DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING

COMPILER DESIGN LAB
LAB MANUAL – 2020 -21

Flex Windows
GnuWin32

1. Programs using Lex Tool

(a) Lex specification to demonstrate different regular expressions.
LEX Program for different regular expressions: digits, numbers, identifiers and keywords

```
%{
#include<stdio.h>
%}
digit           [0-9]
letter          [a-zA-Z]
id              {letter}({letter}|{digit})*
num             {digit}+
keyword         begin|end|int|if|while
relop           <|<=|>|>=|==|!=
arithop         \-|\+|\*|\/
assign          =
%%
{keyword}       {printf("It is a keyword:%s\n", yytext);}
{digit} {printf("It is a digit :%s\n",yytext);}
{num}           {printf("It is a number: %s\n", yytext);}
{id}            {printf("It is a identifier:%s\n", yytext);}
{relop}                 {printf("It is a relational op: %s\n", yytext);}
{arithop}       {printf("It is a arith op: %s\n", yytext);}
{assign}        {printf("It is a assignment op: %s\n", yytext);}
%%
int main(int argc, char *argv[])
{
    printf("Enter Something:");
    yylex();
}
int yywrap()
{
    return 1;
}
```

Execution:

```
F:\Subjects_Material\Compiler Design\cdlab>flex sample.l
```

```
F:\Subjects_Material\Compiler Design\cdlab>gcc lex.yy.c

F:\Subjects_Material\Compiler Design\cdlab>a.exe
Enter Something:a=b+sum*25
It is a identifier:a
It is a assignment op: =
It is a identifier:b
It is a arith op: +
It is a identifier:sum
It is a arith op: *
It is a number: 25
```

1(b). Lex specification to print two digit numbers in words.

```
%{
#include<stdio.h>
int twodigitsnum=0;
int onesdigit = 0, secondsdigit = 0;
%}
ones                    [0-9]
seconds                 [1-9]
num                     {seconds}{ones}
%%
{num}           {onesdigit=(int)yytext[1];
                 secondsdigit=(int)yytext[0];
                 onesdigit -=48;
                 secondsdigit -=48;
                 return(onesdigit+secondsdigit*10);}


%%
int yywrap()
{
return 1;
}

int main()
{
    int print_inwords(int);
    int temp;
    printf("Enter a two digit number:");
    twodigitsnum = yylex();
    temp = twodigitsnum;
```

```
      printf("\nInput Number = %d\n",twodigitsnum);
      onesdigit = twodigitsnum%10;
      twodigitsnum =twodigitsnum/10;
      secondsdigit = twodigitsnum%10;
      printf("In words : ");

      print_inwords(secondsdigit);
      print_inwords(onesdigit);
}
int print_inwords(int digit)
{
      switch(digit)
   {
            case 0: printf(" Zero"); break;
            case 1: printf(" One");  break;
            case 2: printf(" Two"); break;
            case 3: printf(" Three"); break;
            case 4: printf(" Four"); break;
            case 5: printf(" Five"); break;
            case 6: printf(" Six"); break;
            case 7: printf(" Seven"); break;
            case 8: printf(" Eight"); break;
            case 9: printf(" Nine"); break;
   }

}
```

Execution:

```
F:\Subjects_Material\Compiler Design\cdlab>flex two_digit_in_words.l
F:\Subjects_Material\Compiler Design\cdlab>gcc lex.yy.c
F:\Subjects_Material\Compiler Design\cdlab>a.exe
Enter a two digit number:79

Input Number = 79
In words :  Seven Nine
F:\Subjects_Material\Compiler Design\cdlab>
```

