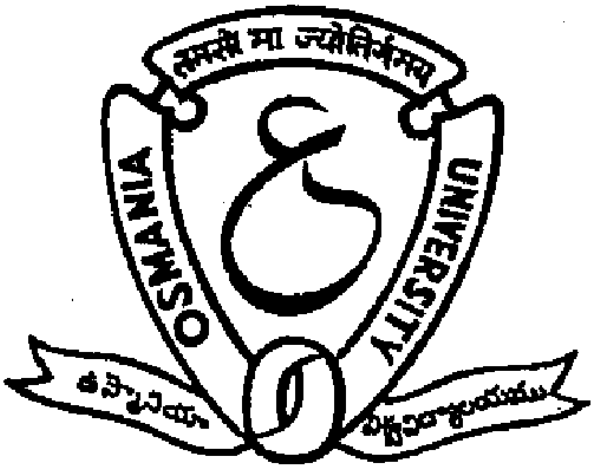
UNIVERSITY COLLEGE OF ENGINEERING (A)

Osmania University, Hyderabad – 500 007

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING



CERTIFICATE

This is to certify that _____ bearing roll no: **1005177330__** studying B.E 4/4 1st semester has successfully completed **COMPILER CONSTRUCTIONS LAB** for the academic year 2020-2021.

INTERNAL EXAMINER EXTERNAL EXAMINER

TABLE OF CONTENTS

1) LEX program to print all numbers in a file

```
%{
#include <stdio.h>
%}
%%
[0-9]+ { printf("%s\n", yytext); }
.|\n     ;
%%
main()
{
       yylex();
}
```

Input:

There are 73 members in the class.

Output: 73

2) LEX program to print all HTML tags in a file

```
%{
#include <stdio.h>
%}
%%
"<"[^>]*> { printf("VALUE: %s\n", yytext); }
.|\n    ;
%%
main()
{
      yylex();
}
```

Input:

```
<html>
<body>
this is html document.
</body>
</html>
```

Output:

```
<html>
<body>
</body>
</html>
```

3) LEX program to do word count of wc command in UNIX

```
%{
int charcount=0,linecount=0;
%}
%option noyywrap
%%

. charcount++;
\n {charcount++,linecount++;}

%%

int main()
{
        yylex();
        printf("There were %d characters in %d lines\n",charcount,linecount);
        return 0;
}
```

Input:
A lexer to do word count
function of the wc
command in UNIX

Output:
59 characters and 13 words in 3 lines

4) LEX program to classify tokens as words

```
%{
   int tokenCount=0;
%}
%%
[a-zA-Z]+ { printf("%d WORD \"%s\"\n", ++tokenCount, yytext); }
[0-9]+    { printf("%d NUMBER \"%s\"\n", ++tokenCount, yytext); }
[^a-zA-Z0-9]+   { printf("%d OTHER \"%s\"\n", ++tokenCount, yytext); }
%%

main()
{
   yylex();
}
```

Input:
Hello! World ... this is 21st century

Output:
1 WORD Hello
2 OTHER !
3 WORD World
4 OTHER …
5 WORD this
6 WORD is
7 NUMBER 21
8 WORD st century

5) LEX program to find factorial of a number

```
%{
#include <stdio.h>
int count =1;

%}
%%
[0-9]+ {count= factorial(a to i (yytext));
        printf("input is %s\n", yytext);
        printf("output is %d",count);}

%%
main()
{
        yylex();
}

int factorial(int fact)
{
if(fact ==1)
return 1;
else
return fact*factorial(fact-1);
}
```

Input: 6

Output: 720

6) Scanner Program Using LEX

```
%{
        #include<stdio.h>
        int lineno=1;
%}
%option noyywrap
%%
"/n" lineno++;
Int | float | char | if | else | break | switch | continue | case | while | do | for
{printf("%s keyword,length %d,lineno %d\n",yytext,yyleng,lineno);}
[a-zA-Z]([a-zA-Z|0-9])* {printf("%s identifier,length %d,lineno %d\n",yytext,yyleng,lineno);}
";" {printf("; SEMI");}
">=" {printf(">= GTE");}
"<>" {printf(">= NTE");}
"<" {printf(">= LT");}
"<=" {printf(">= LTE");}
">" {printf(">= GT");}
"{"
"}"
"(" {printf("LEFT PARAN");}
")" {printf("RIGHT PARAN");}
"++" {printf("UNARY PLUS");}
"__" {printf("UNARY MINUS");}
"+" {printf("%s plus", yytext);}
"_" {printf( " - MINUS");}
"*" {printf(" * mul");}
"%" {printf("%  mod");}
"/" {printf("/ divide");}
%%
main(int argc,char *argv[])
{
        FILE * fd;
        fd=fopen(argv[1],"r");
        yyin=fd;
        yylex();
        return 0;
}
```

