NMAM INSTITUTE OF TECHNOLOGY

(An Autonomous Institution affiliated to VTU, Belgaum)
(NBA Accredited, ISO 9001:2008 Certified)
Nitte – 574110, Karkala, Udupi District, Karnataka, India

Department of Computer Science and Engineering

Laboratory Manual

Compiler Design Lab

19CS706

Compiler Design Lab

19CS706

VII sem

Software tools used

1. Vi editor,vim,gedit
2. byacc, flex
3. Linux O.S(Ubuntu 16.04 LTS)

LAB EVALUATION SCHEME

CIE Marks Distribution:

Assessment	Weightage in Marks
Continuous evaluation	10
MiniProject	20
LAB Internal	20
Total	**50**

RUBRICS FOR CONTINUOUS EVALUATION IN REGULAR LABS:

Rubrics for Evaluation	Distribution of Marks
Execution (Number of programs completed in a day)	5
Quality of output, neatness in coding and noting the observation	3
Viva (preparedness towards lab experiments)	2
TOTAL	10

Marks Distribution for Lab MSE:

Evaluation	Distribution of Marks
Writeup and Execution of Lex program	8
Writeup and Execution of YACC program	8
viva	4
TOTAL	20

Marks Distribution for MINI PROJECT:

Rubrics for Evaluation	Distribution of Marks
Implementing of a compiler for a hypothetical language(lexical+parser)	15(6+9)
Report	05
TOTAL	20

Marks Distribution of SEE:

Evaluation	Distribution of Marks
Lex Progarm	15+5=20
Yacc Program	15+5=20
Viva	10
Total	50

Guidelines for mini project:

1. Students should implement a compiler for a hypothetical language.

2. A group can have maximum 3 members.

3. Students should submit a neat report before MSE-II as apart of evaluation.

Prepared by:

Mrs.Anusha Anchan

Assistant Professor G-I

Dept of CSE

NMAMIT, Nitte

List of Experiments

Flex (Fast Lexical Analyzer Generator)

-

FLEX (fast lexical analyzer generator) is a tool/computer program for generating lexical analyzers (scanners or lexers) written by Vern Paxson in C around 1987. It is used together with Berkeley Yacc parser generator or GNU Bison parser generator. Flex and Bison both are more flexible than Lex and Yacc and produces faster code. Bison produces parser from the input file provided by the user.

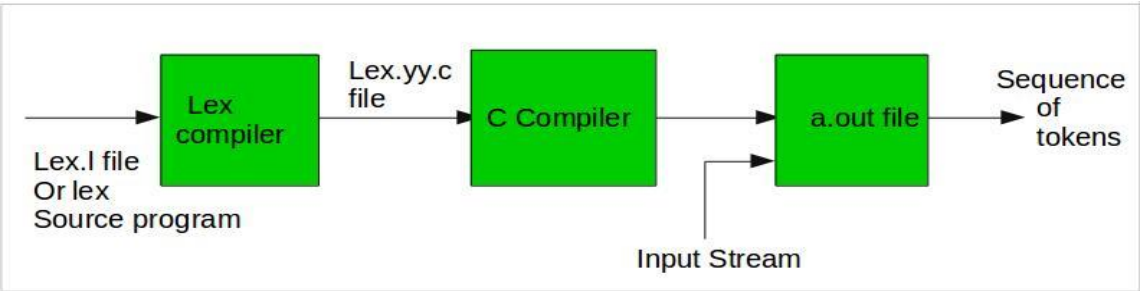
The function **yylex()** is automatically generated by the flex when it is provided with a **.l file** and this yylex() function is expected by parser to call to retrieve tokens from current/this token stream.

Note: The function yylex() is the main flex function which runs the Rule Section and extension (.l) is the extension used to save the programs.

