University of Engineering & Management, Kolkata
Department of Computer Science and Engineering
Compiler Design Laboratory
List of Experiments

Week 1

1. Write a C program to check if a user given string is a valid identifier or not?
2. Write a C program to check if a user given C program statement is a valid Comment or not?

Week 2

3. Write a C program to read a program written in a file and remove all comments. After removing all comments, rewrite the program in a separate file.
4. Write a C program to convert an infix statement into a postfix statement.

Week 3

5. Write a C program to evaluate an arithmetic expression which is given as a string. Consider the input has no parentheses and contains the following operators only: +, -, *, /
6. Write a Lex Program to count the number of vowels and consonants in a given string

Week 4

7. Write a Lex Program to count the number of characters, words, spaces, end of lines in a given input file.
8. Write a Lex Program to count no of: a) +ve and –ve integers b) +ve and –ve fractions

Week 5

9. Write a Lex Program to count the no of comment line in a given C program. Also eliminate them and copy that program into separate file.
10. Write a Lex Program to count the no of ‘scanf’ and ‘printf’ statements in a C program. Replace them with ‘readf’ and ‘writef’ statements respectively

Week 6

11. Write a Lex Program to recognize a valid arithmetic expression and identify the identifiers and operators present. Print them separately.
12. Write a Lex Program to recognize whether a given sentence is simple or compound.

Week 7

13. Write a Lex Program to recognize and count the number of identifiers in a given input file.
14. Write a YAAC Program to test the validity of a simple expression involving operators +, -, * and /

Week 8

15. Write a YAAC Program to recognize nested IF control statements and display the levels of nesting.

Week 9

16. Write a YAAC Program to check the syntax of a simple expression involving operators +, -, * and /
17. Write a YAAC Program to evaluate an arithmetic expression involving operating +, -, * and /

Week 10

18. Write a YAAC Program to recognize a valid variable, which starts with a letter, followed by any number of letters or digits.

Week 11

19. Write a YAAC Program to recognize strings 'aaab', 'abbb', 'ab' and 'a' using grammar (an b n , n>=0)

Week 12

20. Write a YAAC Program to recognize the grammar (an b, n>=10)

1. Write a C program to check if a user given string is a valid identifier or not?

Solution:

Program:

```
#include <stdio.h>
#include<string.h>

//Function to check for valid identifier
int isValid(char str[], int n)
{
        int i;
  // If first character is invalid
  if (!((str[0]>='a' && str[0]<='z')||(str[0]>='A' && str[0]<='Z')
      || str[0]=='_'))
     return 0;

  // Traverse the string for the rest of the characters
  for (i=1; i<n; i++)
        {
     if(!((str[i]>='a'&& str[i]<='z')||(str[i]>='A'&& str[i]<='Z')
         || (str[i]>='0' && str[i]<= '9')|| str[i] == '_'))
        return 0;
  }

  return 1;
}
int main()
{
  char str[30],ch;
  int length;
  printf("\nEnter an identifier:");
        gets(str);
  length=strlen(str);
  if (isValid(str,length)==1)
  {
        printf("\nValid Identifier");
  }
  else
        {
        printf("\nInValid Identifier");
  }


}
```

Input and Output:

```
Enter an identifier: first
Valid identifier
Enter an identifier:1aqw
Not a valid identifier
```

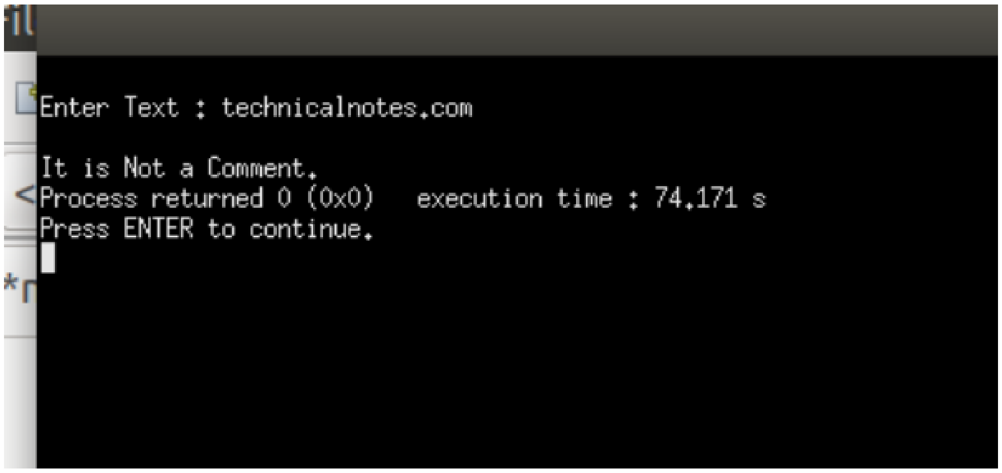
2. **Write a C program to check if a user given C program statement is a valid Comment or not?**