Input:

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

Output:

```
#include identifier, length 7, lineno 1
<= LT stdio identifier,length 5,lineno 1
```

.h identifier, length 1,lineno 1
>= GT

7) Recursive Decent Parser

```
/*  for grammar
   E->x+T
   T->(E)
   T->x
*/

#include<iostream.h>
#include<conio.h>
int p=0;

 int match(char c,char *s)
 {
        p++;
        if(c==s[p-1])
                return 1;
        else
                return 0;
 }
 int E(char *s);

 int T(char *s)
 {
        int i=0;
        switch(s[p])
        {
                case '(':
                        i=match('(',s) && E(s) && match(')',s);
                        cout<<"t"<<i<<endl;
                        break;
                case 'x':
                        i=match('x',s);
                        break;
        }
        cout<<i<<endl;
        return i;
 }

int E(char * s)
 {
    int i=match('x',s) && match('+',s) && T(s);
    cout<<i<<"\n";
    return i;
 }
```

```
void main()
{
        char s[50]="x+(x+x)";
        if(E(s))
                cout<<"accepted";
        else
                cout<<"rejected";
        getch();
  }
```

Output: accepted

8) Write a C Program to check Ambiguity in the given Grammar.

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

void main()
{
        int i,j,k,l,n=1;
        char a[20];
        printf("enter production:\n");
        gets(a);
        l=strlen(a);
        for(i=3;i<=l;i++)
        {
            for(j=i+1;j<=l;j++)
            {
                 for(k=65;k<=90;k++)
                 {
                      if(a[i]==k)
                           {
                             if(a[j]==a[i])
                             n++;
                           }
                 }
            }
        }
        if(n>1)
        printf("Ambiguity");
        else
        printf("No Ambiguity");
}
```

Output:

```
enter production:
E->E+E|E*E
Ambiguity.

enter production:
S->Sb|b
No Ambiguity.
```

9) Program to check if given grammar is left factored

```
#include<stdio.h>
#include<string.h>
char alpha[20]={0};
char beta[20]={0};
char grammar[30]={0};
int i=0,j=0,k=0;
char c;

void leftfactoring()
{
        int flag=0;
        for(i=0;grammar[i]!='\0';i++)
        if(grammar[i]=='>')
        break;
        c=grammar[i+1];
        for(i=i+1;grammar[i]!='\0';i++)
        {
                if(grammar[i]==c)
                 {
                    for(i=i+1;grammar[i]!='|'&&grammar[i]!='\0';i++)
                    {
                        flag=1;
                        beta[k++]=grammar[i];
                    }
                    if(flag==0)
                    beta[k++]=238;
                    beta[k++]=';';
                    flag=0;
                }

                else
                {
                    while(grammar[i]!='\0'&&grammar[i]!='|')
                    alpha[j++]=grammar[i++];
                    alpha[j++]=';';
                }

        }

        alpha[j]='\0';
        beta[k]='\0';
}
```

```
int main()
{
	printf("\nEnter grammar\n");
	scanf("%s",grammar);
	leftfactoring();
	printf("the grammar after left factoring:\n");

	if(strlen(alpha)==0)
	printf("%c->%cX",grammar[0],c);

	else
	printf("%c->%cx|",grammar[0],c);
	for(i=0;alpha[i+1]!='\0';i++)
	{
		if(alpha[i]==';')
		printf("|");
		else
		printf("%c",alpha[i]);
	}
	printf("\nX->");
	for(i=0;beta[i+1]!='\0';i++)
	{
		if(beta[i]==';')
		printf("|");
		else
		printf("%c",beta[i]);
	}

	return 0;

}
```

Input:
S → Sa

Output:
S → SX
X → a

10) Program to check for terminals and non terminal in a grammar

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

char terminals[30]={0};
char nonterminals[30]={0};
char grammar[30]={0};
int i=0,j=0,k=-1,l=0;

void identify()
{
	if(((grammar[0]<65)||(grammar[0]>90))||(grammar[1]!=45)||(grammar[2]!=62))
		printf("\nInvalid!\n");
	else
	{
		while((k<30)&&(grammar[++k]!=0))
		{
			if((grammar[k]>=65)&&(grammar[k]<=90))
			{
				printf("\nNon-terminal: %c",grammar[k]);
			}
			else if (grammar[k]==124);
			else if (k>2)
			{
				printf("\nTerminal: %c",grammar[k]);
			}
		}
	}
}

int main()
{
	while(1)
	{
		printf("\nEnter a production of the grammar\n");
		scanf("%s",grammar);
		identify();
		i=0;
		j=0;
		k=-1;
		l=0;
	}
}
```

