

编译原理实验报告

班级：计科1504

学号：0902150310

姓名：马福龙

目录

[实验一 词法分析 3](#_Toc502864117)

[一、实验内容与要求 3](#_Toc502864118)

[二、实验分析与设计 3](#_Toc502864119)

[三、源代码 9](#_Toc502864120)

[四，调试结果 17](#_Toc502864121)

[五、实验总结与心得 18](#_Toc502864122)

[实验二 语法分析 19](#_Toc502864123)

[一、实验内容与要求 19](#_Toc502864124)

[二、实验分析与设计 20](#_Toc502864125)

[三、源代码 21](#_Toc502864126)

[四，调试结果 25](#_Toc502864127)

[五、实验总结与心得 25](#_Toc502864128)

[实验三 词法 语法分析 27](#_Toc502864129)

[一、实验内容与要求 27](#_Toc502864130)

[二、实验分析与设计 27](#_Toc502864131)

[三、源代码 27](#_Toc502864132)

[四，调试结果 35](#_Toc502864133)

[五、实验总结与心得 35](#_Toc502864134)

实验一 词法分析

## 一、实验内容与要求

(1)实验目的

* 根据PL/0语言的文法规范，编写PL/0语言的词法分析程序。
* 通过设计调试词法分析程序，实现从源程序中分出各种单词的方法；加深对课堂教学的理解；提高词法分析方法的实践能力。
* 掌握从源程序文件中读取有效字符的方法和产生源程序的内部表示文件的法。
* 掌握词法分析的实现方法。
* 上机调试编出的词法分析程序。

(2)实验内容

已给PL/0语言文法，输出单词符号（关键字、专用符号以及其它标记）。

(3)实验要求

* 确定编译中使用的表格、标识符与关键字的区分方法等。
* 把词法分析器设计成一个独立一遍的过程。
* 词法分析器的输出形式采用二元式序列，即：

(单词种类, 单词的值)

## 二、实验分析与设计

**思路：**通过map映射建立指定内容与编码的映射。

**FileUtil类**

建立FileUtil类应用于文件处理，读取文件并提取内容、写入文件、清空文件

类结构及主要函数：

public class FileUtil {

/\*\*

\* 文件读取到缓冲区

\* @param buffer 缓冲区

\* @param fileSrc 文件路径

\* @return true : success

\* false : filed

\*/

public static boolean readFile(StringBuffer buffer, String fileSrc) ;

/\*\*

\* 追加方式写文件

\* @param args 需要写入字符串

\* @return true : success

\* false : filed

\*/

public static boolean writeFile(String args) ; /\*\*

\* 清空文件

\*/

public static boolean clearFile() ;

}

**TypeUtil类**

/\*\*

\* 对读取字符分类相关操作,父类

\* @author mfl

\*/

public class TypeUtil {

//dec new add by mfl:

private final String keyWords[] = {"begin","call","const","do","end","if","odd","procedure","read",

"then","var","while","write"}; // 关键字数组

public final static Map<String,String> map = new HashMap();

public final String keyWordsEncoding[]={"beginsym","callsym","constsym","dosym","endsym",

"ifsym","oddsym","proceduresym","readsym","thensym","varsym","whilesym","writesym"};

public final String operatorsEncoding[]={"plus","minus","times","slash","eql","neq","lss",

"leq","gtr","geq","becomes",":"};

public final String separatorsEncoding[]={"Lparen","rparen","comma","semicolon","period"};

private final String operators[] = {"+","-","\*","/","=","<>","<","<=",">",">=",":=",":"};

public final String separtors[]={"(", ")", ",", ";", "."};

/\*\*

\* 构造函数,init

\*/

public TypeUtil(){

initialize();

}

/\*\*

\* 初始化map，建立映射编码

\*/

public void initialize(){

}

/\*\*

\* 获取map value

\* @param str map key

\* @return map value

\*/

public String getValue(String str){

return map.get(str);

}

/\*\*

\* 判断是否为字母

\* @param ch 需判断的字符

\* @return boolean

\*/

public boolean isLetter(char ch) {

}

/\*\*

\* 判断是否为数字

\* @param ch 需判断的字符

\* @return boolean

\*/

public boolean isDigit(char ch) {

}

/\*\*

\* 判断是否为关键字

\* @param s 需判断的字符串

\* @return boolean

\*/

public boolean isKeyWord(String s) {

}

/\*\*

\* 判断是否为运算符

\* @param ch 需判断的字符

\* @return boolean

\*/

public boolean isOperator(char ch)

}

/\*\*

\* 判断字符串是否为操作符

\* @param ch 字符串

\* @return boolean

\*/

public boolean isOperator1(String ch)

}

/\*\*

\* 判断是否为分隔符

\* @param ch 需判断的字符

\* @return boolean

\*/

public boolean isSeparators(char ch)

}

}

/\*\*

\* 词法分析

\* 关键字，运算符一符一类

\* 标识符，常数，分隔符各自一类

\* 运算符未处理组合运算 ++、--、+= 等

\* @author mfl

\* @version 1.1

\*/

**TestLexer类**

public class TestLexer extends TypeUtil{

public static StringBuffer buffer = new StringBuffer(); // 缓冲区

private int i = 0;

private char ch; // 字符变量，存放最新读进的源程序字符

private String strToken; // 字符数组，存放构成单词符号的字符串

public static String result;

/\*\*

\* 默认构造函数，无调用

\*/

public TestLexer() {

}

/\*\*

\* 读取指定路径文件

\* @param fileSrc 读取文件路径

\*/

public TestLexer(String fileSrc) {

}

/\*\*

\* 词法分析

\*/

public void analyse() {

}

/\*\*

\* 将下一个输入字符读到ch中，搜索指示器前移一个字符

\*/

public void getChar() {

ch = buffer.charAt(i);

i++;

}

/\*\* 检查ch中的字符是否为空白，若是则调用getChar()直至ch中进入一个非空白字符\*/

public void getBC() {

//isSpaceChar(char ch) 确定指定字符是否为 Unicode 空白字符。

//上述方法不能识别换行符

while (Character.isWhitespace(ch))//确定指定字符依据 Java 标准是否为空白字符。

getChar();

}

/\*\*将ch连接到strToken之后\*/

public void concat() {

strToken += ch;

}

/\*\* 将搜索指示器回调一个字符位置，将ch值为空白字 \*/

public void retract() {

i--;

ch = ' ';

}

/\*\*

\* 按照二元式规则写入文件

\* @param file 字符类型

\* @param s 当前字符

\*/

public void writeFile(String file,String s) {

// System.out.println(i);

System.out.println("("+file+", "+s+")");

file="("+file+","+s+")"+"\r\n";

result+=file;

FileUtil.writeFile(file);

}

}

**Main类**

/\*\*

\* main

\*/

public class MainTest {

public static void main(String[] args) {

/\*创建词法分析类\*/

TestLexer testLexer = new TestLexer("./src/input.txt");

//FileUtil.clearFile();//清空文件

testLexer.analyse();

// System.out.println("mafulong"+TestLexer.result);

}

}

**输入输出例子：**

输入：PL/0源程序。例:

const a=10;

var b,c;

begin

read(b);

c:=a+b;

write(c)

end.

