Chittagong University of Engineering and Technology

Department of Computer Science and Engineering

Title: Write a Program using flex to identify C tokens

name: Nishan Paul
ID: 1604085
course: CSE-432
date: 02/10/2021

## Project Title:

Write a program using flex that takes input and divide the lexemes of that input into following tokens:

1. keywords: if, then, int, ....
2. relational operators: >, <, ==, ...
3. arithmatic operators: +, −, *, ....
4. assignment operators: =, +=, *=, ....
5. logical operators: &&, ||, ....
6. valid numbers: 0, 1, 2, 3, 30, ....
7. valid identifiers: length, Len123, ....
8. function name: main(), random(), ...
9. other symbol: {, }, [, ], ....
10. string: "hello", "world", ....

## Objectives:

1. To identify C tokens
2. To learn flex for identifying tokens

Introduction: Flex (fast lexical analyzer generator) is a tool or computer program for generating lexical analyzer. Flex is more flexible than lex and produces faster code. The function yylex() is automatically generated by the flex when it is provided with a .l file and yylex() function is expected by the parser to call to retrieve tokens from current token stream. The function yylex() is the main felex function that runs the rule section and extension (.l) is the extension used to save the program.

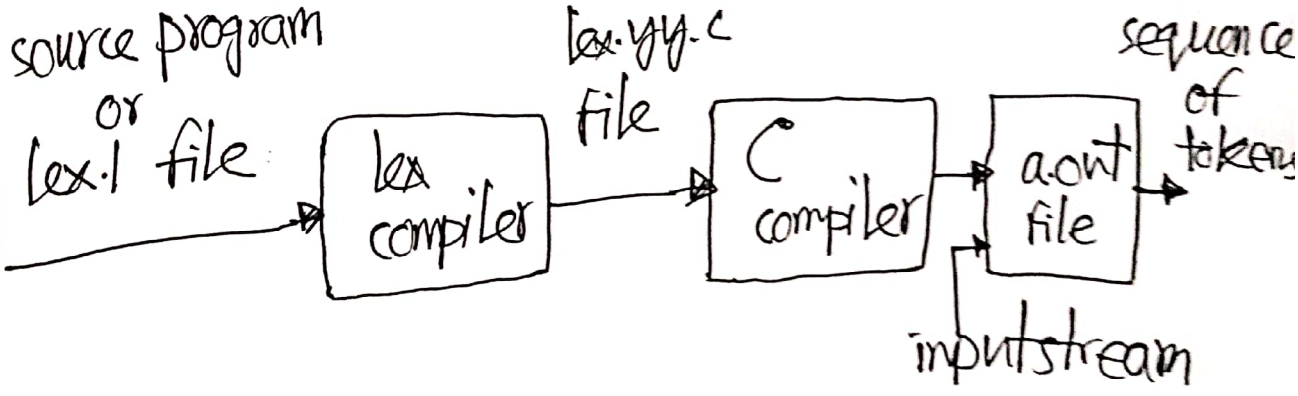


figure: Block diagram how flex works

The total steps of the work is done by three level. In the first step, we will give an input. Input file describes the lexical analyzer to be generated named lex is written in lex language. The lex compiler transforms lex.l to C program, in a file that is always named lex.yy.c.

In 2nd step, the compiler compile lex.yy.c file into an executable file called a.out. Finally, the output file a.out take a stream of input characters and produce a streams of token

## Structure of a lex file:

A lex file looks like,

```
..... definations ....
%%
..... rules .....
%%
..... code .....
```

1. Defination section: The defination section contains the declaration of variables, regular definations, Here text is enclosed in "%{ %}" brackets. Anything written in this brackets is copied directly to the file Lex.yy.c.

2. Rules section: The rules section contains a series of rules form of: pattern, action and pattern must be unintended and action begin on the same line in {} brackets. The rule section is enclosed in "%.%.%.%".

