*(*"target.txt"*)*;  
  
 // 创建一个文件字节输出流（从内存 -> 文件）  
 FileOutputStream outIdentiflier = new FileOutputStream*(*"identifier.txt"*)*;  
  
 // 用于存放文件内容  
 String*[]* array = list.toArray*(*new String*[*list.size*()])*;  
  
 // 遍历数组的当前指针  
 int current = 0;  
 // 存放字符串  
 String token = "";  
 // 遍历数组内容  
 while*(*!*(*array*[*current*]*.equals*(*"@"*))) {* // 如果当前为空格、换行、回车，则继续循环  
 if*(*array*[*current*]*.equals*(*" "*)* || array*[*current*]*.equals*(*"\n"*)* || array*[*current*]*.equals*(*"\r"*)* || array*[*current*]*.equals*(*"\t"*)) {* current++;  
 continue;  
 *}* // 1. 判断是否存在状态矩阵中  
 if*(isExist(*array*[*current*])* && !flag*) {* token += array*[*current*]*;  
 // 1.1 获取接收字符后的状态值  
 String nextCurrentState = *getNextCurrentState(*array*[*current*]*, *currentState)*;  
 *currentState* = nextCurrentState;  
 // 单独检查是否为终态  
 String nextCurrentState1 = *getNextCurrentState(*array*[*current*]*, "0"*)*;  
 if*(isFinalState(*nextCurrentState1*)* && token.length*()* > 1*) {* if*(*!*map*.containsKey*(*token.substring*(*0,token.length*()*-1*))) {  
 map*.put*(*token.substring*(*0,token.length*()*-1*)*, 1+ "：" + *index)*;  
 outIdentiflier.write*((index* + "："*)*.getBytes*(*StandardCharsets.*UTF\_8))*;  
 *index*++;  
 outIdentiflier.write*(((*token.substring*(*0,token.length*()*-1*)* + "\n"*)*.getBytes*(*StandardCharsets.*UTF\_8)))*;  
 System.*out*.println*(*"(" + "1：" + *index* + "： "+ token + ")"*)*;  
 *}  
 }* // 如果查询状态并不是下一个状态  
 if*(*nextCurrentState == null || nextCurrentState.equals*(*""*)) {* flag = true;  
 *currentState* ="0";  
 token = "";  
 *}* if*(*nextCurrentState != null&& !nextCurrentState.equals*(*""*)) {* // 1.2 判断是否为终态  
 boolean finalState = *isFinalState(*nextCurrentState*)*;  
 // a. 如果为终态  
 if*(*finalState*) {  
 insert(*token*)*;  
 *currentState* = "0";  
 token = "";  
 *}* current++;  
 *}  
 }*else *{* // 判断是数字 但是没用标识符状态  
 while*(*Character.*isDigit(*array*[*current*]*.charAt*(*0*))* && !flag*) {* token += array*[*current*]*;  
 current++;  
 isShuZi = true;  
 *}* if*(*isShuZi*){  
 map*.put*(*token,"2"*)*;  
 System.*out*.println*(*"(" + "2：" +token + ")"*)*;  
 token = "";  
 isShuZi = false;  
 *}* // 标识符：遇到非字符 or 数字停止  
 if*(*Character.*isAlphabetic(*array*[*current*]*.charAt*(*0*))* || Character.*isDigit(*array*[*current*]*.charAt*(*0*))) {* token += array*[*current++*]*;  
 // 设置状态标志  
 flag = true;  
 end = true;  
 *}*else *{* if*(*flag && end*) {* // 解除标识符状态标志  
 if*(*!*map*.containsKey*(*token*)) {  
 map*.put*(*token,1+ "：" + *index)*;  
 outIdentiflier.write*((index*+ "："*)*.getBytes*(*StandardCharsets.*UTF\_8))*;  
 *index*++;  
 outIdentiflier.write*((*token + "\n"*)*.getBytes*(*StandardCharsets.*UTF\_8))*;  