输出：

(constsym,const)

(ident , a)

(eql , =)

(number, 10)

(semicolon, ;)

(varsym, var )

(ident, b)

(comma, , )

(ident, c )

(semicolon, ;)

(beginsym,begin)

(readsym, read )

(lparen, ( )

(ident, b)

(rparen, ) )

(semicolon, ;)

(ident, c )

(becomes, := )

(ident, a )

(plus, + )

(ident, b )

(semicolon, ;)

(writesym,write)

(lparen, ( )

(ident, c )

(rparen, ) )

(endsym, end )

(period, .)

## 三、源代码

/\*\*

\* main

\*/

public class MainTest {

public static void main(String[] args) {

/\*创建词法分析类\*/

TestLexer testLexer = new TestLexer("./src/input.txt");

//FileUtil.clearFile();//清空文件

testLexer.analyse();

// System.out.println("mafulong"+TestLexer.result);

}

}

import java.io.BufferedReader;

import java.io.BufferedWriter;

import java.io.File;

import java.io.FileNotFoundException;

import java.io.FileReader;

import java.io.FileWriter;

import java.io.IOException;

/\*\*

\* 文件操作

\* @author mfl

\*/

public class FileUtil {

/\*\*

\* 文件读取到缓冲区

\* @param buffer 缓冲区

\* @param fileSrc 文件路径

\* @return true : success

\*       false : filed

\*/

public static boolean readFile(StringBuffer buffer, String fileSrc) {

try {

FileReader fileReader = new FileReader(fileSrc);

BufferedReader br = new BufferedReader(fileReader);

String temp = null;

while ((temp = br.readLine()) != null) {

buffer.append(" "+temp);

}

return true;

} catch (FileNotFoundException e) {

e.printStackTrace();

} catch (IOException e) {

e.printStackTrace();

}

return false;

}

/\*\*

\* 追加方式写文件

\* @param args  需要写入字符串

\* @return  true : success

\*       false : filed

\*/

public static boolean writeFile(String args) {

try {

File file = new File("./src/output.txt");

if (!file.exists()) {

file.createNewFile();

}

FileWriter fw = new FileWriter(file.getAbsoluteFile(),true);

BufferedWriter bw = new BufferedWriter(fw);

bw.write(args);

bw.close();

return true;

} catch (IOException e) {

e.printStackTrace();

return true;

}

}

/\*\*

\* 清空文件

\*/

public static boolean clearFile() {

try {

File file = new File("./src/output.txt");

if (!file.exists()) {

file.createNewFile();

}

FileWriter fw = new FileWriter(file.getAbsoluteFile());

BufferedWriter bw = new BufferedWriter(fw);

bw.write("");

bw.close();

return true;

} catch (IOException e) {

e.printStackTrace();

return true;

}

}

}

import java.util.\*;

/\*\*

\* 对读取字符分类相关操作,父类

\* @author mfl

\*/

public class TypeUtil {

//dec new add by mfl:

private final String keyWords[] = {"begin","call","const","do","end","if","odd","procedure","read",

"then","var","while","write"}; // 关键字数组

public final static Map<String,String> map = new HashMap();

public final String keyWordsEncoding[]={"beginsym","callsym","constsym","dosym","endsym",

"ifsym","oddsym","proceduresym","readsym","thensym","varsym","whilesym","writesym"};

public final String operatorsEncoding[]={"plus","minus","times","slash","eql","neq","lss",

"leq","gtr","geq","becomes",":"};

public final String separatorsEncoding[]={"Lparen","rparen","comma","semicolon","period"};

private final String operators[] = {"+","-","\*","/","=","<>","<","<=",">",">=",":=",":"};

public final String separtors[]={"(", ")", ",", ";", "."};

/\*\*

\* 构造函数,init

\*/

public TypeUtil(){

initialize();

}

/\*\*

\* 初始化map，建立映射编码

\*/

public void initialize(){

map.clear();

for(int i=0;i<keyWords.length;i++){

map.put(keyWords[i], keyWordsEncoding[i]);

}

for(int i=0;i<operators.length;i++){

map.put(operators[i], operatorsEncoding[i]);

}

for(int i=0;i<separtors.length;i++){

map.put(separtors[i], separatorsEncoding[i]);

}

}

/\*\*

\* 获取map value

\* @param str map key

\* @return map value

\*/

public String getValue(String str){

return map.get(str);

}

/\*\*

\* 判断是否为字母

\* @param ch 需判断的字符

\* @return boolean

\*/

public boolean isLetter(char ch) {

return Character.isLetter(ch);

}

/\*\*

\* 判断是否为数字

\* @param ch 需判断的字符

\* @return boolean

\*/

public boolean isDigit(char ch) {

return Character.isDigit(ch);

}

/\*\*

\* 判断是否为关键字

\* @param s 需判断的字符串

\* @return boolean

\*/

public boolean isKeyWord(String s) {

for (int i = 0; i < keyWords.length; i++) {

if (keyWords[i].equals(s))

return true;

}

return false;

}

/\*\*

\* 判断是否为运算符

\* @param ch 需判断的字符

\* @return boolean

\*/

public boolean isOperator(char ch) {

for (int i = 0; i < operators.length; i++) {

if (Character.toString(ch).equals(operators[i]))

return true;

}

return false;

