SRM UNIVERSITY AP
Andhra Pradesh

Department of Computer Science and Engineering
Compiler Design Lab (CS 306)

Week 4: Implementation of lexical analyser using LEX

Week 4 Program

Implement lexical analyser using LEX for recognizing the following tokens:

- A minimum of 10 keywords of your choice
- Identifiers with the regular expression : letter(letter | digit)*
- Signed as well as unsigned integers
- Signed as well as unsigned Floats in fractional as well as exponential notation.
- Relational operators: <, >, <=, >=, ==, !=
- Assignment Operator:=
- Ignores everything between comments: single line as well as multiline comments as in C
- Storing identifiers in symbol table
- Using files for input and output.

Instructions:

- Explanation and code of a few concepts explaining the requirements in the program are given below.
- You are required to develop a lexical analyser recognizing all given tokens in the program description.
- Upload into your Github accounts under the folder **Week4-Lab-exercise**

Programs:

1. **LEX Program for identifying the below and print the identified token along with information.**

   Keywords: int,char,double,void,main
   Identifier: letter(letter|digit)*
   Integer, Float and Relational operators

   Code:

```
digit   [0-9]*
id      [a-zA-Z][a-zA-Z0-9]*
num     [0-9]*\.[0-9]*

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

```
%%

int |
float |
char |
double |
void        { printf("\n %s is keyword",yytext);}
"<="        {printf("\n %s is Relational operator Lessthan or Equal to",yytext);}
"<"         {printf("\n %s is Relational operator Lessthan",yytext);}
">="        {printf("\n %s is Relational operator Greaterthan or Equal to",yytext);}
">"         {printf("\n %s is Relational operator Greaterthan",yytext);}
"=="        {printf("\n %s is Relational operator Equal to",yytext);}
"!="        {printf("\n %s is Relational operator Not Equal to",yytext);}
"="         {printf("\n %s is Assignment operator",yytext);}
{id}        {printf("\n %s is identifier",yytext); }
{num}       {printf("\n %s is float",yytext);}
{digit}     {printf("\n %s is digit",yytext);}

%%

main()
{
            yylex();
}
int yywrap()
{
            return 1;
}
```

2. LEX Program for identifying

Keywords: int,char,double,void,main
Identifier: letter(letter|digit)* and storing in Symbol table
Integer and Float

Code:

```
digit [0-9]*
id [a-zA-Z][a-zA-Z0-9]*
num [0-9]*\.[0-9]*

%{
#include<stdio.h>
#include<string.h>
int cnt=0,i=0,j=0;
char st[10][10];
int look_up(char st[10][10],char *id,int n);
%}

%%

int |
float |
char |
double |
void |
main        { printf(" \n %s is keyword",yytext);}

{num}       { printf("\n %s is float",yytext);}
{id}        {  printf("\n %s is identifier",yytext);
               if (!lookup(st,yytext,i)){
                    strcpy(st[i++],yytext);cnt++;}
            }
```

```
{digit}    {printf("\n %s is digit",yytext);}

%%

main()
{
        yylex();
        printf(" No of id are : %d ",cnt);
        printf("\n the contents of symbol table are :\n");
        for(j=0;j<i;j++)
          printf("\n %s",st[j]);
}

int yywrap()
{
        return 1;
}

int lookup(char st[10][10],char *id,int n)
{
    for(j=0;j<n;j++)
      if(!strcmp(st[j],id))
        return 1;
    return 0;
}
```

3. Dealing with comments

Code::

```
digit [0-9]*
id [a-zA-Z][a-zA-Z0-9]*
num [0-9]*\.[0-9]*
%{
#include<stdio.h>
#include<string.h>
int i=0,j=0,cnt=0,n=0,com=0,scom=0;
char st[10][10];
%}

%%

\n          {scom=0;n++;}
"//"        {scom=1;printf("\n single line comment\n\n");}
"/*"        {com=1;printf("\n comment start\n");}
"*/"        {com=0;printf("\n comment end\n");}
int |
float |
char |
double |
void        {if(!com&&!scom) printf(" \n %s is keyword",yytext);}
"<="        {if (!com&&!scom) printf("\n %s is Relational operator Lessthan or Equal to",yytext);}
"<"         {if(!com&&!scom) printf("\n %s is Relational operator Lessthan",yytext);}
">="        {if(!com) printf("\n %s is Relational operator Greaterthan or Equal to",yytext);}
">"         {if(!com&&!scom) printf("\n %s is Relational operator Greaterthan",yytext);}
"=="        {if(!com&&!scom) printf("\n %s is Relational operator Equal to",yytext);}
"!="        {if (!com&&!scom) printf("\n %s is Relational operator Not Equal to",yytext);}