1(c). Lex specification to check the validity of given date.

```
   /**/
%{
#include<stdio.h>
int i, dd,mm,year,valid=1;
%}
```

```
slash   [/]
ddmm30          ([0-2][0-9]|30){slash}([0][4|6|9]|11)
ddmm31          ([0-2][0-9]|[3][0-1]){slash}([0][1|3|5|7|8]|10|12)
ddmm29          ([0-2][1-9]){slash}02
yyyy            [0-2][0-9][0-9][0-9]
date            {ddmm31}{slash}{yyyy}|{ddmm30}{slash}{yyyy}|{ddmm29}{slash}{yyyy}
%%
{date}          {
                 printf("Input date:%s\n", yytext);
                 dd=(int)(yytext[0]-'0');
                 dd = dd * 10 + (int)(yytext[1]-'0');
                 mm=(int)(yytext[3]-'0');
                 mm = mm * 10 + (int)(yytext[4]-'0');
                 i=6;
                 year=0;
                 while(i<=9)
                        year = year * 10 + (int)(yytext[i++]-'0');
                 if(mm==2)
                 {
                        if( (year % 4 ==0 && year %100 !=0)|| year % 400 == 0 )
                        {
                                if(dd <= 29)
                                return(valid);
                        }
                        else
                        {
                                if(dd <= 28)
                                return(valid);
                                else return(0);
                        }
                }
                else return(valid);
        }
%%
int main()
{
    int date_valid=0;
    printf("Enter Something:");
    date_valid = yylex();
    if(date_valid == 1)
     printf("\nValid Date Format\n");
    else
    printf("\nInvalid Date Format\n");

}
int yywrap()
```

```
{
return 1;
}
```

Execution:

```
F:\Subjects_Material\Compiler Design\cdlab>flex date_format.l
F:\Subjects_Material\Compiler Design\cdlab>gcc lex.yy.c
F:\Subjects_Material\Compiler Design\cdlab>a.exe
Enter Something:12/10/2020
Input date:12/10/2020

Valid Date Format

F:\Subjects_Material\Compiler Design\cdlab>a.exe
Enter Something:20/01/2020
Input date:20/01/2020

Valid Date Format

F:\Subjects_Material\Compiler Design\cdlab>a.exe
Enter Something:29/02/2019
Input date:29/02/2019

Invalid Date Format

F:\Subjects_Material\Compiler Design\cdlab>a.exe
Enter Something:30/11/2020
Input date:30/11/2020

Valid Date Format

F:\Subjects_Material\Compiler Design\cdlab>
```

2. Programs using Lex Tool

(a). Lex specification to convert given octal number into decimal equivalent.

```
%{
#include<stdio.h>
int decnum, n;
%}
octdigit          [0-8]
octnum            {octdigit}+
%%
{octnum}  {printf("\nInput Octal Number: %s\n", yytext);
```

```
            n=0;
            while(n<yyleng)
            {
                decnum = (decnum * 8) + (int)(yytext[n]-'0');
                n=n+1;
            }
            return(decnum);
        }
%%
int main(int argc, char *argv[])
{
    int m;
    printf("Enter a Octal Number:");
    m = yylex();
    printf("\nDecimal Equivalent = %d", m);
}
int yywrap()
{
return 1;
}
```

Execution:

```
F:\Subjects_Material\Compiler Design\cdlab>flex octal_dec.l
F:\Subjects_Material\Compiler Design\cdlab>gcc lex.yy.c
F:\Subjects_Material\Compiler Design\cdlab>a.exe
Enter a Octal Number:12
Input Octal Number: 12
Decimal Equivalent = 10

F:\Subjects_Material\Compiler Design\cdlab>a.exe
Enter a Octal Number:125
Input Octal Number: 125
Decimal Equivalent = 85

F:\Subjects_Material\Compiler Design\cdlab>
```

2. (b). Lex specification to count no of vowels, consonants, characters, words and lines in a file.

```
/*  Lex file name : vowels_consonants.l */
%{
    #include<stdio.h>
    int vowels_count,cons_count,chars_count;
    int words_count, lines_count,spaces;
%}
vowel               [aeiouAEIOU]
```

```
consonant           [^aeiouAEIOU]
/*Rules section*/
%%
\n                  {lines_count++; chars_count++;}
{vowel}             {vowels_count++; chars_count++;}
{consonant}         {cons_count++; chars_count++;}
%%

int yywrap()
{
    return(1);
}
int main(int argc, char *argv[])
{
    yyin=fopen(argv[1], "r");
    yylex();
    printf("\n No.of Lines = %d", lines_count);
    printf("\n No.of Chars = %d", chars_count);
    printf("\n No.of Vowels = %d", vowels_count);
    printf("\n No.of Consonants = %d", cons_count);
    return(0);
}
```

```
/*Input Text file: myfile.txt*/
Hyderabad is a clean city
Telangan
```