}

/\*\*

\* 判断字符串是否为操作符

\* @param ch 字符串

\* @return boolean

\*/

public boolean isOperator1(String ch) {

for (int i = 0; i < operators.length; i++) {

if (ch.equals(operators[i]))

return true;

}

return false;

}

/\*\*

\* 判断是否为分隔符

\* @param ch 需判断的字符

\* @return boolean

\*/

public boolean isSeparators(char ch) {

for (int i = 0; i < separtors.length; i++) {

if (Character.toString(ch).equals(separtors[i]))

return true;

}

return false;

}

}

/\*\*

\* 词法分析

\* 关键字，运算符一符一类

\* 标识符，常数，分隔符各自一类

\* 运算符未处理组合运算 ++、--、+= 等

\* @author mfl

\* @version 1.1

\*/

public class TestLexer extends TypeUtil{

public static StringBuffer buffer = new StringBuffer(); // 缓冲区

private int i = 0;

private char ch; // 字符变量，存放最新读进的源程序字符

private String strToken; // 字符数组，存放构成单词符号的字符串

public static String result;

/\*\*

\* 默认构造函数，无调用

\*/

public TestLexer() {

}

/\*\*

\* 读取指定路径文件

\* @param fileSrc 读取文件路径

\*/

public TestLexer(String fileSrc) {

FileUtil.readFile(buffer, fileSrc);

result="";

}

/\*\*

\* 词法分析

\*/

public void analyse() {

System.out.println(buffer.length());

strToken = ""; // 置strToken为空串

FileUtil.clearFile();//清空文件

while (i <buffer.length()) {

getChar();

// System.out.println("!!!"+ch);

getBC();

if (isLetter(ch)) { // 如果ch为字母

while (isLetter(ch) || isDigit(ch)){

concat();

getChar();

}

retract(); // 回调

if (isKeyWord(strToken)) {

writeFile(getValue(strToken),strToken);//strToken为关键字

}

else {

writeFile("ident",strToken);//strToken为标识符

}

strToken = "";

} else if (isDigit(ch)) {

while (isDigit(ch)) {//ch为数字

concat();

getChar();

}

if(!isLetter(ch)){//不能数字+字母

retract(); // 回调

writeFile("number",strToken); // 是整形

}else writeFile("error",strToken); // 非法

strToken = "";

} else if (isOperator(ch)) { //运算符

while(isOperator(ch)){

concat();

getChar();

}

retract();

if(isOperator1(strToken)){

writeFile(getValue(strToken),strToken);

}else{

writeFile("error", strToken);

}

strToken="";

}else if(isSeparators(ch)){

concat();

writeFile(getValue(strToken),strToken);

strToken="";

}else writeFile("error",ch+"");

}

}

/\*\*

\* 将下一个输入字符读到ch中，搜索指示器前移一个字符

\*/

public void getChar() {

ch = buffer.charAt(i);

i++;

}

/\*\* 检查ch中的字符是否为空白，若是则调用getChar()直至ch中进入一个非空白字符\*/

public void getBC() {

//isSpaceChar(char ch) 确定指定字符是否为 Unicode 空白字符。

//上述方法不能识别换行符

while (Character.isWhitespace(ch))//确定指定字符依据 Java 标准是否为空白字符。

getChar();

}

/\*\*将ch连接到strToken之后\*/

public void concat() {

strToken += ch;

}

/\*\* 将搜索指示器回调一个字符位置，将ch值为空白字 \*/

public void retract() {

i--;

ch = ' ';

}

/\*\*

\* 按照二元式规则写入文件

\* @param file 字符类型

\* @param s 当前字符

\*/

public void writeFile(String file,String s) {

// System.out.println(i);

System.out.println("("+file+", "+s+")");

file="("+file+","+s+")"+"\r\n";

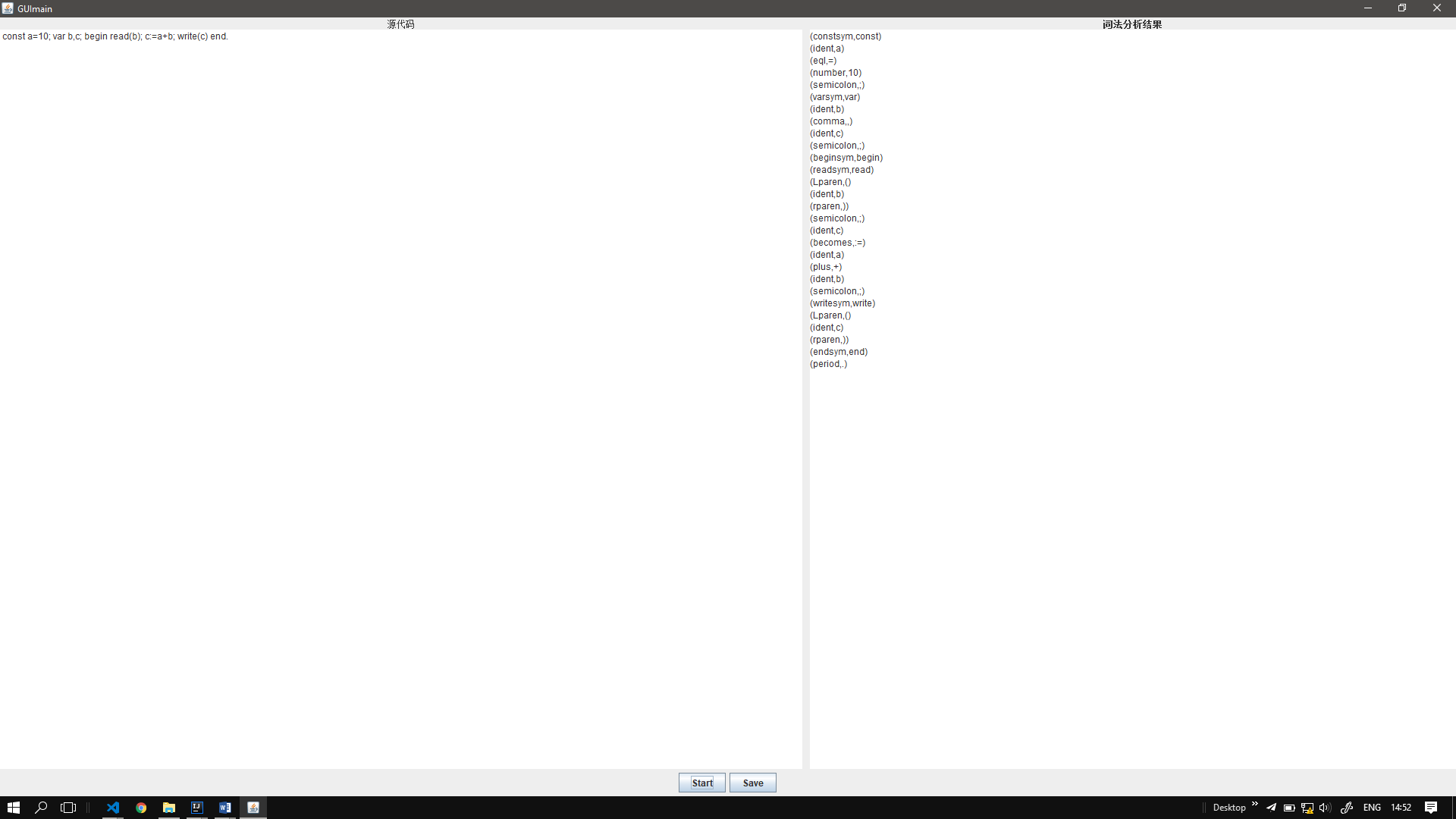
result+=file;