{id}        {if(!com&&!scom) printf("\n %s is identifier",yytext); }
{num}       {if(!com&&!scom) printf("\n %s is float",yytext);}
{digit}     {if (!com&&!scom) printf("\n %s is digit",yytext);}

%%
```

```
main()
{
        yylex();
        printf(" \n\n\n no of lines = %d\n\n",n);
        return 0;
}
yywrap()
{
       return 1;
}
```

4. LEX Program for identifying the following tokens

Keywords: int, char, double, void, main
Identifier: letter(letter|digit)* and storing in Symbol table
Integer and Float
By taking the source program in file.

```
digit   [0-9]*
id      [a-zA-Z][a-zA-Z0-9]*
num     digit*\.digit*
%{
#include<stdio.h>
#include<string.h>
int cnt=0,i=0,j=0;
char st[10][10];
FILE *ifp,*ofp;
int look_up(char st[10][10],char *id,int n);
%}

%%
int |
float |
char |
double |
void |
main    { fprintf(ofp," \n %s is keyword",yytext);}
{num}   { fprintf(ofp,"\n %s is float",yytext);}

{id}    {  fprintf(ofp,"\n %s is identifier",yytext);
        cnt++;
        if(!look_up(st,yytext,i))
            strcpy(st[i++],yytext);
        }

{digit} {fprintf(ofp,"\n %s is digit",yytext);}

%%

main()
{

        char ip_file[10],op_file[10];;
        printf("\n Enter input file name\n");
        scanf("%s",ip_file);
        printf("\n Enter output file name\n");
        scanf("%s",op_file);

        printf("The input file u entered is :%s",ip_file);
        ifp=fopen(ip_file,"r");
        ofp=fopen(op_file,"w");

        if(ifp==NULL)
        {
```

```
                printf("\n Input file Doesnot exists\n");
                exit(0);
        }
        else
        {
                yyin=ifp;yyout=ofp;
                yylex();
                fclose(ifp); fclose(ofp);
                printf("\n the contents of symbol table are :\n");
                for(j=0;j<i;j++)
                        printf("\n %s",st[j]);
                printf("\n\n");
        }
        return 0;
}

int yywrap()
{
        return 1;
}
int look_up(char st[10][10],char *id,int n)
{
      for(j=0;j<n;j++)
        if(!strcmp(st[j],id))
          return 1;
      return 0;
}
```

5. LEX Program that use command line arguments.

Command line arguments: If command line arguments are to be used in program, the main function is to take the following form.

int main(int argc, char **argv) where

- **argc (ARGument Count)** is int and stores number of command-line arguments passed by the user including the name of the program. So if we pass a value to a program, value of argc would be 2 (one for argument and one for program name)
- **argv(ARGument Vector)** is array of character pointers listing all the arguments.
- If argc is greater than zero, the array elements from argv[0] to argv[argc-1] will contain pointers to strings.
- argv[0] is the name of the program , After that till argv[argc-1] every element is command -line arguments.

Example program which passes input file as command line argument is given below:

```
digit           [0-9]*
id              [a-zA-Z][a-zA-Z0-9]*
num             digit*\.digit*
%{
      #include<stdio.h>
      #include<string.h>
      int cnt=0,i=0,j=0;
      char st[10][10];
      int look_up(char st[10][10],char *id,int n);
%}

%%
int |
float |
char |
double |
void{ printf(" \n %s is keyword",yytext);}
{num}           { printf("\n %s is float",yytext);}

{id}            { printf("\n %s is identifier",yytext);
```

```
                cnt++;
              if(!look_up(st,yytext,i))
                   strcpy(st[i++],yytext);
              }

{digit} {printf("\n %s is digit",yytext);}

%%

main(int argc, char **argv) {

   yyin=fopen(argv[1],"r");    // passing  input file name as argv[1]

   yylex();
   return 0;
}

int yywrap()
{
      return 1;
}
int look_up(char st[10][10],char *id,int n)
{
       for(j=0;j<n;j++)
         if(!strcmp(st[j],id))
           return 1;
       return 0;
}
```

Executing this program would be

```
:\> flex prog.l
:\> cc lex.yy.x
:\> a <inputfilename>
```

Testcases: Test your program with test cases covering all requirements.