Khulna University of Engineering & Technology,

Khulna

Dept. of Computer Science and Engineering

Report on:

Writing a program in flex to design the syntax of my programming language.

Course No.: CSE 3212

Course Title: Compiler Design Laboratory

Submission Date: 05-10-2023

Submitted by-

Farzana Rahman

Roll: 1907028

Section: A

Year: 3rd

Semester: 2nd

Objectives:

- To design my own programming language
- To create regular expressions for matching the input
- To check if a statement is valid or not
- To match single line and multiline comment
- To check if the flex program is giving the correct output or not
- To count the total number or variables, statements is given input

Introduction:

Flex is a tool used in compiler design. It helps us to generate tokens, match the syntax with the given input and tells us that if there is any error or not. It also helps to create own programming language.

Using this I have designed my own programming language. The components of this language are given below-

Datatypes:

fint - it will take both positive and negative integer

ffloat - it will take both positive and negative floating point number

fstring - it will take string type data

fbool - it will take only two values, true or false

Directives:

Here I have changed the syntax of the directives from c.

The syntax will be like:

#include<<flang or flang+.h>>

Main Function:

I have designed the main function of my language as:

fmain [ ]

$

--here will be the body of the program

$

Variables:

The language will take the variables as "_"{fletter}

It means that first character will always be "_"followed by any number of letters.

So, _a is a correct variable but _2 or abc these are wrong.

Keywords:

Keywords will include all the datatypes as well as fmain, fret, ffor, fwhile, fif, felsif, felse.

Comments:

Single-line comment: To identify a single line comment, I have used two '#' symbols.

Syntax will be like:

##any kind of comments

Multi-line comment: The multiline comment will be started with /* and ended with */. For example,

/*
abcd..
..xyz
*/

This will be treated as multiline comment.

Operators:

<=, <, >=, >, ==, != : These will be used as relational operators

+, -, *, / : These will be used as arithmetic operators = : assignment operation ++, -- : These two operators will be used in ffor and fwhile loop for increment and decrement the index.

Statements:

There will be two types of statements. One will be valid operation and the other will be valid declaration. Both of them will be ended with this '|'. This '|' will act as ';' in c.

Valid declaration: Here, variables will be declared correctly. I couldn't handle multiple variable declaration. I tried but it was not giving correct answer. That's why I have omitted that.

The syntax will be like:

Data_type variable_name |

Or, Data_type variable_name = some_values |

Valid operation: This is mainly for arithmetic operations.

The syntax will be like:

Varibable_name = variable_name arithmetic_operators variable_name|

Loops:

Here I have declared two loops. One is ffor which is similar to for loop in c and the other one is fwhile loop.

Suppose ‘a’ represents variable, ‘<=’ represents all the relational operators.

The syntax for ffor loop is:

```
 ffor @ a=0;a<=5;a++ @
(
--here will be some valid operations
)
```

In the condition section it can compare with variables also.

The syntax for fwhile loop is:

```
fwhile @ a<=10 @
(
--here will be some valid operations
```

a++ or a--|
)
In the condition section it can compare with variables also.

Conditional statement:

To define conditional statements, I have designed a fif-felsif-felse similar to if-elseif-else in c.

Suppose ‘a’ and ‘b’ are two variables. Then this loop will work as-

fif @ a<=b @
(
--some valid operations
)
felsif @ a>=b @
(
--some valid operations
)
felse
(
--some valid operations
)
In the condition section it can compare with digits also.

Discussion:

In this assignment, we are asked to design some components of our programming language. We are asked to design the variable's syntax, keywords, loops, operators, conditional statements etc. which are needed for any programming language. But to design these, I have faced some errors that's why I couldn't fulfil all the requirements. But this helps me to get the clear idea of the first stage of compiler design that is lexical analysis. We are also asked to count the number of variables, keywords, statements in the source program and detect the multiline and single line comments. I have done this in my assignment.