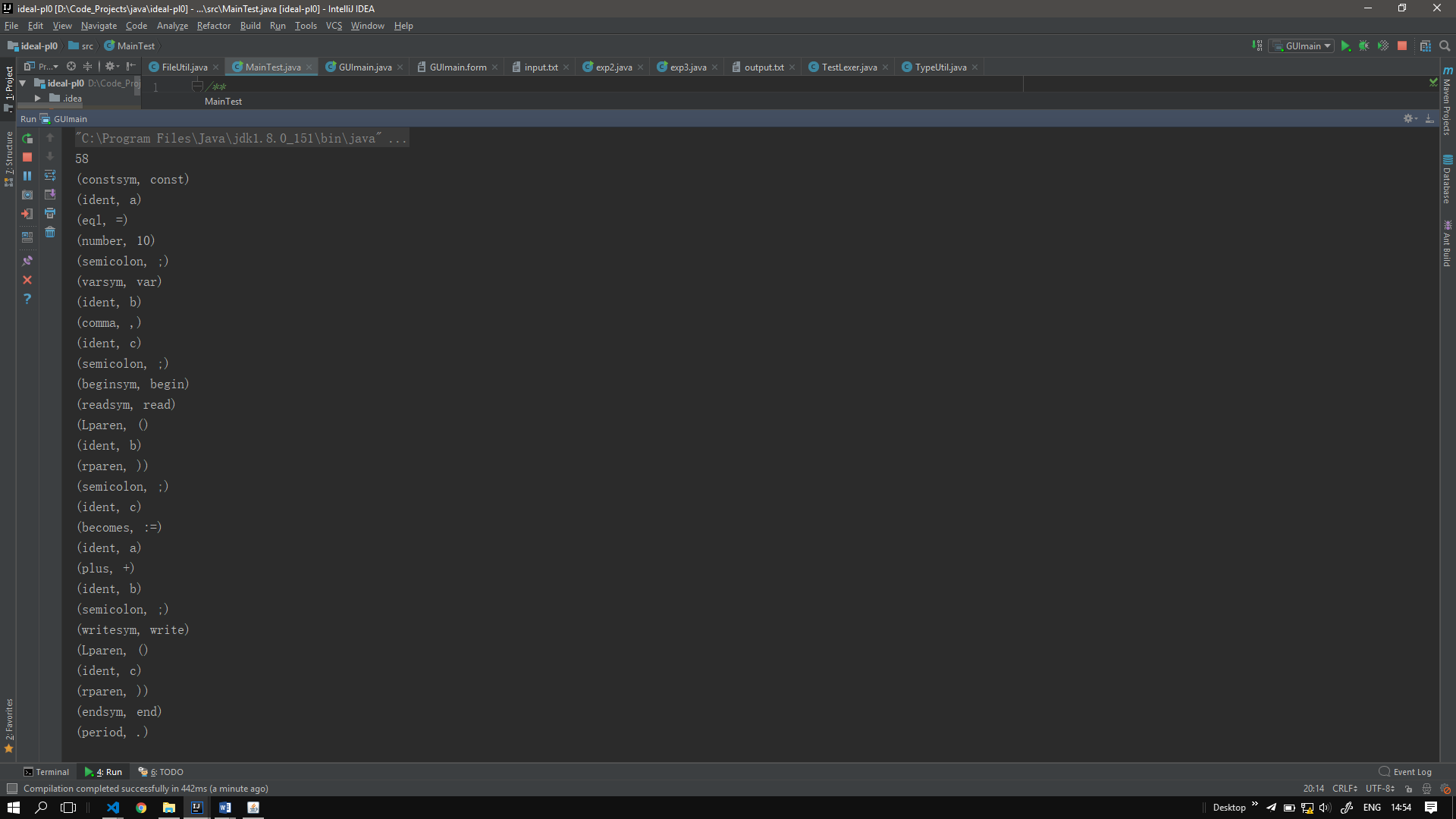
FileUtil.writeFile(file);