Installing Flex on Ubuntu:

```
sudo apt-get update
sudo apt-get install flex
```

Given image describes how the Flex is used:



Step 1: An input file describes the lexical analyzer to be generated named lex.l is written in lex language. The lex compiler transforms lex.l to C program, in a file that is always named lex.yy.c.
Step 2: The C complier compile lex.yy.c file into an executable file called a.out.
Step 3: The output file a.out take a stream of input characters and produce a stream of tokens.

Program Structure:

In the input file, there are 3 sections:

1. Definition Section: The definition section contains the declaration of variables, regular definitions, manifest constants. In the definition section, text is enclosed in "**%{ %}**" brackets. Anything written in this brackets is copied directly to the file **lex.yy.c**

Syntax:

```
%{
   // Definitions
%}
```

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

Syntax:

```
%%
pattern  action
%%
```

3. User Code Section: This section contain C statements and additional functions. We can also compile these functions separately and load with the lexical analyzer.

Basic Program Structure:

```
%{
// Definitions
%}

%%
Rules
%%

User code section
```

Examples: Table below shows some of the pattern matches.[Step 2]

Pattern	It can match with
[0-9]	all the digits between 0 and 9
[0+9]	either 0, + or 9
[0, 9]	either 0, ', ' or 9
[0 9]	either 0, ' ' or 9
[-09]	either -, 0 or 9
[-0-9]	either – or all digit between 0 and 9
[0-9]+	one or more digit between 0 and 9
[^a]	all the other characters except a
[^A-Z]	all the other characters except the upper case letters

Pattern	It can match with
a{2, 4}	either aa, aaa or aaaa
a{2, }	two or more occurrences of a
a{4}	exactly 4 a's i.e, aaaa
.	any character except newline
a*	0 or more occurrences of a
a+	1 or more occurrences of a
[a-z]	all lower case letters
[a-zA-Z]	any alphabetic letter
w(x | y)z	wxz or wyz

How to run the program:

To run the program, it should be first saved with the extension **.l or .lex**. Run the below commands on terminal in order to run the program file.

Step 1: lex filename.l or lex filename.lex depending on the extension file is saved with

Step 2: gcc lex.yy.c

Step 3: ./a.out

Step 4: Provide the input to program in case it is required

Note: Press **Ctrl+D** or use some **rule** to stop taking inputs from the user. Please see the output images of below programs to clear if in doubt to run the programs.

Example 1: Count the number of characters in a string

```
/*** Definition Section has one variable  which can be
accessed inside yylex()
and main() ***/
%{
int count = 0;
%}

/*** Rule Section has three rules, first rule matches
with capital letters, second rule  matches with any
character except newline and third rule does not take
input after the enter***/
%%
[A-Z] {printf("%s capital letter\n", yytext);
count++;}
. {printf("%s not a capital letter\n", yytext);}
\n  {return 0;}
%%

/*** Code Section prints the number of  capital letter
present in the given input***/

int yywrap(){}
```

```
int main()
{
     // Explanation:
     // yywrap() - wraps the above rule section
     /* yyin - takes the file pointer which contains
the input*/
     /* yylex() - this is the main flex function which
runs the Rule Section*/
     // yytext is the text in the buffer
     // Uncomment the lines below to take input from
file
     // FILE *fp;
     // char filename[50];
     // printf("Enter the filename: \n");
    // scanf("%s",filename);
    // fp = fopen(filename,"r");
    // yyin = fp;

     yylex();
     printf("\nNumber of Captial letters in the given
input - %d\n", count);
     return 0;
}
```

LEX PROGRAMS

1: Program to count the number of vowels and consonants in a given String

```
%{
        #include<stdio.h>
        int vowels=0;
        int cons=0;
%}

%%
[aeiouAEIOU] {vowels++;}
[a-zA-Z] {cons++;}
. {;}
%%

int yywrap()
{
        return 1;
}

void main()
{
        printf("Enter the string.. at end press Ctrl d\n");
        yylex();
        printf("No of vowels=%d\nNo of consonants=%d\n",vowels,cons);
}
```