Solution:

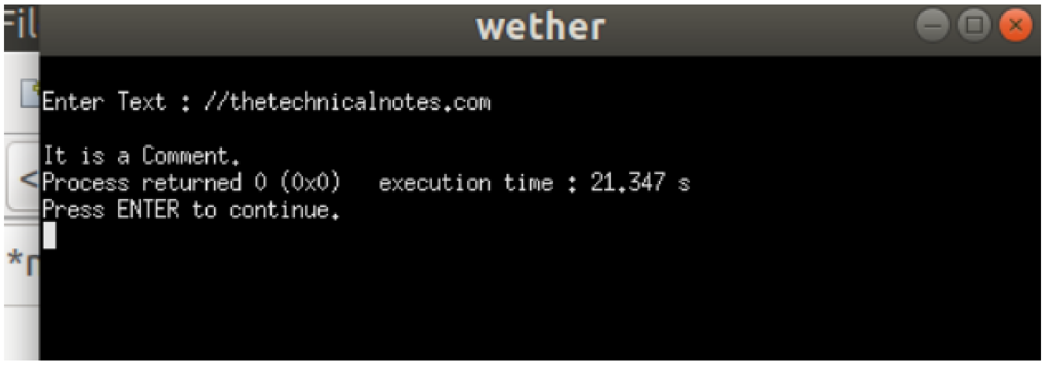
Program:

```
#include<stdio.h>
#include<string.h>

// Function to check if the given string is a comment or not
int isComment(char line[])
{

    // If two continuous slashes precedes the comment
    if (line[0] == '/' && line[1] == '/' && line[2] != '/')
            {
        return 1;
    }

    if(line[strlen(line)-2] == '*' && line[strlen(line)-1]=='/'
            && line[0]== '/' && line[1]== '*')
            {
        return 2;
    }

}

int main()
{
    char str[30];
    printf("Enter a line:\n");
    gets(str);
    if(isComment(str)==1)
    {
            printf("It is a single-line comment");
    }
    else if(isComment(str)==2)
    {
            printf("It is a multi-line comment");
    }
    else
    {
            printf("It is not a comment");
    }

    return 0;
}
```

Input and Output:



```
Enter Text : technicalnotes.com

It is Not a Comment.
Process returned 0 (0x0)   execution time : 74.171 s
Press ENTER to continue.
```



```
Enter Text : //thetechnicalnotes.com

It is a Comment.
Process returned 0 (0x0)   execution time : 21.347 s
Press ENTER to continue.
```

3. **Write a C program to read a program written in a file and remove all comments. After removing all comments, rewrite the program in a separate file?**

Solution:

Program:

```
#include <stdio.h>
#include <stdlib.h>
void check_comment (char) ;
void block_comment () ;
void single_comment () ;
FILE *fp1 , *fp2;
int main()
{
   char c;
   fp1 = fopen ("file1.txt","r") ;
   fp2 = fopen ("mynewfile.txt","w") ;
```

```
    while((c=fgetc(fp1))!=EOF)
        check_comment(c);

    fclose(fp1);
    fclose(fp2);

    return 0;
}
//handles both types of comments
void check_comment(char c)
{
    char d;

    if( c == '/')
            {
        if((d=fgetc(fp1))=='*')
         block_comment();

        else if( d == '/')
        {
          single_comment();

        }
        else
        {
            // if both the cases fail, it is not a comment, so we add the character as it is in the new file.
            fputc(c,fp2);
            fputc(d,fp2);
        }
    }

    // again, if all above fails, we add the character as it is in the new file.
    else
        fputc(c,fp2);
}
// function that handles block comments
void block_comment()
{

 char d,e;
```

```
    while((d=fgetc(fp1))!=EOF)
    {
    /* keep reading the characters and do nothing,
    as they do not have to be copied into the new file
    */
        if(d=='*')    // if the comment 'seems' like ending
        {
            e=fgetc(fp1);  // check if it actually ends (block comments end with '*/')

            if(e=='/')  // if the comment 'has' ended, return from the function
                return;
        }
    }

}
// function that handles single line comments
void single_comment()
{
 char d,e;

    while((d=fgetc(fp1))!=EOF)
    {
    /* keep reading the characters and do nothing,
    as they do not have to be copied into the new file
    */
        if(d=='\n')   // check if the comment ends
            return;  // if the comment 'has' ended, return from the function

    }

}
```