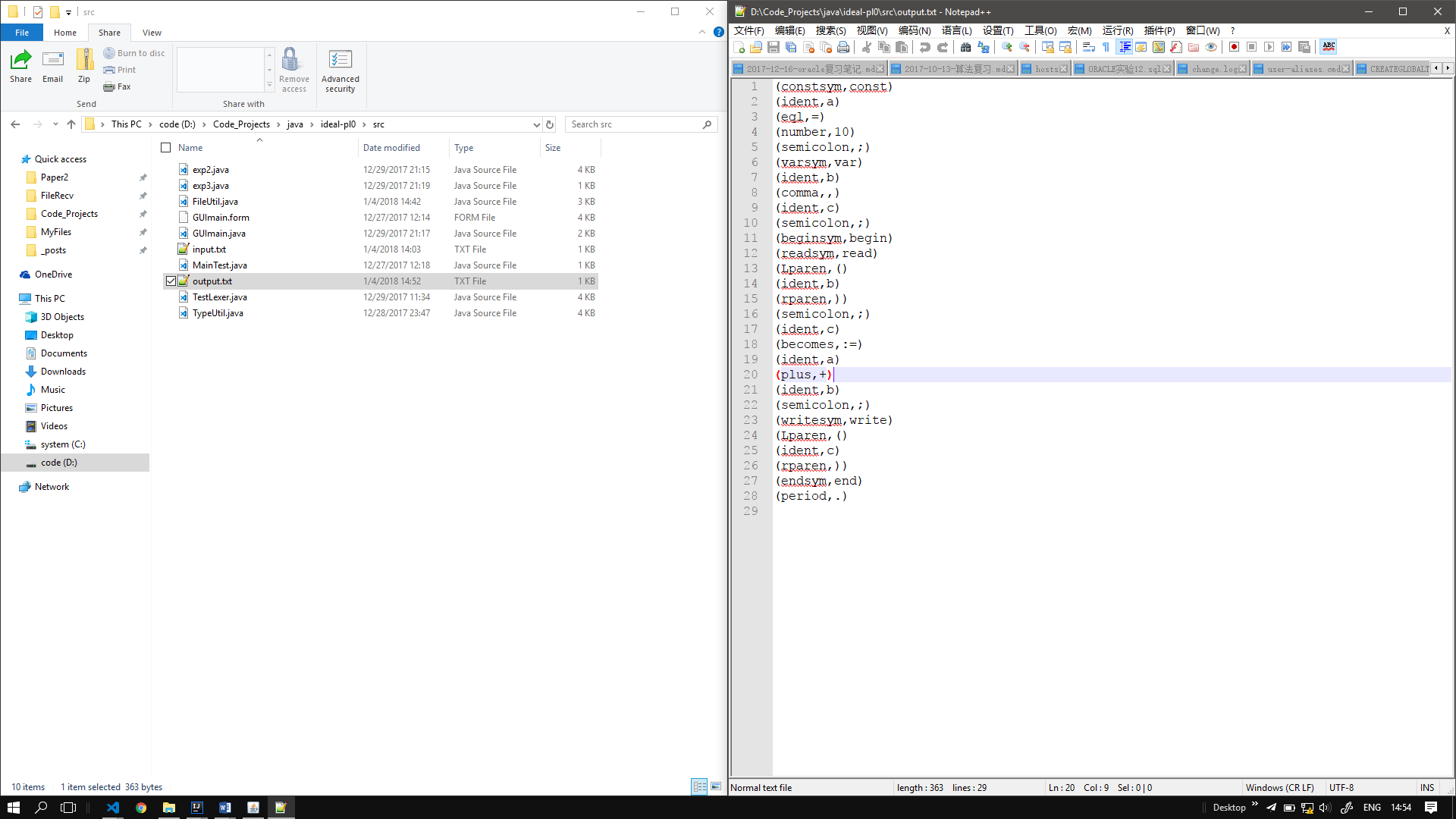
}

}

## 四，调试结果







## 五、实验总结与心得

　通过把该算法的内容，算法的执行顺序在计算机上实现，知道和理解了该理论在计算机中是怎样执行的，对该理论在实践中的应用有深刻的理解。

实验二 语法分析

## 一、实验内容与要求

(1)实验目的

* 给出PL/0文法规范，要求编写PL/0语言的语法分析程序。
* 通过设计、编制、调试一个典型的语法分析程序，实现对词法分析程序所提供的单词序列进行语法检查和结构分析，进一步掌握常用的语法分析方法。
* 选择最有代表性的语法分析方法，如递归子程序法；选择对各种常见程序语言都具备的语法结构，如赋值语句，特别是表达式，作为分析对象。

(2)实验内容

* 已给PL/0语言文法，构造表达式部分的语法分析器。
* 分析对象〈算术表达式〉的BNF定义如下：

<表达式> ::= [+|-]<项>{<加法运算符> <项>}

<项> ::= <因子>{<乘法运算符> <因子>}

<因子> ::= <标识符>|<无符号整数>| ‘(’<表达式>‘)’

<加法运算符> ::= +|-

<乘法运算符> ::= \*|/

<关系运算符> ::= =|#|<|<=|>|>=

(3)实验要求

* 将实验一“词法分析”的输出结果，作为表达式语法分析器的输入，进行语法解析，对于语法正确的表达式，报告“语法正确”；　　　　对于语法错误的表达式，报告“语法错误”， 指出错误原因。
* 把语法分析器设计成一个独立一遍的过程。
* 语法分析器的编写方法采用递归子程序法。

## 二、实验分析与设计

思路: 定义数据结构list，每个List的项都有两个值a和b，即读取文件后将内容通过识别’,’的位置读取到List中。

public class a {

public int index;//list index

String sym;//当前单元值

ArrayList<d> dataList=new ArrayList<d>();//存储语法分析的源内容

boolean res;

/\*\*

\* MAIN

\* @param args

\* @throws IOException

\*/

public static void main(String [] args) throws IOException {

}

/\*\*

\* 初始化，读取词法分析结果，并转为List

\* @throws IOException

\*/

public void init() throws IOException

/\*\*

\* 语法分析

\*/

public void analyze()

/\*\*

\* 表达式

\*/

public void expression()

/\*\*

\* 项

\*/

public void term()

/\*\*

\* 因子

\*/

public void factor()

/\*\*

\* 语法分析错误

\*/

public void Rerror()

/\*\*

\* 语法分析正确

\*/

public void Rright()

/\*\*

\* 获取d的第二位，index++

\* @return

\*/

public void getsym()

/\*\*

\* 数据单元：(a,b)，转为list单元

\*/

class d{

public d(String a,String b){

this.a=a;

this.b=b;

}

String a;

String b;

/\*\*

\* 打印

\*/

public void print(){

System.out.println("("+a+","+b+")");

}

}

## 三、源代码

import java.io.\*;

import java.util.ArrayList;

public class a {

public int index;//list index

String sym;//当前单元值

ArrayList<d> dataList=new ArrayList<d>();//存储语法分析的源内容

boolean res;

/\*\*

\* MAIN

\* @param args

\* @throws IOException

\*/

public static void main(String [] args) throws IOException {

a ta=new a();

ta.init();

ta.analyze();

}

/\*\*

\* 初始化，读取词法分析结果，并转为List

\* @throws IOException

\*/

public void init() throws IOException {

BufferedReader br = new BufferedReader(new InputStreamReader(new FileInputStream("./src/input1.txt")));