P2:Program to count the number of characters, words, spaces and lines in a given input file

```
%{
        #include<stdio.h>
        int c=0,w=0,s=0,l=0;
%}
WORD [^ \t\n,\.:]+
EOL [\n]
BLANK [ ]
%%
{WORD} {w++; c=c+yyleng;}
{BLANK} {s++;}
{EOL} {l++;}
%%

int yywrap()
```

```
{
	return 1;
}

void main(int argc, char *argv[])
{
	if(argc!=2)
	{
		printf("Usage: ./a.out p2in.txt\n");
		exit(0);
	}
	yyin=fopen(argv[1],"r");
	yylex();
	printf("\nNo of characters=%d\nNo of words=%d\nNo of
spaces=%d\nNo of lines=%d\n",c,w,s,l);
}
```

P3:Program to count no of:a) +ve and –ve integers b) +ve and –ve fractions

```
%{
	#include<stdio.h>
	int posint=0, negint=0,posfrac=0, negfrac=0;
%}

%%
[-][0-9]+ {negint++;}
[+]?[0-9]+ {posint++;}
[+]?[0-9]*\.[0-9]+ {posfrac++;}
[-][0-9]*\.[0-9]+ {negfrac++;}
%%


int yywrap()
{
	return 1;
}

void main(int argc, char *argv[])
{
	if(argc!=2)
	{
		printf("Usage: ./a.out p3in.txt\n");
		exit(0);
	}
	yyin=fopen(argv[1],"r");
	yylex();
printf("No of +ve integers=%d\n No of –ve integers=%d\n No of +ve
```

```
fractions=      %d\n No of –ve fractions=%d\n", posint, negint, posfrac,
        negfrac);
}
```

P4:Program to count the no of comment line in a given C program. Also eliminate them and copy that program into separate file

```
/*l4:Program  to count the no of comment line in a given C program.
Also eliminate them and copy that program into separate file*/
%{
        #include<stdio.h>
        int com=0;
%}
%s COMMENT
%%
"//".* {com++;}
"/*" {BEGIN COMMENT ;}
<COMMENT>"*/" {BEGIN 0;com++ ;}
<COMMENT>\n {com++;}
<COMMENT>. {;}
.|\n {fprintf(yyout,"%s",yytext);}
%%
int yywrap()
{
return 1;
}

main(int argc, char *argv[])
{
        if(argc!=3)
        {
                printf("Usage: ./a.out p4in.txt p4out.txt\n");
                exit(0);
        }
        yyin=fopen(argv[1],"r");
        yyout=fopen(argv[2],"w");
        yylex();
        printf("No. of comment lines=%d\n",com);
}
```

Input:p4in.txt

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

```
int a,b;
printf("Enter the value of a and b\n");
/* Input values for a and b*/
// Input values for a and b
scanf("%d%d",&a,&b);/*values entered are assigned to a and b
vsdcfsd
dsg
sdb */
printf("The numbers are %d and %d\n",a,b)
printf("The numbers are %d and %d\n",a,b)
printf("The numbers are %d and %d\n",a,b);
}
```

Output:p4out.txt

```
#include<stdio.h>
main()
{
int a,b;
printf("Enter the value of a and b\n");

scanf("%d%d",&a,&b);
printf("The numbers are %d and %d\n",a,b)
printf("The numbers are %d and %d\n",a,b)
printf("The numbers are %d and %d\n",a,b);
}
```

P5:Program to count the no of ‘scanf’ and ‘printf’ statements in a C program. Replace them with ‘readf’ and ‘writef’ statements respectively.

```
%{
#include<stdio.h>
int pc=0, sc=0;
%}
%%
"printf" { fprintf(yyout,"writef"); pc++;}
"scanf" { fprintf(yyout,"readf"); sc++;}
%%
int yywrap()
{
return 1;
}

main(int argc, char *argv[])
{
```

```
        if(argc!=3)
        {
                printf("Usage: ./a.out p5in.txt p5out.txt \n");
                exit(0);
        }
        yyin=fopen(argv[1],"r");
        yyout=fopen(argv[2],"w");
        yylex();
        printf("No of printf statements = %d\n No of scanf
statements=%d\n", pc, sc);
}
```