Execution:

```
F:\Subjects_Material\Compiler Design\cdlab>flex vowels_consonants.l
F:\Subjects_Material\Compiler Design\cdlab>gcc lex.yy.c
F:\Subjects_Material\Compiler Design\cdlab>a.exe myfile.txt

 No.of Lines = 2
 No.of Chars = 35
 No.of Vowels = 11
 No.of Consonants = 22

F:\Subjects_Material\Compiler Design\cdlab>
```

3. Programs using Yacc Tool

(a). Yacc specification to demonstrate different grammars.

```
/*Lex program : file name:: expr.l
Grammar:
                stmt → expr \n
                expr → expr + term | expr - term | term
                term → term * fact | term /  fact | fact
                fact → ( expr ) | INTEGER
*/

%{
#include <stdlib.h>
void yyerror(char *);
#include "expr.tab.h"
%}
%%
[0-9]+          {
                yylval = atoi(yytext);
                return INTEGER;
                }
[-+*/)(\n]      return *yytext;
[ \t]           ;       /* skip whitespace */
.               yyerror("Invalid character");
%%

int yywrap(void)
{
return 1;
}

/*YACC program : file name:: expr.y
Grammar:
                stmt →expr \n
                expr → expr + term | expr - term | term
                term → term * fact | term /  fact | fact
                fact → ( expr ) | INTEGER
*/
%{
#include <stdio.h>
int yylex(void);
void yyerror(char *);
%}
%token INTEGER

%%
stmt:   expr '\n'               { printf("%d\n", $1); }
                |
                ;
expr:   expr '+' term           { $$ = $1 + $3; }
                | expr '-' term         { $$ = $1 - $3; }
```

```
            | term           { $$ = $1; }
            ;
term:   term '*' fact        { $$ = $1 * $3;}
            | term '/' fact { $$ = $1 / $3; }
            | fact           { $$ = $1; }
            ;
fact:   INTEGER              { $$ = $1; }
            | '(' expr ')'   { $$ = $2; }
            ;
%%

void yyerror(char *s)
{
fprintf(stderr, "%s\n", s);
}

int main(void)
{
printf("\nEnter arithmetic Expression :");
yyparse();
return 0;
}
```

Execution:

```
F:\Subjects_Material\Compiler Design\cdlab>yacc -d expr.y
F:\Subjects_Material\Compiler Design\cdlab>flex expr.l
F:\Subjects_Material\Compiler Design\cdlab>gcc lex.yy.c expr.tab.c -o expr
F:\Subjects_Material\Compiler Design\cdlab>expr.exe
Enter arithmetic Expression :( 3 + 5 ) * 4
32
^Z

F:\Subjects_Material\Compiler Design\cdlab>expr.exe
Enter arithmetic Expression :2 + 3 + 4 * 5 - 6
19
```

3.(b). Yacc specification to find sentence validity.

```
/*Lex program for a sentence validity: file name:: sentence_valid.l
Grammar:
            S →CC
            C →cC
            C →d
*/
```

```
%{
  /* Definition section */
 #include "sentence_valid.tab.h"
%}

/* Rule Section */
%%
[c]  {return c;}
[d]            { return d;}
\n             {return NL;}
.              {return yytext[0];}
%%

int yywrap()
{
 return 1;
}

/*
YACC program for a sentence validity: file name:: sentence_valid.y
Grammar:
               S→CC
               C→cC
               C→d
*/
/*Definitions Section*/
%{
#include<stdio.h>
#include<stdlib.h>
int yylex(void);
void yyerror(char *);
%}
%token c d NL

/*Grammar Rules and semantic rules*/
%%
L:   S NL    {printf("\nValid sentence"); exit(0);}
     ;
S:   C C      |
     ;
C:   c C
   | d
     ;
```

```
%%

void yyerror(char *msg)
{
printf("\nInvalid sentence or error");
}

int main()
{
printf("\nEnter a sentence : ");
yyparse();
return 0;
}
```