String data = null;

while((data = br.readLine())!=null)

{

// System.out.println(data);

int i;//获取“，”的位置,截取

for(i=0;i<data.length();i++){

if(data.charAt(i)==',')

break;

}

dataList.add(new d(data.substring(1,i),data.substring(i+1,data.length()-1)));

}

for(int i=0;i<dataList.size();i++){

dataList.get(i).print();

}

}

/\*\*

\* 语法分析

\*/

public void analyze(){

res=false;

index=-1;

getsym();

expression();

if(res==true){

System.out.println("right");

}else{

System.out.println("error");

}

}

/\*\*

\* 表达式

\*/

public void expression(){

System.out.println("expression:--------------");

if (sym.equals("+") || sym.equals("-")) {

getsym();

term();

}else{

term();

}

while (sym.equals("+") || sym.equals("-")){

getsym();

term();

}

}

/\*\*

\* 项

\*/

public void term() {

System.out.println("term:--------------");

factor();

while(sym.equals("\*") || sym.equals("/")){

getsym();

factor();

}

}

/\*\*

\* 因子

\*/

public void factor(){

System.out.println("factor:--------------");

if(!dataList.get(index).a.equals("ident")) {

if (!dataList.get(index).a.equals("number")) {

if (sym.equals("(")) {

getsym();

expression();

if (sym.equals(")")) {

getsym();

} else

Rerror();

} else

Rerror();

} else

getsym();

}else

getsym();

}

/\*\*

\* 语法分析错误

\*/

public void Rerror(){

// System.out.println("Error");

res=false;

}

/\*\*

\* 语法分析正确

\*/

public void Rright(){

// System.out.println("Right");

res=true;

}

/\*\*

\* 获取d的第二位，index++

\* @return

\*/

public void getsym(){

index=index+1;

if(index==dataList.size()){

Rright();

}else{

sym=dataList.get(index).b;

System.out.println(sym);

}

}

}

/\*\*

\* 数据单元：(a,b)，转为list单元

\*/

class d{

public d(String a,String b){

this.a=a;

this.b=b;

}

String a;

String b;

/\*\*

\* 打印

\*/

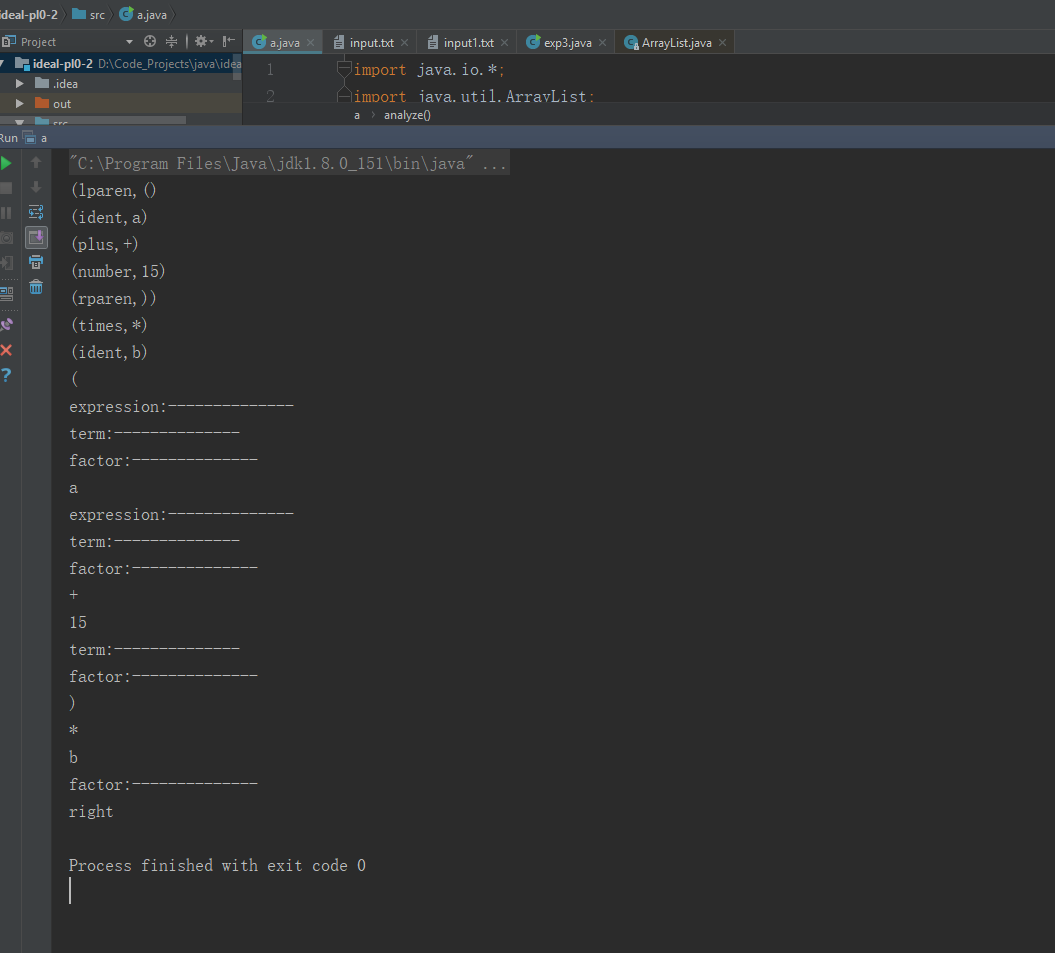
public void print(){

System.out.println("("+a+","+b+")");

}

}

## 四，调试结果



## 五、实验总结与心得

通过本次实验，掌握了什么是编译程序，编译程序工作的基本过程及其各阶段的基本任务，熟悉了编译程序总流程框图，了解了编译程序的生成过程、构造工具及其相关的技术对课本上的知识有了更深的理解，课本上的知识师机械的，表面的。通过把该算法的内容，算法的执行顺序在计算机上实现，把原来以为很深奥的书本知识变的更为简单，对实验原理有更深的理解。