Lex-program:

```
%{
#include<stdio.h>
#include<string.h>
int cnt_keyword=0;
int cnt_variables=0;
int cnt_stmnts=0;
int f=0;
char str[10000];
char str2[10000];
char str3[10000];
char str4[10000];
char str5[10000];

%}

fdir
    [#("include")("<<")("flang"|"flang+")[.]("h")(">>")
[ \n][ \n\t]*
fdigit       [0-9]+
fletter      [a-zA-Z]+
stmnt        ("|")
fint     ([\+\-]?{fdigit})
ffloat       ([\+\-]?{fdigit}"."{fdigit})
fbool        ("true"|"false")
fstring      [\"].*[\"]

fvariable        ("_"{fletter})
ffmain       ("fmain"[ ]"["[ ]"]"[ \n\t]*"$"[ \n\t]*)
freturn      ([ \n\t]*"fret"[ ]{fdigit}[ ]?"|"[
\n\t]*"$")
fdt      ("fint"|"ffloat"|"fstring"|"fbool")
fkw
    ("fint"|"ffloat"|"fstring"|"fbool"|"fif"|"felsif"|"
felse"|"ffor"|"fwhile")

frel_op      (">"|"<"|">="|"<="|"=="|"!=")
farith_op        ("+"|"-"|"*"|"/")
finc_dec     ("++"|"--")
fassign_op       [=]
```

```
f1line_cmnt       ("##")[^\n]*
fmultiline_cmnt       "/*"[^*/]*"*/"


valid_declaration     {fdt}[ ]+{fvariable}[ ]*([=][
]*({fint}|{ffloat}|{fstring}|{fbool})[ ]*)?"|"[ \n\t]*

valid_operation
     ({fvariable}[=]{fvariable}{farith_op}{fvariable}"|"
[ \n\t]*)


ffor     ("ffor"[ ]"@"[ ]{fdt}[
]{fvariable}[=]{fdigit}";"{fvariable}{frel_op}({fdigit}
|{fvariable})";"{fvariable}{finc_dec}[ ]"@"[ \n\t]*[(][
\n\t]*{valid_operation}*[ \n\t]*[)][ \n\t]*)

fwhile       ("fwhile"[ ]"@"[
]{fvariable}{frel_op}({fvariable}|{fdigit})[ ]"@"[
\n\t]*[(][ \n\t]*{fvariable}{finc_dec}"|"[ \n\t]*[)][
\n\t]*)

fif      ("fif"[ ]"@"[ ]{fvariable}{frel_op}{fvariable}[
]"@"[ \n\t]*[(][ \n\t]*{valid_operation}*[ \n\t]*[)][
\n\t]*)

felsif       ("felsif"[ ]"@"[
]{fvariable}{frel_op}{fvariable}[ ]"@"[ \n\t]*[(][
\n\t]*{valid_operation}*[ \n\t]*[)][ \n\t]*)

felse        ("felse"[ \n\t]*[(][
\n\t]*{valid_operation}*[ \n\t]*[)][ \n\t]*)
%%
{fdir}       {printf("\nDirective is declared correctly
\n");}

{f1line_cmnt}         {printf("Single line comment
detected \n");}
```

```
{fmultiline_cmnt}    {printf("Multi line comment
detected \n");}
{fkw}        {
             ++cnt_keyword;
             }
{fvariable}      {
                     ++cnt_variables;
                     }

{ffmain}     {printf("In the main function \n");}

{valid_operation}    {
                         printf("This is a valid
operation \n");
                         ++cnt_stmnts;
                         }
{valid_declaration}      {
                         printf("This is a valid
declaration \n");
                         ++cnt_stmnts;
                         }

{ffor}       {printf("for loop detected \n");
             strcpy(str2,yytext);
             int i;
             for(i=0;i<strlen(str2);i++)
             {if(str2[i]=='|')
             {++cnt_stmnts;}
             if(str2[i]=='_')
             {++cnt_variables;}}}

{fwhile}     {printf("fwhile loop detected \n");
                     strcpy(str3,yytext);
                     int i;
                     for(i=0;i<strlen(str3);i++)
                     {if(str3[i]=='|')
                     {++cnt_stmnts;}
                     if(str3[i]=='_')
                     {++cnt_variables;}}
                     }
```

```
{fif}({felsif}?){false}       {printf("fif-felsif-felse
detected\n");
                          strcpy(str4,yytext);
                          int i;
                          for(i=0;i<strlen(str4);i++)
                          {if(str4[i]=='|')
                          {++cnt_stmnts;}
                          if(str4[i]=='_')
                          {++cnt_variables;}}
                          }
{freturn}         {printf("Program ended \n");
                      ++cnt_stmnts;}
%%



int yywrap()
{
    return 1;
}

int main()
{
    yyin=fopen("input.txt", "r");
    yylex();
    printf("Total variables: %d\n", cnt_variables);
    printf("Total statements: %d\n", cnt_stmnts);
    printf("Total keywords: %d\n", cnt_keyword+2);
    return 0;
}
```

Input File:

```
#include<<flang.h>>
fmain [ ]
$
fint _f|
fstring _j|
ffloat _a=9.0|
##single line comment
/*
multiline
comment
*/
ffor @ fint _ng=0;_ng<=10;_ng++ @
(
_j=_k+_m|
)
fwhile @ _m>=10 @
(
_m++|
)
fif @ _n<=_j @
(
_j=_k+_m|
)
felsif @ _n<=_j @
(
_j=_k+_m|
)
felse
(
_j=_k+_m|
)
fret 0|
$
```

Output file:

```
Directive is declared correctly
In the main function
This is a valid declaration
This is a valid declaration
This is a valid declaration
Single line comment detected

Multi line comment detected

for loop detected
fwhile loop detected
fif-felsif-felse detected
Program ended
Total variables: 21
Total statements: 9
Total keywords: 2
```