Execution:

```
F:\Subjects_Material\Compiler Design\cdlab>yacc -d sentence_valid.y
F:\Subjects_Material\Compiler Design\cdlab>flex
sentence_valid.l
F:\Subjects_Material\Compiler Design\cdlab>gcc lex.yy.c sentence_valid.tab.c

F:\Subjects_Material\Compiler Design\cdlab>a.exe
Enter a sentence : cdccd
Valid sentence

F:\Subjects_Material\Compiler Design\cdlab>a.exe
Enter a sentence : cccd
Invalid sentence or error
F:\Subjects_Material\Compiler Design\cdlab>
```

3 (c). Yacc specification to evaluate expressions using precedence.

```
/*Lex program : file name:: eval_prec.l
Grammar:
            stmt → expr \n
            expr → expr + expr | expr - expr
            expr → expr * expr | expr /  expr
            expr → ( expr ) | INTEGER
*/

%{
#include <stdlib.h>
void yyerror(char *);
#include "expr.tab.h"
%}
%%
```

```
[0-9]+          {
                        yylval = atoi(yytext);
                        return INTEGER;
                        }
[-+*/)(\n]      return *yytext;
[ \t]           ;       /* skip whitespace */
.                       yyerror("Invalid character");
%%

int yywrap(void)
{
return 1;
}

/*YACC program : file name:: eval_prec.y
Grammar:
                stmt →expr \n
                expr → expr + expr | expr - expr
                expr → expr * expr | expr /  expr
                expr → ( expr ) | INTEGER
*/
%{
#include <stdio.h>
int yylex(void);
void yyerror(char *);
%}
%token INTEGER
%left '+' '-'
%left '*' '/'

%%
stmt:   expr '\n'               { printf("%d\n", $1); }
                |
                ;
expr:   expr '+' expr           { $$ = $1 + $3; }
                | expr '-' expr { $$ = $1 - $3; }
                | expr '*' expr { $$ = $1 * $3;}
                | expr '/' expr         { $$ = $1 / $3; }
                ;
expr:   INTEGER                 { $$ = $1; }
                | '(' expr ')'  { $$ = $2; }
                ;
%%

void yyerror(char *s)
{
```

```
fprintf(stderr, "%s\n", s);
}

int main(void)
{
printf("\nEnter arithmetic Expression :");
yyparse();
return 0;
}
```

Execution:

```
F:\Subjects_Material\Compiler Design\cdlab>yacc -d eval_prec.y
F:\Subjects_Material\Compiler Design\cdlab>flex eval_prec.l
F:\Subjects_Material\Compiler Design\cdlab>gcc lex.yy.c eval_prec.tab.c -o eval_prec
F:\Subjects_Material\Compiler Design\cdlab>eval_prec.exe

Enter arithmetic Expression :2 + 3 * 4
14
^Z
F:\Subjects_Material\Compiler Design\cdlab>eval_prec.exe
Enter arithmetic Expression :5 + 6 - 2 * 7 / 3 + 8
15

F:\Subjects_Material\Compiler Design\cdlab>
```

4. Programs using Yacc Tool

a. Yacc specification to convert binary numbers to decimal numbers

```
/*Lex program : file name:: yacc_bin_dec.l
Grammar:
                N -> L \n
                L -> L B | B
                B -> 0 | 1
*/

%{
#include <stdlib.h>
void yyerror(char *);
#include "yacc_bin_dec.tab.h"
%}
%%
0               {       //printf("Bit : %s ", yytext);
                        yylval = atoi(yytext);
                        //printf("yy Bit : %d ", yylval);
                        return a;
                }

1               {
                        //printf("Bit : %s ", yytext);
                        yylval = atoi(yytext);
                        //printf("yy Bit : %d", yylval);
                        return b;
                }
\n              { return *yytext; }
[ \t]           ;       /* skip whitespace */
.                       yyerror("Invalid character");
%%

int yywrap(void)
{
return 1;
}

/*
YACC program : file name:: yacc_bin_dec.y
Grammar:
                N -> L \n
                L -> L B | B
                B -> 0 | 1
*/

%{
#include <stdio.h>
#include<stdlib.h>
int yylex(void);
void yyerror(char *);
%}
%token a b

%%
N:      L '\n'          { printf("\nDecimal Number: %d\n", $$);
                                 exit(0);
                                }
L:      L B     { $$ = $1 * 2 + $2; }
        | B     { $$ = $1; }
B:      a       { $$ = $1; }
        | b     { $$ = $1; };
%%
```

```
void yyerror(char *s)
{
fprintf(stderr, "%s\n", s);
}

int main(void)
{
printf("\nBinary Number?: ");
yyparse();
return 0;
}
```