P6:Program to recognize a valid arithmetic expression and identify the identifiers and operators present. Print them separately.

```
%{
#include<stdio.h>
#include<string.h>
int noprt=0, nopnd=0, valid=1, top=-1,l=0, j=0;
char opnd[10][10], oprt[10][10], a[100];
%}
%%
"(" { top++; a[top]='(' ; }
"{" { top++; a[top]='{' ; }
"[" { top++; a[top]='[' ; }
")" { if(a[top]!='(')
      {
       valid=0;  return;
      }
      else
       top--;
   }
"}" { if(a[top]!='{')
      {
       valid=0;  return;
      }
      else
       top--;
   }
"]" { if(a[top]!='[')
      {
       valid=0;  return;
      }
      else
       top--;
   }
```

```
"+"|"-"|"*"|"/" { noprt++;strcpy(oprt[l], yytext);l++; }
[0-9]+|[a-zA-Z][a-zA-Z0-9]* {nopnd++; strcpy(opnd[j],yytext);j++;}
%%
int yywrap()
{
return 1;
}
main()
{
        int k;
        printf("Enter the expression.. at end press Ctrl d\n");
        yylex();
        if(valid==1 && (nopnd-noprt)==1 && top==-1)
        {
                printf("The expression is valid\n");
                printf("The operators are\n");
                for(k=0;k<l;k++)
                        printf("%s\n",oprt[k]);
            printf("Operands are\n");
                for(k=0;k<j;k++)
                        printf("%s\n",opnd[k]);
        }
        else
                printf("The expression is invalid\n");
}
```

P7:Program to recognize whether a given sentence is simple or compound

```
%{
        #include<stdio.h>
        int is_simple=1;
%}
%%
[ \t\n]+[aA][nN][dD][ \t\n]+ {is_simple=0;}
[ \t\n]+[oO][rR][ \t\n]+ {is_simple=0;}
[ \t\n]+[bB][uU][tT][ \t\n]+ {is_simple=0;}
. {;}
%%
int yywrap()
{
return 1;
}

main()
{
```

```
    printf("Enter the sentence.. at end press ^d\n");
    yylex();
    if(is_simple==1)
    {
          printf("The given sentence is simple");
    }
    else
    {
          printf("The given sentence is compound");
    }
}
```

P8: Program to recognize and count the number of identifiers in a given input file

```
%{
   #include<stdio.h>
   int id=0;
%}
%%
[a-zA-Z][a-zA-Z0-9_]* {id++;ECHO;printf("\n");}
.+ {;}
\n {;}
%%
int yywrap()
{
    return 1;
}
main(int argc,char *argv[])
{
  if(argc!=2)
  {
     printf("Usage: ./a.out p8in.txt\n");
     exit(0);
  }
 yyin=fopen(argv[1],"r");
 printf("Valid identifiers are\n");
 yylex();
 printf("No.of identifiers=%d\n",id);
}
```

****

YACC PROGRAMS

Steps to Execute

yacc -d pgmname.y
lex pgmname.l
cc y.tab.c lex.yy.c -ly -ll
./a.out

Y1:Program to recognize valid declaration statements

```
%{
   #include<stdio.h>
%}
% token SP SC CM NL ID INT CHAR
%%
S:type SP list SC NL( printf (“valid”); exit(0);}
;
type:INT
       |CHAR
       ;
list: list SP CM SP ID
     | list CM ID
     | ID
     ;
%%
int yyerror(char *msg)
{
        printf("Invalid Expression\n");
        exit(0);
}
main ()
{
        printf("Enter the declaration statement\n");
        yyparse();
}
```