实验三 词法 语法分析

## 一、实验内容与要求

(1)实验目的

给出PL/0文法规范，要求编写PL/0语言的语义分析程序。

(2)实验内容

已给PL/0语言文法，条件语句的翻译分析程序设计（输出四元式）

(3)实验要求

* 将实验一“词法分析器”与实验二“语法分析器”之间的衔接方式由独立一遍改为独立子程序。
* 语法分析器的编写方法采用递归子程序法。

## 二、实验分析与设计

输入：

PL/0表达式源语言，例如: （a+15）\*b作为输入。

输出：

对于语法正确的表达式，报告“语法正确”；

对于语法错误的表达式，报告“语法错误”， 指出错误原因。

## 三、源代码

import java.io.BufferedReader;

import java.io.BufferedWriter;

import java.io.File;

import java.io.FileNotFoundException;

import java.io.FileReader;

import java.io.FileWriter;

import java.io.IOException;

/\*\*

\* 文件操作

\* @author mfl

\*/

public class FileUtil {

/\*\*

\* 文件读取到缓冲区

\* @param buffer 缓冲区

\* @param fileSrc 文件路径

\* @return true : success

\*       false : filed

\*/

public static boolean readFile(StringBuffer buffer, String fileSrc) {

try {

FileReader fileReader = new FileReader(fileSrc);

BufferedReader br = new BufferedReader(fileReader);

String temp = null;

while ((temp = br.readLine()) != null) {

buffer.append(" "+temp);

}

return true;

} catch (FileNotFoundException e) {

e.printStackTrace();

} catch (IOException e) {

e.printStackTrace();

}

return false;

}

/\*\*

\* 追加方式写文件

\* @param args  需要写入字符串

\* @return  true : success

\*       false : filed

\*/

public static boolean writeFile(String args) {

try {

File file = new File("./src/output.txt");

if (!file.exists()) {

file.createNewFile();

}

FileWriter fw = new FileWriter(file.getAbsoluteFile(),true);

BufferedWriter bw = new BufferedWriter(fw);

bw.write(args);

bw.close();

return true;

} catch (IOException e) {

e.printStackTrace();

return true;

}

}

/\*\*

\* 清空文件

\*/

public static boolean clearFile() {

try {

File file = new File("./src/output.txt");

if (!file.exists()) {

file.createNewFile();

}

FileWriter fw = new FileWriter(file.getAbsoluteFile());

BufferedWriter bw = new BufferedWriter(fw);

bw.write("");

bw.close();

return true;

} catch (IOException e) {

e.printStackTrace();

return true;

}

}

}

import java.util.\*;

/\*\*

\* 对读取字符分类相关操作,父类

\* @author mfl

\*/

public class TypeUtil {

//dec new add by mfl:

private final String keyWords[] = {"begin","call","const","do","end","if","odd","procedure","read",

"then","var","while","write"}; // 关键字数组

public final static Map<String,String> map = new HashMap();

public final String keyWordsEncoding[]={"beginsym","callsym","constsym","dosym","endsym",

"ifsym","oddsym","proceduresym","readsym","thensym","varsym","whilesym","writesym"};

public final String operatorsEncoding[]={"plus","minus","times","slash","eql","neq","lss",

"leq","gtr","geq","becomes",":"};

public final String separatorsEncoding[]={"Lparen","rparen","comma","semicolon","period"};

private final String operators[] = {"+","-","\*","/","=","<>","<","<=",">",">=",":=",":"};

public final String separtors[]={"(", ")", ",", ";", "."};

/\*\*

\* 构造函数,init

\*/

public TypeUtil(){

initialize();

}

/\*\*

\* 初始化map，建立映射编码

\*/

public void initialize(){

map.clear();

for(int i=0;i<keyWords.length;i++){

map.put(keyWords[i], keyWordsEncoding[i]);

}

for(int i=0;i<operators.length;i++){

map.put(operators[i], operatorsEncoding[i]);

}

for(int i=0;i<separtors.length;i++){

map.put(separtors[i], separatorsEncoding[i]);

}

}

/\*\*

\* 获取map value

\* @param str map key

\* @return map value

\*/

public String getValue(String str){

return map.get(str);

}

/\*\*

\* 判断是否为字母

\* @param ch 需判断的字符

\* @return boolean

\*/

public boolean isLetter(char ch) {

return Character.isLetter(ch);

}

/\*\*

\* 判断是否为数字

\* @param ch 需判断的字符

\* @return boolean

\*/

public boolean isDigit(char ch) {

return Character.isDigit(ch);

}

/\*\*

\* 判断是否为关键字

\* @param s 需判断的字符串

\* @return boolean

\*/

public boolean isKeyWord(String s) {

for (int i = 0; i < keyWords.length; i++) {

if (keyWords[i].equals(s))

return true;

}

return false;

}

/\*\*

\* 判断是否为运算符

\* @param ch 需判断的字符

\* @return boolean

\*/

public boolean isOperator(char ch) {

for (int i = 0; i < operators.length; i++) {

if (Character.toString(ch).equals(operators[i]))

return true;

}

return false;

}

/\*\*

\* 判断字符串是否为操作符

\* @param ch 字符串

\* @return boolean

\*/

public boolean isOperator1(String ch) {

for (int i = 0; i < operators.length; i++) {

if (ch.equals(operators[i]))

return true;

}

return false;

}

/\*\*

\* 判断是否为分隔符

\* @param ch 需判断的字符

\* @return boolean

\*/

public boolean isSeparators(char ch) {

for (int i = 0; i < separtors.length; i++) {

if (Character.toString(ch).equals(separtors[i]))

return true;

}

return false;

}

}

/\*\*

\* 词法分析

\* 关键字，运算符一符一类

\* 标识符，常数，分隔符各自一类

\* 运算符未处理组合运算 ++、--、+= 等

\* @author mfl

\* @version 1.1

\*/

public class TestLexer extends TypeUtil{

public static StringBuffer buffer = new StringBuffer(); // 缓冲区

private int i = 0;

private char ch; // 字符变量，存放最新读进的源程序字符

private String strToken; // 字符数组，存放构成单词符号的字符串

public static String result;

/\*\*

\* 默认构造函数，无调用

\*/

public TestLexer() {

}

/\*\*

\* 读取指定路径文件

\* @param fileSrc 读取文件路径

\*/

public TestLexer(String fileSrc) {

FileUtil.readFile(buffer, fileSrc);

result="";

}

/\*\*

\* 词法分析

\*/

public void analyse() {

System.out.println(buffer.length());

strToken = ""; // 置strToken为空串

FileUtil.clearFile();//清空文件

while (i <buffer.length()) {

getChar();