4. Write a C program to convert an infix statement into a postfix statement.

Solution:

Program:

```
#include<stdio.h>
#include<string.h>

int stack2[30],temp,length=0,indx=0,pos=0,top=-1;
char symbol,infix[20],postfix[20],stack[30];


/*----Function Prototypes-----*/
void push(char);
void push2(int);
char pop();
int pop2();
int precedence(char);
void infix_to_postfix(char[]);
void eval_postfix(char []);
/*--------------------------*/


void push(char symbol)          //push() starts
{
        top=top+1;
        stack[top]=symbol;
}       //end of push()

char pop()      //pop() starts
{
        temp=stack[top];
        top=top-1;
        return temp;
}       //end of pop()

int precedence(char symbol)
{
        int priority;
        switch(symbol)
        {
                case '#':
                                priority=0;
                                break;

                case '(':
                case ')':
                                priority=1;
                                break;

                case '+':
                case '-':
                                priority=2;
                                break;

                case '*':
                case '/':
                                priority=3;
                                break;
                case '^':
                                priority=4;
                                break;
        }//end of switch()
        return priority;
}//end of precedence()

/* Logic of Infix to postfix*/
void infix_to_postfix(char infix[])
{
        length=strlen(infix);
        push('#');
        while(indx<length)
        {
                symbol=infix[indx++];
                switch(symbol)
                {
                        case '(':
                                        push(symbol);
                                        break;
                        case ')':
```

```
                    temp=pop();
                    while(temp!='(')
                    {
                        postfix[pos++]=temp;
                        temp=pop();
                    }//end of while()
                    break;
            case '-':
            case '+':
            case '*':
            case '/':
            case '^':
                    while(precedence(stack[top])>=precedence(symbol))
                    {
                        temp=pop();
                        postfix[pos++]=temp;
                    }//end of while()
                    push(symbol);
                    break;
            default:
                    postfix[pos++]=symbol;
                    break;
        }//end of switch()
    }//end of while()

    while(top>0)
    {
        temp=pop();
        postfix[pos++]=temp;
        postfix[pos]='\0';

    }//end of while()
}//end of infix_to_postfix

void push2(int x)
{
   stack2[++top] = x;
}

int pop2()
```

```
{
   return stack2[top--];
}

void evaluate_postfix(char postfix[])
{
   char *temp;
   int n1,n2,n3,num;
   temp = postfix;
   while(*temp != '\0')
   {
      if(isdigit(*temp))
      {
         num = *temp - 48;
         push2(num);
      }
      else
      {
         n1 = pop2();
         n2 = pop2();
         switch(*temp)
         {
         case '+':
         {
            n3 = n1 + n2;
            break;
         }
         case '-':
         {
            n3 = n2 - n1;
            break;
         }
         case '*':
         {
            n3 = n1 * n2;
            break;
         }
         case '/':
         {
            n3 = n2 / n1;
```

```
                break;
            }
            }
            push2(n3);
        }
        temp++;
    }
    printf("\nThe result of expression=%d",pop2());

}
int main()        //main() starts
{
            printf("\nEnter an infix expression:\n");
            gets(infix);
            infix_to_postfix(infix);
            //printf("\nThe equivalent postfix expression:\n");
            //puts(postfix);
            evaluate_postfix(postfix);

return(0);
}           //end of main()
```

5. **Write a C program to evaluate an arithmetic expression which is given as a string. Consider the input has no parentheses and contains the following operators only: +, -, *, /**

   Solution:

