# 编译器设计实验报告

**一、实验目的**

本实验旨在通过编写flex词法分析器，掌握词法分析的基本原理和flex工具的使用方法，能够正确识别和处理源代码中的关键字、标识符、操作符、注释等词法单元。

**二、实验环境**

* 操作系统： Windows（支持flex工具）
* 开发工具：flex、文本编辑器、vscode

**三、实验内容**

1. 编写flex的.l文件，定义源代码中的词法单元及其正则表达式规则。
2. 使用flex工具生成词法分析器的C代码。
3. 编译并运行生成的词法分析器，测试其对源代码的词法分析功能。观察out.txt文件的输出内容。

**四、实验步骤**

1. **编写.l文件**  
   根据实验要求，我编写了如下.l文件内容，用于识别C语言中的关键字、标识符、操作符、注释等词法单元。

**Lexer.l代码**：

%top{

#include <stdio.h>

#include <string.h>

}

white [ \t\n]

digit [0-9]

letter [A-Za-z]

id ({letter}|\_)(({letter}|{digit}|\_)\*)

number [1-9]{digit}\*|0

commentbegin "/\*"

commentelement .|\n

commentend "\*/"

%x COMMENT

%%

{commentbegin} {BEGIN COMMENT; fprintf(yyout,"注释开始:\n");}

<COMMENT>{commentelement} {fprintf(yyout,"%s",yytext);}

<COMMENT>{commentend} {BEGIN INITIAL; fprintf(yyout,"\n注释结束\n");}

"\"" {fprintf(yyout,"%-10s %s\n","QUOTE",yytext);}

{white}+ {}

{number} {fprintf(yyout,"%-10s %s\n","NUMBER",yytext);}

"void" {fprintf(yyout,"%-10s %s\n","VOID",yytext);}

"char" {fprintf(yyout,"%-10s %s\n","CHAR",yytext);}

"int" {fprintf(yyout,"%-10s %s\n","INT",yytext);}

"sizeof" {fprintf(yyout,"%-10s %s\n","SIZEOF",yytext);}

"const" {fprintf(yyout,"%-10s %s\n","CONST",yytext);}

"return" {fprintf(yyout,"%-10s %s\n","RETURN",yytext);}

"continue" {fprintf(yyout,"%-10s %s\n","CONTINUE",yytext);}

"break" {fprintf(yyout,"%-10s %s\n","BREAK",yytext);}

"if" {fprintf(yyout,"%-10s %s\n","IF",yytext);}

"else" {fprintf(yyout,"%-10s %s\n","ELSE",yytext);}

"switch" {fprintf(yyout,"%-10s %s\n","SWITCH",yytext);}

"case" {fprintf(yyout,"%-10s %s\n","CASE",yytext);}

"default" {fprintf(yyout,"%-10s %s\n","DEFAULT",yytext);}

"for" {fprintf(yyout,"%-10s %s\n","FOR",yytext);}

"do" {fprintf(yyout,"%-10s %s\n","DO",yytext);}

"while" {fprintf(yyout,"%-10s %s\n","WHILE",yytext);}

"scanf" {fprintf(yyout,"%-10s %s\n","SCANF",yytext);}

"printf" {fprintf(yyout,"%-10s %s\n","PRINTF",yytext);}

{id} {fprintf(yyout,"%-10s %s\n","ID",yytext);}

"{" {fprintf(yyout,"%-10s %s\n","{",yytext);}

"}" {fprintf(yyout,"%-10s %s\n","}",yytext);}

"[" {fprintf(yyout,"%-10s %s\n","LB",yytext);}

"]" {fprintf(yyout,"%-10s %s\n","RB",yytext);}

"(" {fprintf(yyout,"%-10s %s\n","LP",yytext);}

")" {fprintf(yyout,"%-10s %s\n","RP",yytext);}

"~" {fprintf(yyout,"%-10s %s\n","LOGRE",yytext);}

"++" {fprintf(yyout,"%-10s %s\n","INPLUS",yytext);}

"--" {fprintf(yyout,"%-10s %s\n","INMINUS",yytext);}

"!" {fprintf(yyout,"%-10s %s\n","LOCRE",yytext);}