// System.out.println("!!!"+ch);

getBC();

if (isLetter(ch)) { // 如果ch为字母

while (isLetter(ch) || isDigit(ch)){

concat();

getChar();

}

retract(); // 回调

if (isKeyWord(strToken)) {

writeFile(getValue(strToken),strToken);//strToken为关键字

}

else {

writeFile("ident",strToken);//strToken为标识符

}

strToken = "";

} else if (isDigit(ch)) {

while (isDigit(ch)) {//ch为数字

concat();

getChar();

}

if(!isLetter(ch)){//不能数字+字母

retract(); // 回调

writeFile("number",strToken); // 是整形

}else writeFile("error",strToken); // 非法

strToken = "";

} else if (isOperator(ch)) { //运算符

while(isOperator(ch)){

concat();

getChar();

}

retract();

if(isOperator1(strToken)){

writeFile(getValue(strToken),strToken);

}else{

writeFile("error", strToken);

}

strToken="";

}else if(isSeparators(ch)){

concat();

writeFile(getValue(strToken),strToken);

strToken="";

}else writeFile("error",ch+"");

}

}

/\*\*

\* 将下一个输入字符读到ch中，搜索指示器前移一个字符

\*/

public void getChar() {

ch = buffer.charAt(i);

i++;

}

/\*\* 检查ch中的字符是否为空白，若是则调用getChar()直至ch中进入一个非空白字符\*/

public void getBC() {

//isSpaceChar(char ch) 确定指定字符是否为 Unicode 空白字符。

//上述方法不能识别换行符

while (Character.isWhitespace(ch))//确定指定字符依据 Java 标准是否为空白字符。

getChar();

}

/\*\*将ch连接到strToken之后\*/

public void concat() {

strToken += ch;

}

/\*\* 将搜索指示器回调一个字符位置，将ch值为空白字 \*/

public void retract() {

i--;

ch = ' ';

}

/\*\*

\* 按照二元式规则写入文件

\* @param file 字符类型

\* @param s 当前字符

\*/

public void writeFile(String file,String s) {

// System.out.println(i);

System.out.println("("+file+", "+s+")");

file="("+file+","+s+")"+"\r\n";

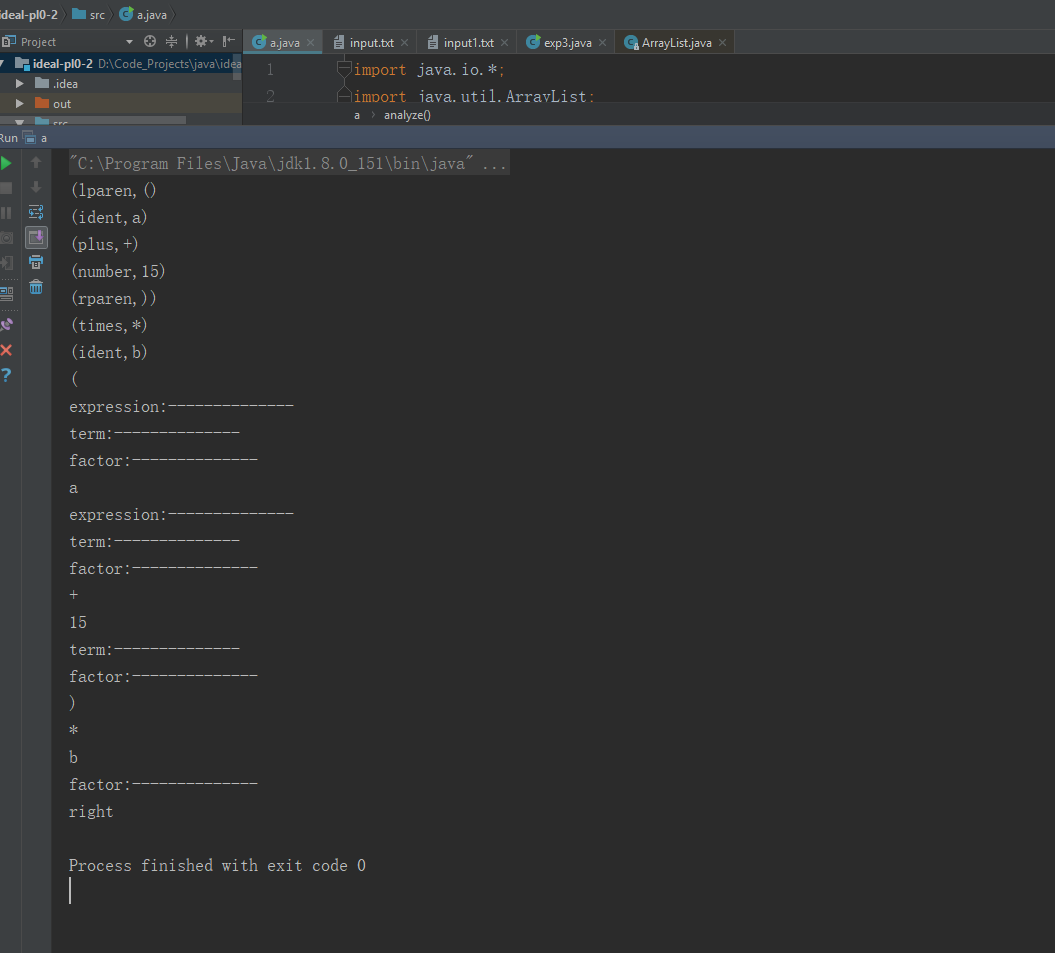
result+=file;

FileUtil.writeFile(file);

}

}

## 四，调试结果



## 五、实验总结与心得

通过本次实验，全面系统的理解了编译原理程序构造的一般原理和基本实现方法。把死板的课本知识变得生动有趣，激发了学习的积极性。把学过的计算机编译原理的知识强化，能够把课堂上学的知识通过自己设计的程序表示出来，加深了对理论知识的理解。以前对与计算机操

　　作系统的认识是模糊的，概念上的，现在通过自己动手做实验，从实践上认识了操作系统是如何处理命令的，如何协调计算机内部各个部件运行，对计算机编译原理的认识更加深刻。课程设计中程序比较复杂，在调试时应该仔细，在程序调试时，注意指针，将不必要的命令去除。