实验报告

实验一：词法分析

1.1 实验目的

1. 设计并实现一个Cminus语言的词法分析器。

2. 能够识别Cminus语言中的各种关键字、运算符、标识符等。

3. 为后续的语法分析和语义分析打下基础。

1.2 实验环境

1. 硬件环境：鲲鹏开发板。

2. 软件环境：openEuler操作系统。

3. 测试环境：提供一组Cminus语言的测试代码。

1.3 实验内容

1. 设计并实现一个词法分析器，能够识别Cminus语言中的各种元素。

2. 编写代码以枚举类型定义Cminus语言的词法单元。

3. 实现词法分析器的核心功能，包括读取源代码、识别词法单元、生成Token流。

1.4 实验过程

1.4.1 定义词法单元

定义一个枚举类型`Token`，包含Cminus语言的所有词法单元，如下所示：

|  |
| --- |
| typedef enum cminus\_token\_type {  // 运算符  ADD = 259, // +  SUB = 260, // -  MUL = 261, // \*  DIV = 262, // /  LT = 263, // <  LTE = 264, // <=  GT = 265, // >  GTE = 266, // >=  EQ = 267, // ==  NEQ = 268, // !=  ASSIN = 269, // =  // 符号  SEMICOLON = 270, // ;  COMMA = 271, // ,  LPARENTHESE = 272, // (  RPARENTHESE = 273, // )  LBRACKET = 274, // [  RBRACKET = 275, // ]  LBRACE = 276, // {  RBRACE = 277, // }  // 关键字  ELSE = 278, // else  IF = 279, // if  INT = 280, // int  FLOAT = 281, // float  RETURN = 282, // return  VOID = 283, // void  WHILE = 284, // while  // 标识符和数值  IDENTIFIER = 285, // 标识符，如变量名、函数名等  INTEGER = 286, // 整数值，如1、2、3等  FLOATPOINT = 287, // 浮点数值，如1.1、2.1等  ARRAY = 288, // 数组声明，如int data[10]等  LETTER = 289, // 字母字符，如'a'等  // 其他  EOL = 290, // 行终止符，如'\n'等  COMMENT = 291, // 注释，如// ...等  BLANK = 292, // 空白字符，如空格、制表符等  ERROR = 258 // 错误或无法识别的标记  } Token; |

1.4.2 实现词法分析器

1.4.2.1 读取源代码

实现一个函数，该函数能够逐字符读取源代码，并根据字符的性质进行分类处理。

1.4.2.2 设计token的正则表达式、识别词法单元

根据读取到的字符序列，识别出对应的词法单元，并生成相应的Token。

**运算**

|  |
| --- |
| "+" {pos\_start=pos\_end;pos\_end=pos\_start+yyleng;**return** ADD;}  "-" {pos\_start=pos\_end;pos\_end=pos\_start+yyleng;**return** SUB;}  "\*" {pos\_start=pos\_end;pos\_end=pos\_start+yyleng;**return** MUL;}  "/" {pos\_start=pos\_end;pos\_end=pos\_start+yyleng;**return** DIV;}  "<" {pos\_start=pos\_end;pos\_end=pos\_start+yyleng;**return** LT;}  "<=" {pos\_start=pos\_end;pos\_end=pos\_start+yyleng;**return** LTE;}  ">" {pos\_start=pos\_end;pos\_end=pos\_start+yyleng;**return** GT;}  ">=" {pos\_start=pos\_end;pos\_end=pos\_start+yyleng;**return** GTE;}  "==" {pos\_start=pos\_end;pos\_end=pos\_start+yyleng;**return** EQ;}  "!=" {pos\_start=pos\_end;pos\_end=pos\_start+yyleng;**return** NEQ;}  "=" {pos\_start=pos\_end;pos\_end=pos\_start+yyleng;**return** ASSIN;} |

符号

|  |
| --- |
| ";" {pos\_start=pos\_end;pos\_end=pos\_start+yyleng;**return** SEMICOLON;}  "," {pos\_start=pos\_end;pos\_end=pos\_start+yyleng;**return** COMMA;}  "(" {pos\_start=pos\_end;pos\_end=pos\_start+yyleng;**return** LPARENTHESE;}  ")" {pos\_start=pos\_end;pos\_end=pos\_start+yyleng;**return** RPARENTHESE;}  "[" {pos\_start=pos\_end;pos\_end=pos\_start+yyleng;**return** LBRACKET;}  "]" {pos\_start=pos\_end;pos\_end=pos\_start+yyleng;**return** RBRACKET;}  "{" {pos\_start=pos\_end;pos\_end=pos\_start+yyleng;**return** LBRACE;}  "}" {pos\_start=pos\_end;pos\_end=pos\_start+yyleng;**return** RBRACE;} |