 System.*out*.println*(*"(" + "1：" + *index* + "："+token + ")"*)*;  
 *}* flag = false;  
 end = false;  
 token = "";  
 *}*else *{* flag = false;  
 *}  
 }  
 }  
 }* // 遍历map集合  
 *map*.forEach*((*key , value*)* ->*{* try *{* outputStream.write*((*value + "："*)*.getBytes*(*StandardCharsets.*UTF\_8))*;  
 outputStream.write*((*key + "\n"*)*.getBytes*(*StandardCharsets.*UTF\_8))*;  
 *}* catch *(*IOException e*) {* e.printStackTrace*()*;  
 *}  
 })*;  
 *}* public static void insert*(*String s*)* throws Exception *{* if*(isKeywords(*s*)) {  
 map*.put*(*s,"3"*)*;  
 System.*out*.println*(*"(" + "3：" + s + ")"*)*;  
 *}*else if*(isOperator(*s*)) {  
 map*.put*(*s, "4"*)*;  
 System.*out*.println*(*"(" + "4：" + s + ")"*)*;  
 *}*else if*(isSeparator(*s*)) {  
 map*.put*(*s, "5"*)*;  
 System.*out*.println*(*"(" + "5：" + s + ")"*)*;  
 *}  
 }* // 判断是否为关键字  
 public static boolean isKeywords*(*String s*)* throws Exception *{* // 创建一个文件字节输入流（从文件 -> 内存）  
 FileInputStream fileInputStream = new FileInputStream*(*"keywords.txt"*)*;  
 ArrayList*<*String*>* list = new ArrayList*<>()*;  
 int read = 0;  
 String str = "";  
 while*((*read = fileInputStream.read*())* != -1*) {* str += *(*char*)*read;  
 *}* // 用于存放文件内容  
 String*[]* keyArray = str.split*(*","*)*;  
 for*(*int i=0 ; i< keyArray.length; i++*){* if*(*s.equals*(*keyArray*[*i*])) {* return true;  
 *}  
 }* return false;  
 *}* // 判断当前字符是否是运算符  
 public static boolean isOperator*(*String s*)* throws Exception *{* // 创建一个文件字节输入流（从文件 -> 内存）  
 FileInputStream fileInputStream = new FileInputStream*(*"operator.txt"*)*;  
 ArrayList*<*String*>* list = new ArrayList*<>()*;  
 int read = 0;  
 while*((*read = fileInputStream.read*())* != -1*) {* list.add*(((*char*)* read + ""*))*;  
 *}* // 用于存放文件内容  
 String*[]* operator = list.toArray*(*new String*[*list.size*()])*;  
 for*(*int i=0 ; i< operator.length; i++*){* if*(*s.equals*(*operator*[*i*])) {* return true;  
 *}  
 }* return false;  
 *}* // 判断当前字符是否是分隔符  
 public static boolean isSeparator*(*String s*)* throws Exception *{* // 创建一个文件字节输入流（从文件 -> 内存）  
 FileInputStream fileInputStream = new FileInputStream*(*"separator.txt"*)*;  
 ArrayList*<*String*>* list = new ArrayList*<>()*;  
 int read = 0;  
 while*((*read = fileInputStream.read*())* != -1*) {* list.add*(((*char*)* read + ""*))*;  
 *}* // 用于存放文件内容  
 String*[]* separator = list.toArray*(*new String*[*list.size*()])*;  
 for*(*int i=0 ; i< separator.length; i++*){* if*(*s.equals*(*separator*[*i*])) {* return true;  
 *}  
 }* return false;  
 *}* // 判断是否为字母  
// public static boolean isAlphabet(String s) {  
// char c = s.charAt(0);  
// if (((c >= 'a' && c <= 'z') || (c >= 'A' && c <= 'Z'))) {  
// return true;  
// }  
// return false;  
// }  
*}*