Execution:

```
F:\Subjects_Material\Compiler Design\cdlab>yacc -d yacc_bin_dec.y
F:\Subjects_Material\Compiler Design\cdlab>flex yacc_bin_dec.l
F:\Subjects_Material\Compiler Design\cdlab>gcc lex.yy.c yacc_bin_dec.tab.c
F:\Subjects_Material\Compiler Design\cdlab>a.exe
Binary Number?: 101011

Decimal Number: 43

F:\Subjects_Material\Compiler Design\cdlab>
```

4. (b). Yacc specification to check the validity of given date.

```
  /*
Lex program for a date validity : file name: file name:: yacc_date_valid.l
Grammar:
L -> dt \n
dt -> dd / mm / yyyy
dd -> DM
mm -> DM
yyyy -> YYYY
*/
%{
#include <stdlib.h>
void yyerror(char *);
#include "yacc_date_valid.tab.h"
int month, year;
int i;
%}
slash       [/]
ddmm                ([0-2][0-9]|[3][0-1])
yyyy        ([0-9][0-9][0-9][0-9])
%%
{ddmm}      { yylval = atoi(yytext);
              return(DM);
            }
{yyyy}      { yylval = atoi(yytext);
              return(YYYY);
            }
{slash}     { return *yytext; }
[ \t]       ;       /* skip whitespace */
.           yyerror("Invalid character");
%%
int yywrap()
{
return 1;
}
/*
YACC program : for a date validity:  file name:: yacc_date_valid.y
Grammar:
L -> dt \n
dt -> dd / mm / yr  { code for date validation }
dd -> DM
mm -> DM
yr -> YYYY
*/

%{
#include <stdio.h>
#include<stdlib.h>
int yylex(void);
void yyerror(char *);
%}
%token DM YYYY

%%
L:   dt '\n'
dt:  dd '/' mm '/' yr { if($3 == 2)
                     {
                       if( ($5 % 4 == 0 && $5 % 100 !=0)||($5 % 400 ==0))
                      {
                             if($1 <=29)
                             printf("\nIt is a valid date.");
                      }
                      else if( $1 <= 28)
                             printf("\nIt is a valid date.");
                      else
                     {
                             printf("\nIt is NOT a valid date.");
                             printf("\nNon Leap Year: %d. Its month:%d can't contain %d
                                     Days",$5,$3,$1);
                     }
                    }
                    else if( $3==1||$3==3||$3==5||$3==7||$3==8||$3==10||$3==12)
                    {
                      if( $1 <= 31)
                             printf("\nIt is a valid date.");
                      else
                     {
                       printf("\nIt is NOT a valid date.");
```

```
          printf("\nmonth:%d can't contain %d Days",$3,$1);
         }
        }
        else if( $3==4||$3==6||$3==9||$3==11)
        {
            if( $1 <= 30)
                printf("\nIt is a valid date.");
            else
            {
                printf("\n It is NOT a valid date.");
                printf("\nmonth:%d can't contain %d Days",$3,$1);
            }
        }
        else
        {
            printf("\nIt is NOT a valid date.");
            printf("\nMonth:%d is invalid.",$3);
        }
        exit(0);
      }

dd:     DM          { $$ = $1; }
mm:     DM          { $$ = $1; }
yr:     YYYY  { $$ = $1; }

%%
void yyerror(char *msg)
{
printf("\nError msg: %s", msg);
}

int main()
{
printf("\nEnter a date(DD/MM/YYYY) :");
yyparse();
return 0;
}
```