Y2:Program to recognize nested IF control statements and display the levels of nesting

```
%{
   #include<stdio.h>
   #include<stdlib.h>
   int count=0;
%}
%token IF RELOP S NUMBER ID NL
```

```
%%
stmt : if_stmt NL { printf("No of nested if statements=%d\n",count);
exit(0);}
          ;
if_stmt : IF '(' cond ')' '{' if_stmt '}' {count++;}
        | S
        ;
cond :  x RELOP x
          ;
x : ID
  |NUMBER
  ;
%%
int yyerror(char *msg)
{
          printf("Invalid Expression\n");
          exit(0);
}
main ()
{
          printf("Enter the statement\n");
          yyparse();
}
```

LEX PART

```
%{
  #include "y.tab.h"
%}
%%
"if" {return IF;}
[sS][0-9]* {return S;}
[a-zA-Z][a-zA-Z0-9_]* {return ID;}
[0-9]+ {return NUMBER;}
">"|"<"|">="|"<="|"=="|"!=" {return RELOP;}
\n {return NL;}
. {return yytext[0];}
%%
```

Y3:Program to check the syntax of a simple expression involving operators +, -, * and /

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

```
%token NUMBER ID NL
%left '+' '-'
%left '*' '/'
%%
stmt : exp NL { printf("Valid Expression\n"); exit(0);}
     ;
exp : exp '+' exp
    | exp '-' exp
          | exp '*' exp
          | exp '/' exp
          | '(' exp ')'
    | '{' exp '}'
    | '[' exp ']'
          | ID
          | NUMBER;
%%
int yyerror()
{
          printf("Invalid Expression\n");
          exit(0);
}
main ()
{
          printf("Enter the expression\n");
          yyparse();
}
```

Lex Part

```
%{
  #include "y.tab.h"
%}

%%
[0-9]+ {return NUMBER;}
[\n] {return NL;}
. {return yytext[0];}
%%
```

Y4:Program to evaluate an arithmetic expression involving operating +, -, * and /

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

```
%token NUMBER NL
%left '+' '-'
%left '*' '/'
%%
stmt: exp NL { printf("Value=%d\n",$1);exit(0); };
exp: exp '+' exp {$$=$1+$3;}
   |exp '-' exp {$$=$1-$3;}
   |exp '*' exp {$$=$1*$3;}
   |exp '/' exp { if($3==0)
                  {
                    printf("Canot divide by zero\n");
                    exit(0);
                  }
                  else
                    $$=$1/$3;
                }
   |'(' exp ')' {$$=$2;}
   |NUMBER {$$=$1;}
   ;
%%
int yyerror(char *msg)
{
   printf("Invalid Expression\n");
   exit(0);
}
main()
{
  printf("Enter an Expression\n");
  yyparse();
}
```

Lex part

```
%{
   #include "y.tab.h"
   extern int yylval;
%}
%%
[0-9]+ {yylval=atoi(yytext);return NUMBER;}
\n {return NL;}
. {return yytext[0];}
%%
```

Y5:Program to recognize a valid variable, which starts with a letter, followed by any number of letters or digits

```
%{
  #include<stdio.h>
  #include<stdlib.h>
%}
%token DIGIT LETTER UND NL
%%
stmt : variable NL { printf("Valid Identifier\n"); exit(0);};

variable : LETTER alphanumeric;

alphanumeric: LETTER alphanumeric
            | DIGIT alphanumeric
            | UND alphanumeric
            | LETTER
            | DIGIT
            | UND ;
%%
int yyerror()
{
        printf("Invalid Identifier\n");
        exit(0);
}
main ()
{
        printf("Enter the variable name\n");
        yyparse();
}
```

lex part

```
%{
  #include "y.tab.h"
%}
%%
[a-zA-Z] {return LETTER;}
[0-9] {return DIGIT;}
[_] {return UND;}
[\n] {return NL;}
. {return yytext[0];}
```

```
%%
```