   Program:

```
// C++ program to evaluate a given expression
#include <iostream>
using namespace std;

// A utility function to check if a given character is operand
bool isOperand(char c) { return (c >= '0' && c <= '9'); }

// utility function to find value of and operand
int value(char c) { return (c - '0'); }

// This function evaluates simple expressions. It returns -1 if the
// given expression is invalid.
```

```
int evaluate(char *exp)
{
        // Base Case: Given expression is empty
        if (*exp == '\0') return -1;

        // The first character must be an operand, find its value
        int res = value(exp[0]);

        // Traverse the remaining characters in pairs
        for (int i = 1; exp[i]; i += 2)
        {
                // The next character must be an operator, and
                // next to next an operand
                char opr = exp[i], opd = exp[i+1];

                // If next to next character is not an operand
                if (!isOperand(opd)) return -1;

                // Update result according to the operator
                if (opr == '+')    res += value(opd);
                else if (opr == '-') res -= value(opd);
                else if (opr == '*') res *= value(opd);
                else if (opr == '/') res /= value(opd);

                // If not a valid operator
                else                                    return -1;
        }
        return res;
}

// Driver program to test above function
int main()
{
        char expr1[] = "1+2*5+3";
        int res = evaluate(expr1);
        (res == -1)? cout << expr1 << " is " << "Invalid\n":
                        cout << "Value of " << expr1 << " is " << res << endl;

        char expr2[] = "1+2*3";
        res = evaluate(expr2);
        (res == -1)? cout << expr2 << " is " << "Invalid\n":
                              cout << "Value of " << expr2 << " is " << res << endl;

        char expr3[] = "4-2+6*3";
        res = evaluate(expr3);
        (res == -1)? cout << expr3 << " is " << "Invalid\n":
                              cout << "Value of " << expr3 << " is " << res << endl;

        char expr4[] = "1++2";
        res = evaluate(expr4);
        (res == -1)? cout << expr4 << " is " << "Invalid\n":
                              cout << "Value of " << expr4 << " is " << res << endl;
        return 0;
}
```

Input and Output:

```
Value of 1+2*5+3 is 18

Value of 1+2*3 is 9

Value of 4-2+6*3 is 24

1++2 is Invalid
```

6. **Write a Lex Program to count the number of vowels and consonants in a given string.**

Solution:

Program:

```
%{

#include<stdio.h>

int vowels=0;

int cons=0;

%}

%%

[aeiouAEIOU] {vowels++;}
```

```
[a-zA-Z] {cons++;}

%%

int yywrap()

{

return 1;

}

main()

{

printf("Enter the string.. at end press ^d\n");

yylex();

printf("No of vowels=%d\nNo of consonants=%d\n",vowels,cons);

}
```

7. **Write a Lex Program to count the number of characters, words, spaces, end of lines in a given input file.**

   Solution:

   Program:

```
%{
#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++;}
. {c++;}
%%
int yywrap()
{
return 1;
}
main(int argc, char *argv[])
{
```

```
If(argc!=2)
{
printf("Usage: <./a.out> <sourcefile>\n");
exit(0);
}
yyin=fopen(argv[1],"r");
yylex();
printf("No of characters=%d\nNo of words=%d\nNo of
spaces=%d\n No of lines=%d",c,w,s,l);
}
```

8. Write a Lex Program to count no of: a) +ve and –ve integers b) +ve and –ve fractions

Solution:

Program:

```
%{
#include<stdio.h>
int posint=0, negint=0,posfraction=0, negfraction=0;
%}
%%
[-][0-9]+ {negint++;}
[+]?[0-9]+ {posint++;}
[+]?[0-9]*\.[0-9]+ {posfraction++;}
[-][0-9]* \.[0-9]+ {negfraction++;}
%%
int yywrap()
{
return 1;
}
main(int argc, char *argv[])
{
If(argc!=2)
{
printf("Usage: <./a.out> <sourcefile>\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,
posfraction, negfraction);
}
```

9. **Write a Lex Program to count the no of comment lines in a given C program. Also eliminate them and copy that program into a separate file.**

Solution:

Program:

```
%{
#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!=2)
{
printf(“Usage: <./a.out> <sourcefile> <destn file>\n”);
exit(0);
}
yyin=fopen(argv[1],”r”);
yyout=fopen(argv[2],”w”);
yylex();
printf(“No of comment lines=%d\n”,com);
}
```

10. **Write a Lex Program to count the no of ‘scanf’ and ‘printf’ statements in a C program. Replace them with ‘readf’ and ‘writef’ statements respectively**

Solution:

Program:

```
%{
#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!=2)
{
printf(“Usage: <./a.out> <sourcefile> <destn file>\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);
}
```

11. Write a Lex Program to recognize a valid arithmetic expression and identify the identifiers and operators present. Print them separately.