Execution:

```
F:\Subjects_Material\Compiler Design\cdlab>yacc -d yacc_date_valid.y
F:\Subjects_Material\Compiler Design\cdlab>flex yacc_date_valid.l
F:\Subjects_Material\Compiler Design\cdlab>gcc lex.yy.c yacc_date_valid.tab.c
F:\Subjects_Material\Compiler Design\cdlab>a.exe

Enter a date(DD/MM/YYYY) :29/14/2020

It is NOT a valid date.
Month:14 is invalid.
F:\Subjects_Material\Compiler Design\cdlab>a
Enter a date(DD/MM/YYYY) :20/01/2021
```

```
It is a valid date.

F:\Subjects_Material\Compiler Design\cdlab>a

Enter a date(DD/MM/YYYY) :29/02/2021

It is NOT a valid date.
Non Leap Year: 2021. Its month:2 can't contain 29 Days

F:\Subjects_Material\Compiler Design\cdlab>a

Enter a date(DD/MM/YYYY) :31/10/2020

It is a valid date.

F:\Subjects_Material\Compiler Design\cdlab>a

Enter a date(DD/MM/YYYY) :30/09/2021

It is a valid date.

F:\Subjects_Material\Compiler Design\cdlab>a.exe

Enter a date(DD/MM/YYYY) :31/09/2020

 It is NOT a valid date.
month:9 can't contain 31 Days

F:\Subjects_Material\Compiler Design\cdlab>a

Enter a date(DD/MM/YYYY) :01/10/2001

It is a valid date.

F:\Subjects_Material\Compiler Design\cdlab>a

Enter a date(DD/MM/YYYY) :29/02/1200

It is a valid date.

F:\Subjects_Material\Compiler Design\cdlab>a

Enter a date(DD/MM/YYYY) :29/02/1500

It is NOT a valid date.
Non Leap Year: 1500. Its month:2 can't contain 29 Days

F:\Subjects_Material\Compiler Design\cdlab>
```

5. Program to find all meaningful words and generate the tokens for the given input program.

Write a Lex Program for implementing Lexical Analyzer to find Stream of tokens from a given input file. For Example input is "input.c" which contains-

```
----------------------------------
main()
{
int a, sum;
float A[MAX], x;
a=25;
x=54;
if(a<=x)
then
printf("Minimum is a");
else
printf("Maximum is x");
}
-----------------------------------
```

```
/*
The Lex Program(File like: lexanalyzer.l)
*/

%{
#include<stdio.h>
int lineno=1;
%}

letter  [a-zA-Z]
digit   [0-9]
id      {letter}({letter}|{digit})*
num     {digit}+
kword   ("int"|"char"|"float"|"print"|"if"|"main"|"then")
array   ({id}"["({num}|{id})"]")
commt   ("/*"({id}|"\n")*"*/")
spsym   [.,;'"]

%%
["\n"]                {lineno=lineno+1;}
{spsym}               {printf("\nSpecial Symbol%9s   %9d", yytext, lineno);}
"<"|"<="|">"|">="|"=="  {printf("\nRelational Op   %9s  %9d", yytext, lineno);}
{commt}               {printf("\nComment      %9s     %9d", yytext, lineno);}
"="                   {printf("\nAssignmentOp %9s     %9d", yytext, lineno);}
"+"|"-"|"*"|"/"|"%"     {printf("\nArithmetic Op %9s    %9d", yytext, lineno);}
"["|"]"|"{"|"}"|"("|")" {printf("\nParanthesis  %9s     %9d", yytext, lineno);}
{kword}               {printf("\nKeyword      %9s     %9d", yytext, lineno);}
```

```
{array}              {printf("\nArray name   %9s     %9d", yytext, lineno);}
{id}               {printf("\nIdentifier   %9s     %9d", yytext, lineno);}
{num}                {printf("\nNumber Const %9s     %9d", yytext, lineno);}
%%

main(int argc, char *argv[])
{
        printf("\nToken            Lexemes          Line Number\n");
        printf("--------------------------------------------------\n");
        if(argc >1)
           yyin = fopen(argv[1],"r");
        else
           yyin = stdin;
        yylex();
}

yywrap()
{
        printf("\n------------------------------------------------------\n");
        exit(0);
}
```