Y6:Program to recognize strings ‘aaab’, ‘abbb’, ‘ab’ and ‘a’ using grammar ($a^n b^n$, n>=0)

```
%{
  #include<stdio.h>
  #include<stdlib.h>
%}
%token A B NL
%%
stmt:s NL {printf("Valid string\n");exit(0);}
    ;
s: A s B
 |
 ;
%%
int yyerror(char *msg)
{
  printf("Invalid String\n");
  exit(0);
}
main()
{
  printf("Enter the string\n");
 yyparse();
}
```

lex part

```
%{
  #include "y.tab.h"
%}
%%
[aA] {return A;}
[bB] {return B;}
\n {return NL;}
. {return yytext[0];}
%%
```

Y7:Program to recognize the grammar ($a^n b$, n>=10)

METHOD1

```
%{
  #include<stdio.h>
  #include<stdlib.h>
%}
%token A B NL
%%
stmt:A A A A A A A A A A s B NL {printf("Valid string\n");exit(0);}
    ;
s:s A
 |
 ;
%%
int yyerror(char *msg)
{
  printf("Invalid String\n");
  exit(0);
}
main()
{
  printf("Enter the string\n");
 yyparse();
}
```

lex part

```
%{
  #include "y.tab.h"
%}
%%
[aA] {return A;}
[bB] {return B;}
\n {return NL;}
. {return yytext[0];}
%%
```

METHOD2:

```
%{
```

```
  #include<stdio.h>
  #include<stdlib.h>
%}
%token A B NL
%%
stmt:s NL {printf("Valid string\n");exit(0);}
    ;
s: A s B
 |
 ;
%%
int yyerror(char *msg)
{
  printf("Invalid String\n");
  exit(0);
}
main()
{
  printf("Enter the string\n");
 yyparse();
}
```

lex part

```
%{
  #include "y.tab.h"
  int c=0;
%}
%%
aaaaaaaaaa[a]* {return A;}
[bB] {c++;
     if(c==1)
       return B;
     else
     {
       printf("Invalid String\n");
       exit(0);
     }
     }
\n {return NL;}
. {return yytext[0];}
%%
```

****

MINI PROJECT QUESTIONS

Design a compiler using c/c++ for the following hypothetical languages

Q1:

```
int main()
begin
   int n, i, sum = 0;
      for(i=1; i <= n; ++i)
     begin
                    expr= expr+expr;
                    end
End
```

Q2:

```
int main()
begin
    int n1, n2, n3;
```

```
        if( expr relop expr )
        begin
        printf( n1);
        end
        if ( expr relop expr )
        begin
                printf( n2);
        end
        if( expr relop expr )
         begin
        printf( n3);
        end
end
```

Q3:

```
int main()
begin
   int n, re = 0, rem;
   while(expr)
   begin
        expr=expr+expr;
   end
end
```

Q4:

```
int main()
begin
   int n1, n2, i, gcd;
   if(expr relop expr)
        gcd = i;
   for(i=1; expr relop expr; ++i)
   begin
      gcd=1;
   end
end
```

Q5:

```
BEGIN
    PRINT “HELLO”
    INTEGER A, B, C
    REAL   D, E
    STRING X, Y
    A := 2
```

```
B := 4
C := 6
D := -3.56E-8
E := 4.567
X := “text1”
Y := “hello there”

FOR I:= 1 TO 5
      PRINT “Strings are [X] and [Y]”
 END
```

Q6:

```
X: integer ;

Procedure  foo( b : integer )
b := 13;
If x = 12 and b = 13 then
Printf( “by copy-in copy-out” );
Elseif x = 13 and b = 13 then
Printf( “by address” );
Else
Printf( “A mystery” );
End if;
End foo
```

Q7:

```
int main()
begin
   int count=1;
   while(n>1)
      count=count+1;
      n=n/2;
   end while
 return count
end
```

Q8:

```
int main()
begin
      int L[10];
      int maxval=L[0];
      for i=1 to n-1 do
```

```
            if L[i]>maxval
                maxval=L[i];
            endif
        endfor
        return(maxval)
end
```