Solution:

Program:

```
%{
#include<stdio.h>
#include<string.h>
int noprt=0, nopnd=0, valid=1, top=-1, m, 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 ^d\n”);
yylex();
if(valid==1 && i==-1 && (nopnd-noprt)==1)
{
printf(“The expression is valid\n”);
printf(“The operators are\n”);
for(k=0;k<l;k++)
Printf(“%s\n”,oprt[k]);
for(k=0;k<l;k++)
Printf(“%s\n”,opnd[k]);
}
else
```

```
Printf("The expression is invalid");
}
```

12. Write a Lex Program to recognize whether a given sentence is simple or compound.

Solution:

Program:

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

13. Write a Lex Program to recognize and count the number of identifiers in a given input file.

Solution:

Program:

```
%{
#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> <sourcefile>\n");
exit(0);
}
yyin=fopen(argv[1],"r");
printf("Valid identifires are\n");
yylex();
printf("No of identifiers = %d\n",id);
}
```

14. Write a YAAC Program to test the validity of a simple expression involving operators +, -, * and /

Solution:

Program:

Yacc Part

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

Lex Part

```
%{
#include "y.tab.h"
%}
%%
[0-9]+ { return DIGIT; }
[a-zA-Z][a-zA-Z0-9_]* { return ID; }
\n { return NL ;}
. { return yytext[0]; }
%%
```

15. Write a YAAC Program to recognize nested IF control statements and display the levels of nesting.

Solution:

Program:

Yacc Part

```
%token IF RELOP S NUMBER ID
%{
int count=0;
%}
%%
stmt : if_stmt { 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");
yyparse();
}
```

16. Write a YAAC Program to check the syntax of a simple expression involving operators +, -, * and /

Solution:

Program:

```
Yacc Part
%token NUMBER ID NL
%left '+' '-'
%left '*' '/'
%%
stmt : exp NL { printf("Valid Expression"); exit(0);}
;
exp : exp '+' exp
| exp '-' exp
| exp '*' exp
| exp '/' exp
| '(' exp ')'
| ID
| NUMBER
;
%%
int yyerror(char *msg)
{
printf("Invalid Expression\n");
exit(0);
}
main ()
```

```
{
printf("Enter the expression\n");
yyparse();
}
Lex Part
%{
#include "y.tab.h"
%}
%%
[0-9]+ { return NUMBER; }
[a-zA-Z][a-zA-Z0-9_]* { return ID; }
\n { return NL ;}
. { return yytext[0]; }
%%
```

17. Write a YAAC Program to evaluate an arithmetic expression involving operating +, -, * and /

Solution:

Program:

```
Yacc Part
%token NUMBER ID NL
%left '+' '-'
%left '*' '/'
%%
stmt : exp NL { printf("Value = %d\n",$1); exit(0);}
;
exp : exp '+' exp { $$=$1+$3; }
| exp '-' exp { $$=$1-$3; }
| exp '*' exp { $$=$1*$3; }
| exp '/' exp { if($3==0)
{
printf("Cannot divide by 0");
exit(0);
}
else
$$=$1/$3;
}
| '(' exp ')' { $$=$2; }
| ID { $$=$1; }
| NUMBER { $$=$1; }
;
```

```
%%
int yyerror(char *msg)
{
printf("Invalid Expression\n");
exit(0);
}
main ()
{
printf("Enter the 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]; }
%%
```

18. Write a YAAC Program to recognize a valid variable, which starts with a letter, followed by any number of letters or digits.

Solution:

Program:

```
Yacc Part
%token DIGIT LETTER NL UND
%%
stmt : variable NL { printf("Valid Identifiers\n"); exit(0);}
;
variable : LETTER alphanumeric
;
alphanumeric: LETTER alphanumeric
| DIGIT alphanumeric
| UND alphanumeric
| LETTER
| DIGIT
| UND
;
%%
```

```
int yyerror(char *msg)
{
printf("Invalid Expression\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 ; }
[\n] { return NL ;}
[_] { return UND; }
. { return yytext[0]; }
%%
```

19. Write a YAAC Program to recognize strings 'aaab', 'abbb', 'ab' and 'a' using grammar (an b n , n>=0)

Solution:

Program:

```
Yacc Part
%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]; }
%%
```

20. Write a YAAC Program to recognize the grammar (an b, n>=10)

Solution:

Program:

```
%token A B NL
%%
stmt : A A A A A A A A A A s B NL
{
Printf("Valid"); 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]; }
%%
```

Steps to Execute Lex Program:

```
lex <pgm name>
cc lex.yy.c –ll
./a.out
```

Steps to execute YACC program:

```
yacc –d <yacc_pgm name>
lex <lex_pgm_name>
cc y.tab.c lex.yy.c –ly –ll
./a.out
```