output:

```
]# lex lexanalyzer.l
]# cc lex.yy.c -o lexanalyzer -ll
]# ./lexanalyzer input.c

Token             Lexemes       Line Number
---------------------------------------------------

Keyword           main            1
Paranthesis          (           1
Paranthesis          )           1
Paranthesis          {           2
Keyword            int           3
Identifier          a            3
Special Symbol        ,            3
Identifier         sum            3
Special Symbol        ;            3
Keyword           float           4
Array name       A[MAX]            4
Special Symbol        ,            4
Identifier          x            4
Special Symbol        ;            4
Identifier          a            5
Assignment Op         =            5
```

```
Number Constant       25            5
Special Symbol        ;            5
Identifier           x           6
Assignment Op         =            6
Number Constant      54            6
Special Symbol        ;            6
Keyword             if           7
Paranthesis          (           7
Identifier           a           7
Relational Op        <=            7
Identifier           x           7
Paranthesis          )           7
Keyword           then            8
Identifier       printf            9
Paranthesis          (           9
Identifier      Minimum           10
Identifier          is           10
Identifier           a           10
Paranthesis          )           11
Special Symbol        ;           11
Identifier         else           12
Identifier       printf           13
Paranthesis          (           13
Identifier      Maximum           14
Identifier          is           14
Identifier           x           14
Paranthesis          )           15
Special Symbol        ;           15
Paranthesis          }           16
-----------------------------------------------------------
```

7. Implementing Symbol Table for given HLL.

```
/*   YACC program for Symbol Table Name:  symbol_table.y  */
%{
void yyerror(char *str);
#include<stdio.h>
#include<stdlib.h>
#include<ctype.h>

int symbol_table[52];
int symbol_val(char symbol);
void update_symbol_val(char symbol, int val);
int yylex(void);
%}
```

```
%union {int num; char id;}
%start line
%token print
%token done
%token <num> number
%token <id> identifier
%type <num> line exp term
%type <id> assignment

%%
line            : assignment ';'                {;}
                |done ';'                       {exit(1);}
                |print exp      ';'      {printf("%d\n", $2);}
                |line assignment ';'     {;}
                |line print exp ';'      {printf("%d\n",$3);}
                |line done ';'
                ;
assignment      : identifier '=' exp {update_symbol_val($1,$3);}
                ;
exp             : term                   { $$ = $1; }
                | exp '+' term           { $$ = $1 + $3;}
                | exp '-' term           { $$ = $1 - $3;}
                ;
term            : number                 { $$ = $1; }
                | identifier             { $$ = symbol_val($1); }
                ;
%%



int compute_symbol_index(char token)
{
int index =-1;
if(isupper(token))
   index = token - 'A';
if(islower(token))
   index = token - 'a' + 26;
return index;
}

/*to return value of a symbol*/
int symbol_val(char symbol)
{
   int i;
   i = compute_symbol_index(symbol);
   return symbol_table[i];
}

/*to update sybol value*/
void update_symbol_val(char symbol, int val)
{
  int i;
  i= compute_symbol_index(symbol);
  symbol_table[i] = val;
}

int main(void)
{
/* to initialize symbol table*/
int i;

for(i=0;i<52;i++)
symbol_table[i] = 0;

return yyparse();
}
void yyerror(char *s)
{
   printf("Error msg: %s.", s);
}

/* Lex program for Symbol Table. Program Name: symbol_table.l */
%{
#include<stdlib.h>
#include "symbol_table.tab.h"
void yyerror(char *s);
%}

%%
"print"                 {return print;}
"done"                  {return done;}
[a-zA-Z]                {yylval.id = yytext[0];        return identifier;}
[0-9]+                  {yylval.num = atoi(yytext); return number;}
[ \t\n]                 ;
[-+=;]                  {return yytext[0];}
.                       {ECHO; yyerror("unexpected character");}
```

```
%%
int yywrap (void)
{
  return 1;
}
```

```
F:\Subjects_Material\Compiler Design\cdlab>yacc -d symbol_table.y

F:\Subjects_Material\Compiler Design\cdlab>flex symbol_table.l

F:\Subjects_Material\Compiler Design\cdlab>gcc lex.yy.c symbol_table.tab.c -o st


F:\Subjects_Material\Compiler Design\cdlab>st.exe
a=9;
B=5;
C=a+B;
print C;
14
C=B-2;
C=C+a;
print C;
12
done ;

F:\Subjects_Material\Compiler Design\cdlab>
```