**关键字**

|  |
| --- |
| "else" {pos\_start=pos\_end;pos\_end=pos\_start+yyleng;**return** ELSE;}  "if" {pos\_start=pos\_end;pos\_end=pos\_start+yyleng;**return** IF;}  "int" {pos\_start=pos\_end;pos\_end=pos\_start+yyleng;**return** INT;}  "float" {pos\_start=pos\_end;pos\_end=pos\_start+yyleng;**return** FLOAT;}  "return" {pos\_start=pos\_end;pos\_end=pos\_start+yyleng;**return** RETURN;}  "void" {pos\_start=pos\_end;pos\_end=pos\_start+yyleng;**return** VOID;}  "while" {pos\_start=pos\_end;pos\_end=pos\_start+yyleng;**return** WHILE;} |

**ID和num**

|  |
| --- |
| IDENTIFIER [a-zA-Z]+  INTEGER 0|[1-9][0-9]\*  FLOATPOINT [0-9]\*\.[0-9]\*  ARRAY \[\]  LETTER \'.\' //这里的letter，我理解为字符类型中的'a'这种的，所以这里这样表示，并且在样例测试中没有出错 |

|  |
| --- |
| {IDENTIFIER} {pos\_start=pos\_end;pos\_end=pos\_start+yyleng;return IDENTIFIER;}  {INTEGER} {pos\_start=pos\_end;pos\_end=pos\_start+yyleng;return INTEGER;}  {FLOATPOINT} {pos\_start=pos\_end;pos\_end=pos\_start+yyleng;return FLOATPOINT;}  {ARRAY} {pos\_start=pos\_end;pos\_end=pos\_start+yyleng;return ARRAY;}  {LETTER} {pos\_start=pos\_end;pos\_end=pos\_start+yyleng;return LETTER;} |

**其他不需要输出**

这些内容的动作可以在analyzer函数中补充

* 这里的注释 COMMENT正则表达式比较难写，在上方难点解释中说明了，另外在analyzer进行驱动时，需要考虑 /n,进行合理地换行
* 针对空格来说，还应考虑tab空格的问题，所以考虑了’\t’和’ ‘

|  |
| --- |
| LETTER \'.\'  COMMENT ("//"[^\n]\*)|("/\*"([^\*]|\\*+[^\*/])\*\\*+"/")  BLANK ([ ]|[\t])+  "\n" {return EOL;}  {COMMENT} {return COMMENT;}  {BLANK} {pos\_start=pos\_end;pos\_end+=yyleng;return BLANK;}  . {return ERROR;}  case COMMENT:  for(int i=0;i<yyleng;i++)  if(yytext[i]=='\n'){  pos\_end=1;  pos\_start=1;  lines++;  }else pos\_end++;  case BLANK:  break;  case EOL:  pos\_start=1;  pos\_end=1;  lines++;  break;  case ERROR:  printf("[ERR]: unable to analysize %s at %d line, from %d to %d\n", yytext, lines, pos\_start, pos\_end) |

1.4.2.3 生成Token流

将识别出的词法单元封装成Token，并输出一个Token流，供后续的语法分析使用。

1.5 实验结果验证

1.5.1 编译

这是make的结果，成功



1.5.2验证词法分析器的正确性

接下来执行py文件

|  |
| --- |
| python3 ./tests/lab1/test\_lexer.py |



执行成功

接着比较输出与答案所给的输出是否一致

alt text 输出与答案一致，测试通过。

1.5.3 验证词法分析器的健壮性

通过提供包含错误输入和边界情况的测试用例，验证词法分析器的健壮性。

以下是部分自定义的测试样例及运行结果：

**输入代码1**

|  |
| --- |
| */\*\*\*\*\*\*NA\*\*\*/*  **int** a;  */\*\*//\*\*\*\*/*  **int** world=1;  输出结果  int 280 2 1 4  a 285 2 5 6  ; 270 2 6 7  int 280 4 1 4  world 285 4 5 10  = 269 4 10 11  1 286 4 11 12  ; 270 4 12 13 |

经过比照，行号和列号均正确，token类型也正确。

1.6 实验反馈

1.熟悉了FLEX文件的编写，并学会了知晓了其中的一些全局变量,如yytext、yylex、yyleng等，如何利用FLEX生成一个生成词法分析器，完成了实验

2.练习了多个token的正则表达式的书写，对正则表达式更加熟悉，特别是注释，letter等重点内容

3.针对输入中可能出现的非法字符或格式，有健壮的错误处理机制。 这里直接将不匹配的格式作为error进行输出，并在词法分析器中输出对应的内容，行号，起始位置和结束位置

4.实验说明中可以加入flex相关内容的基本说明，在代码中会使用到yyleng和yytext，如全局变量 yytext、yylex()、yyleng的定义

1.7 附录

- 枚举类型定义：定义了Cminus语言的所有词法单元，用于词法分析器的实现。

- 测试用例：提供了一组测试用例，用于验证词法分析器的正确性和健壮性。

姓名：邹林壮 学号：202208040412