"\*" {fprintf(yyout,"%-10s %s\n","STAR",yytext);}

"/" {fprintf(yyout,"%-10s %s\n","DIVOP",yytext);}

"%" {fprintf(yyout,"%-10s %s\n","COMOP",yytext);}

"+" {fprintf(yyout,"%-10s %s\n","PLUS",yytext);}

"-" {fprintf(yyout,"%-10s %s\n","MINUS",yytext);}

">" {fprintf(yyout,"%-10s %s\n","RELG",yytext);}

"<" {fprintf(yyout,"%-10s %s\n","RELL",yytext);}

">=" {fprintf(yyout,"%-10s %s\n","RELGEQ",yytext);}

"<=" {fprintf(yyout,"%-10s %s\n","RELLEQ",yytext);}

"==" {fprintf(yyout,"%-10s %s\n","EQUOP",yytext);}

"!=" {fprintf(yyout,"%-10s %s\n","UEQUOP",yytext);}

"&&" {fprintf(yyout,"%-10s %s\n","ANDAND",yytext);}

"||" {fprintf(yyout,"%-10s %s\n","OROR",yytext);}

"=" {fprintf(yyout,"%-10s %s\n","EQUAL",yytext);}

"/=" {fprintf(yyout,"%-10s %s\n","ASSIGNDIV",yytext);}

"\*=" {fprintf(yyout,"%-10s %s\n","ASSIGNSTAR",yytext);}

"%=" {fprintf(yyout,"%-10s %s\n","ASSIGNCOM",yytext);}

"+=" {fprintf(yyout,"%-10s %s\n","ASSIGNPLUS",yytext);}

"-=" {fprintf(yyout,"%-10s %s\n","ASSIGNMINUS",yytext);}

"," {fprintf(yyout,"%-10s %s\n","COMMA",yytext);}

"#" {fprintf(yyout,"%-10s %s\n","SHA",yytext);}

";" {fprintf(yyout,"%-10s %s\n","SEMI",yytext);}

":" {fprintf(yyout,"%-10s %s\n","COLON",yytext);}

%%

int main() {

yyin=fopen("in.c","r");

yyout=fopen("out.txt","w");

fprintf(yyout,"After Lexical Analysis\n");

yylex();

return 0;

}

int yywrap() {

return 1;

}

2、重点代码分析：

### 定义部分

#### 顶部代码块

|  |  |
| --- | --- |
|  | %top{ |
|  | #include<stdio.h> |
|  | #include<string.h> |
|  | } |

* %top：这是一个flex的特殊指令，用于包含C代码在词法分析器的顶部。
* #include<stdio.h> 和 #include<string.h>：这两行代码包含了标准输入输出和字符串处理的头文件，以便在词法分析器的C代码中使用相应的函数。

### 正则表达式定义

#### 空白字符

|  |  |
| --- | --- |
|  | white [ \t\n] |

* white：定义了一个名为white的正则表达式，用于匹配空格、制表符和换行符。

#### 数字和字母

|  |  |
| --- | --- |
|  | digit [0-9] |
|  | letter [a-zA-Z] |

* digit：定义了一个匹配任意数字的正则表达式。
* letter：定义了一个匹配任意字母的正则表达式，包括大写和小写。

#### 标识符和数字

|  |  |
| --- | --- |
|  | id ({letter}|\_)(({letter}|{digit}|\_)\*) |
|  | number [1-9]{digit}\*|0 |

* id：定义了标识符的正则表达式，标识符可以以字母或下划线开头，后面跟着任意数量的字母、数字或下划线。
* number：定义了数字的正则表达式，包括整数和零。

### 注释处理

#### 单行注释

|  |  |
| --- | --- |
|  | COMMEN1BEGIN "//" |
|  | COMMEN1END "\n" |
|  | COMMEN1ELE . |

* COMMEN1BEGIN：定义了单行注释的开始标识，即//。
* COMMEN1END：定义了单行注释的结束标识，即换行符\n。
* COMMEN1ELE：定义了单行注释中可以出现的任意字符。

#### 多行注释

|  |  |
| --- | --- |
|  | COMMEN2BEGIN "/\*" |
|  | COMMEN2ELE .|\n |
|  | COMMEN2END "\*/" |

* COMMEN2BEGIN：定义了多行注释的开始标识，即/\*。
* COMMEN2ELE：定义了多行注释中可以出现的任意字符或换行符。
* COMMEN2END：定义了多行注释的结束标识，即\*/。