11. Write a program to generate machine code for restricted programming expressions

```
/*
          Program to Generate Target code from Three-Address Code
*/

#include<stdio.h>
#include<conio.h>
#include<string.h>
#include<stdlib.h>
#include<ctype.h>

#define MAX 25

static int reg = 0;
int top = -1;
char stk[MAX];
char temp='1';
void getregister();
void pop();
void push(char);
void empty();

void main()
{
  int i;
  char postfix[10];
  printf("\nEnter Postfix Expression:");
  scanf("%s", postfix);
  i=0;
  while(postfix[i]!='\0')
  {
  while(islower(postfix[i])!=0)
  {
     push(postfix[i]);
     i++;
  }
  getregister();
  printf("\nLOAD %c R%d", stk[top-1], reg);
  switch(postfix[i])
  {
     case '+':
                    printf("\nADD %c R%d\n", stk[top], reg);
                    empty();
                    i++;
                    break;
     case '*':
                    printf("\nMUL %c R%d",stk[top], reg);
                    empty();
                    i++;
                    break;
     case '-':
                    printf("\nSUB %c R%d",stk[top], reg);
                    empty();
                    i++;
                    break;
     case '/':
                    printf("\nDIV %c R%d",stk[top], reg);
                    empty();
                    i++;
                    break;
     default:
                    printf("\nInvalid Instruction.");
                    getch();
                    exit(0);
  }
  }
  getch();
}

void empty()
{
   char t1;
   pop();
   pop();
   printf("\nSTR R%d T%c", reg,temp);
   t1 = temp;
   temp++;
   push(t1);
}

void getregister()
{
   reg++;
   if(reg>2)
   reg = 1;
}
void push(char x)
{
   if(top==MAX-1)
  printf("\nStack is Full.\n");
   else
   {
  top++;
  stk[top] = x;
   }
}
void pop()
{
   if(top == -1)
  printf("\nStack is Empty.\n");
   else
  top--;
}
```

12. Experiments on code optimization of programming expressions.

```
/*
Intermediate Code Representation using Three- Address Code(TAC).
Write a Program to Generate Intermediate code of Three-Address Code
for given expression

Input
Enter postfix expression:ab+cd-*ef*-
Postfix Expression:ab+cd-*ef*-

Intermediate Code(TAC)
_____________________________
 Z =  a  +  b
 Y =  c  -  d
 X =  Z  *  Y
 W =  e  *  f
 V =  X  -  W
*/


#include<stdio.h>
#include<string.h>
#include<stdlib.h>
#include<ctype.h>
void main()
{
 int i, size;
 char pfix[25], ch;
 void TAC(char *, int);

 printf("\nEnter postfix expression:");

 i=0;
 while((ch=getchar())!='\n')
 {

   pfix[i]=ch;
   i++;
 }
 pfix[i]='\0';
 printf("\nPostfix Expression:%s",pfix);

 size=i;
 TAC(pfix, size);
}/*main() close*/

void TAC(char pf[25], int s)
{
char stack[25], *str;
int top=0, i, j;
char temp;
char tac_arg1[25], tac_arg2[25], tac_op[25], tac_res[25];

j=-1;
temp=91;
for(i=0;i<s;i++)
{
if(isalpha(pf[i]))
stack[top++]=pf[i];
else if(pf[i]=='+' ||pf[i]=='-'||pf[i]=='/'||pf[i]=='*'||pf[i]=='=')
{
j = j + 1;
temp = temp - 1;
tac_arg2[j]=stack[--top];
tac_arg1[j]=stack[--top];
tac_op[j]=pf[i];
tac_res[j]=temp;
stack[top++]=temp;
}/*else if() close*/

}/*for() close*/

printf("\n\n\nIntermediate Code(TAC)");
printf("\n______________________________\n");
for(i=0;i<=j; i++)
printf("%2c = %2c %2c %2c\n",tac_res[i],tac_arg1[i],tac_op[i],tac_arg2[i]);
printf("\n______________________________\n");

} /*TAC() function close*/
```

Execution:

```
F:\Subjects_Material\Compiler Design\cdlab>gcc optimized_TAC.c
F:\Subjects_Material\Compiler Design\cdlab>a.exe
Enter postfix expression:abc*d+*
Postfix Expression:abc*d+*


Intermediate Code(TAC)
```

```
 Z =  b  *  c
 Y =  Z  +  d
 X =  a  *  Y


F:\Subjects_Material\Compiler Design\cdlab>
```