Input:

S → a

Output:

Non-terminals : S
Terminals : a

11) YACC Calculator

```
%{#include<stdio.h>
#include<ctype.h>
%}
%token INTEGER
%%
command : expr { printf("%d \n",$1);}
;
expr : expr'+' term { $$=$1 + $3; }
    | expr'-' term { $$=$1 - $3; }
    | term { $$=$1; }
    ;
term : term '*' factor {$$ =$1 + $3; }
    |factor {$$=$1;}
    ;
factor : INTEGER { $$ = $1;}
      | '('expr')' { $$=$2;}
      ;
%%
main()
{
return yyparse();
}
int yylex(void)
{
int c;
while ((c=getchar())== ' ');
if (isdigit(c))
{ ungetc(c,stdin);
scanf ("%d",&yylval);
return (INTEGER);
}
if(c== '\n')
return 0;
return (c);
}
void yyerror(char *s)
{
fprintf (stderr, "%s \n ",s);
}
```

Input: 4*6
Output: 24

12) Program on Code Generation

```
import java.io.*;
class ExpNode
{
	char element;
	ExpNode left;
	ExpNode right;

	public ExpNode(char c)
	 {
		element = c;
		left = null;
		right = null;
	}

	public ExpNode(char c,ExpNode l, ExpNode r)
		{
		element = c;
		left = l;
		right = r;
	}
}

class CodeGen
{
	public static void main(String args[])throws IOException
	{
		System.out.println("Enter a simple expression :");
		BufferedReader br = new BufferedReader(new InputStreamReader(System.in));
		String input = br.readLine();
		ExpNode SyntaxTree = getExpTree(input);
		System.out.println("\nP-Code for the expression is: \n");
		genCode(SyntaxTree);
	}

	public static void genCode(ExpNode t)
	{
		if(t!=null)
		{
			switch(t.element)
			{
				case '+':
					genCode(t.left);
					genCode(t.right);
```

```
                        System.out.println("adi");
                        break;
                case '-':
                        genCode(t.left);
                        genCode(t.right);
                        System.out.println("sbi");
                        break;
                case '*':
                        genCode(t.left);
                        genCode(t.right);
                        System.out.println("mpi");
                        break;
                default:
                        if(Character.isDigit(t.element))
                                System.out.println("ldc" + t.element);
                }
        }
}
public static ExpNode getExpTree(String input)
{
        if(input.length()>1)
        {
                int plus = input.indexOf('+');
                int minus = input.indexOf('-');
                int i;
                if(plus>=0 && minus>=0)
                        i = ((plus<minus)?plus:minus);
                else
                        i = ((plus>minus)?plus:minus);
                if(i!=-1)
                {
                        String Left  = input.substring(0,i-1);
                        ExpNode LeftChild;
                        int index = Left.indexOf('*');

                        if(index!=(-1))
                                LeftChild = getMultChild(input.substring(0,i));
                        else
                                LeftChild = new ExpNode(input.charAt(i-1),null,null);

                        ExpNode NextTerm = getExpTree(input.substring(i+1));

                        return (new ExpNode(input.charAt(i),LeftChild,NextTerm));
                }
                else
                        return getMultChild(input)
```

```
}
        else
                return (new ExpNode(input.charAt(0),null,null));
 }

public static ExpNode getMultChild(String input)
{
        ExpNode Tree = null;
        ExpNode Left = null;
        while(input.length()>1)
        {
                int index = input.indexOf('*');
                int lastindex = input.lastIndexOf('*');
                if(index==lastindex)
                {
                        if(Left==null)
                                Left = new ExpNode(input.charAt(index-1),null,null);
                        int r = input.length()-1;
                        ExpNode Right = new ExpNode(input.charAt(r),null,null);
                        Tree = new ExpNode('*',Left,Right);
                        break;
                }
                Else
                {
                        if(Left==null)
                                Left = new ExpNode(input.charAt(index-1),null,null);
                        else
                                ExpNode temp = new ExpNode('*',Left,new
                        ExpNode(input.charAt(index+1),null,null));
                        Left = temp;
                }
                input = input.substring(index+1);
        }
        return Tree;
}
```

Output:

```
Enter a simple expression
a+b
P-code for the expression is :
Adi
```