3. Code section: This section contains C statements and additional functions. We can compile these functions seperately and load with the lexical analyzer.

## Example of token declarations:

| Token | Expression | Meaning |
|---|---|---|
| number | ([0-9])+ | 1 or more occurence of digit |
| chars | [A-Z a-z] | any character |
| blank | " " | a blank space |
| word | (chars)+ | 1 or more occurence of chars |

## lex variable:

| | |
|---|---|
| yyin | of the type file*. This points to the current file being parsed by the lexer. |
| yyout | of the type file*. This points to the location where the output of the lexer will be written |
| yytext | The text of the matched pattern is stored in variable |

## lex function:

| | |
|---|---|
| yylex() | The function that starts the analysis |
| yywrap() | This function is called when end of file or input is encountered. At the end, yywrap() can return 1 to indicate end of parsing |

1604085_lex_file.l - Notepad

File Edit Format View Help

```
%{
#include <stdio.h>
%}
%%

"++"|"--"  {printf("%s unary operator\n",yytext);}

"+"|"-"|"*"|"/" {printf("%s arithmetic operator\n",yytext);}

["]([^"\\\n]|\\.|\\\n)*["] {printf("%s string\n",yytext);}

"="|"+="|"-="|"*="|"/="|"%="|"<<="|">>="|"&="|"^="|"|=" {printf("%s assignment operator\n",yytext);}

"=="|">="|"<="|"!="|">"|"<"  {printf("%s relational operator\n",yytext);}

"&&"|"||"|"!"  {printf("%s logical operator\n",yytext);}

if|else|then|int|switch|for|char|return|main|string|while|do|break|continue|double|float|EOF|case|long|short|sizeof|void|static|goto|struct|unsigned {printf("%s keyword\n",yytext);}

[a-zA-Z_]([a-zA-Z_]|[0-9])*[(][a-zA-Z0-9," "]*[)] {printf("%s function\n",yytext);}

"{"|"}"|"["|"]"|","|";"|"("|")" {printf("%s symbol\n",yytext);}


[a-zA-Z_][a-zA-Z_0-9]*    {printf("%s identifier\n",yytext);}

[0-9]+[.0-9]* {printf("%s valid number\n",yytext);}

"//".*" "*   {printf("%s single line comment\n",yytext);}

\/\*(.*\n)*.*\*\/ {printf("%s multi line comment\n",yytext);}

"#"|"@"|"$"|"^"|"%"|"^"|"&" {printf("%s special character\n",yytext);}

#.* {printf("%s header\n",yytext);}

[\t\n]+

%%
main()
{
yylex();
return 0;
}
```

Windows (CRLF) Ln 39, Col 1 100%

1604085_input.txt - Notepad

File Edit Format View Help

```
#include <stdio.h>
int main() {
    int number;

    printf("Enter an integer: ");
    scanf("%d", &number);

    // true if number is less than 0
    if (number < 0) {
        printf("You entered %d.\n", number);
    }

    printf("The if statement is easy.");

    return 0;
}
```

Windows (CRLF) Ln 1, Col 1 100%

1604085_output.txt - Notepad

File Edit Format View Help

```
This is Pre-processor directive: #include <stdio.h>

This is Keyword: int

This is Function: main(

This is Delimiter: )

This is Delimiter: {

This is Keyword: int

This is Identifier: number

This is Delimiter: ;

This is Function: printf(

this is String:"Enter an integer: "

This is Delimiter: )

This is Delimiter: ;

This is Function: scanf(

this is String:"%d"

This is Delimiter: ,

This is Special Characters: &

This is Identifier: number

This is Delimiter: )

This is Delimiter: ;

this is single line Commments: // true if number is less than 0

This is Keyword: if

This is Delimiter: (
```

Windows (CRLF) Ln 1, Col 1 100%

Discussion: We have identified the C tokens using flex. All the tokens were identified clearly. For running the flex we had to go to the command prompt. After all the successful command the results were generated in the output file.