### 词法分析规则

#### 注释处理规则

|  |  |
| --- | --- |
|  | {COMMEN1BEGIN} {BEGIN COMMENT1;fprintf(yyout,"单行注释开始：\n");} |
|  | <COMMENT1>{COMMEN1END} {BEGIN INITIAL;fprintf(yyout,"\n单行注释结束\n");} |
|  | <COMMENT1>{COMMEN1ELE} {fprintf(yyout,"%s",yytext);} |
|  |  |
|  | {COMMEN2BEGIN} {BEGIN COMMENT2;fprintf(yyout,"通用注释开始：\n");} |
|  | <COMMENT2>{COMMEN2ELE} {fprintf(yyout,"%s",yytext);} |
|  | <COMMENT2>{COMMEN2END} {BEGIN INITIAL;fprintf(yyout,"\n通用注释结束\n");} |

* 这些规则定义了当词法分析器遇到注释时应该如何处理。当遇到单行注释的开始标识时，进入COMMENT1状态并输出提示信息；当遇到单行注释的结束标识时，返回INITIAL状态并输出提示信息；在COMMENT1状态下，输出单行注释中的每一个字符。多行注释的处理方式类似。

**词法单元处理规则**：

* + {white}+：遇到连续的空白字符时，不执行任何动作。
  + 字符串字面量（由双引号包围的字符序列）被识别并打印出来。
  + 关键字（如main、const、int等）被识别并打印其类型和值。
  + 标识符（id）被识别并打印其类型和值。
  + 数字（number）被识别并打印其类型和值。
  + 一些操作符（如!、&&、||、+）也被识别并打印。

**输出格式**：

* + 每种词法单元都被打印为“类型 - 值”的形式，其中类型是一个描述性的字符串（如ID、NUMBER、CONSTTK等），值则是词法单元在源代码中的文本表示。

### main 函数

1. **打开输入文件**：

**c**

|  |  |
| --- | --- |
|  | yyin = fopen("in.c", "r"); |

这行代码打开名为"in.c"的文件，并将其文件指针赋值给yyin。yyin是Flex的词法分析器使用的全局变量，它指向词法分析器应该从中读取输入的文件或流。

1. **打开输出文件**：

**c**

|  |  |
| --- | --- |
|  | yyout = fopen("out.txt", "w"); |

这行代码打开名为"out.txt"的文件，用于写入，并将其文件指针赋值给yyout。yyout同样是Flex的词法分析器使用的全局变量，用于指定词法分析器应该将识别的词法单元输出到哪里。

1. **调用词法分析器**：

**c**

|  |  |
| --- | --- |
|  | yylex(); |

调用yylex()函数是词法分析的核心步骤。这个函数由Flex生成，负责从yyin读取输入，识别词法单元，并通过yyout输出或进行其他处理。

### yywrap 函数

**c**复制代码

|  |  |
| --- | --- |
|  | int yywrap() { |
|  | return 1; |
|  | } |

yywrap函数是Flex词法分析器的一部分，当词法分析器到达输入文件的末尾时调用。默认情况下，Flex的词法分析器在遇到文件末尾时会继续尝试读取输入，这通常不是期望的行为。通过实现yywrap函数并返回非零值（在这里是1），可以告诉词法分析器在到达文件末尾时停止。

In.c文件：

*const* *int* array[2] **=** {1,2};

*int* main(){

*int* c;

    c **=** getint();*//this is one line*

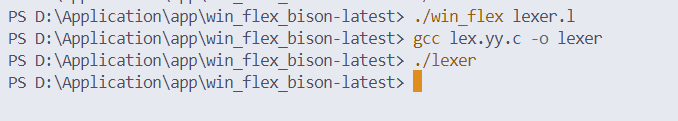
    printf("output is *%d*",c);*/\*this*

*is  two line\*/*

    "this is a string with\""

**return** c;

}



输出结果：



CONSTTK const

 INTTK int

 ID array

LBRACK [

NUM 2

RBRACK ]

 ASSIGN =

 LBRACE {

NUM 1

COMMA ,

NUM 2

RBRACE }

SEMICN ;

INTTK int

 MAINTK main

LPARENT (

RPARENT )

LBRACE {

    INTTK int

 ID c

SEMICN ;

  SSIGN   ID c

 A =

 GETINTTK getint

LPARENT (

RPARENT )

SEMICN ;

单行注释开始：

this is one line

单行注释结束

    PRINTFTK printf

LPARENT (

STRING "output is %d"

COMMA ,

ID c

RPARENT )

SEMICN ;

通用注释开始：

this

is two line

通用注释结束

STRING "this is a string with\""

    RETURNTK return

 ID c

SEMICN ;

RBRACE }