Program:

//Dasarada Ram Reddy - 160114733092

//Program to implement Recursive descent parser for the given grammar.

rdp.c

```
#include<stdio.h>
int i=0;
char a[10];
void e();
void e1();
void t();
void t1();
void f();
int main()
{
	while(1)
	{
		printf("\nEnter  a string :\n");
		scanf("%s",&a);
		if(strcmp(a,"bye"))
		{
			e();
			if(a[i]=='$')
				printf("Successful parse\n");
			else
				printf("Unsucessful parse\n");
		}
		else
		break;
	}
	return 0;
}
void e(void)
{
	t();
	e1();
}
void e1(void)
{
	if(a[i]=='+')
	{
		i++;
		t();
```

```
                e1();
        }
}

void t(void)
{
        f();
        t1();
}
void t1(void)
{
        if(a[i]=='*')
        {
                i++;
                f();
                t1();
        }
}
void f(void)
{
        if(a[i]=='(')
        {
                i++;
                e();
                if(a[i]==')')
                i++;
        }
        else if(a[i]=='i')
                i++;
}
```

Testing:

Input:

```
$(a+b)
```

Expected Output:

```
Enter a string :
$(a+b)
Successful parse
```

Enter a string :
$hello
Unsuccessful parse

Actual Output:

```
dasarada@dasarada-Inspiron-5558:~/dachi/3-2/cc$ ./a.out

Enter  a string :
$(a+b)
Successful parse

Enter  a string :
$hello
Successful parse
```

Result:

Successfully executed the program.

Program:

//Dasarada Ram Reddy - 160114733092
//Program to count the no of comment line in a given C program. Also eliminate them and copy that program into separate file using Lex

```
%{
#include<stdio.h>
#include<string.h>
FILE *out;
char line[50];
int count=0;
%}

%%
"/*"[-_a-zA-Z \n]+"*/"  count++;
"//"[^\n]+"\n"          count++;
['\n'] { fprintf(out,"%s\n",line);}
(.*) {strcpy(line,yytext);}
%%

int yywrap()
{
printf("%d comments\n",count);
return 1;
}
int main()
{
        yyin=fopen("in.txt","r");
        out=fopen("cout.txt","w");
        yylex();
}
```

Testing:

Input:

in.txt

```
#include<stdio.h>//Header file
int main()//Start of execution
{
        int a;//Declaration
```

```
        scanf("%d",a);//Input
        printf("Hello CBIT");//Output}
```

Expected Output:

cout.txt

```
#include<stdio.h>
int main()
{
        int a;
        scanf("%d",a);
        printf("Hello CBIT");
}
```

Actual Output:

```
dasarada@dasarada-Inspiron-5558:~/dachi/3-2/cc$ flex 9comments.l
dasarada@dasarada-Inspiron-5558:~/dachi/3-2/cc$ gcc lex.yy.c
dasarada@dasarada-Inspiron-5558:~/dachi/3-2/cc$ ./a.out
5 comments
```

cout.txt:

```
#include<stdio.h>//Header file
int main()//Start of execution
{
        int a;//Declaration
        scanf("%d",a);//Input
        printf("Hello CBIT");//Output
}
```

Result:

Successfully executed the program.

Program:

```
//Dasarada Ram Reddy – 160114733092
//program to count no: of  +ve and -ve integers and +ve and -ve fractions
%{
        #include<stdio.h>
        #include<string.h>
        int i=0,ip=0,in=0,fp=0,fn=0,flag=0;
        %}

        %%

        [+]?[0-9]+ {ip++;}
        [-][0-9]+  {in++;}
        [+]?[0-9]*[.][0-9.]+   { for(i=0;i<yyleng;i++)
                                 {if(yytext[i]=='.')
                                    flag++;}
                                 if(flag==1)
                                    fp++;
                                    flag=0;
                               }

        [-][0-9]*[.][0-9.]+    { for(i=0;i<yyleng;i++)
                                 {if(yytext[i]=='.')
                                    flag++;}
                                 if(flag==1)
                                    fn++;
                                    flag=0;
                               }

        %%

        int yywrap()
        {
                printf("%d positive integers\n",ip);
                printf("%d negative integers\n",in);
                printf("%d positive fractions\n",fp);
                printf("%d negative fractions\n",fn);
                return 1;
        }
        int main()
        {
                yyin=fopen("f.txt","r");
```

```
        yylex();
}
```

Testing:

Input:

file.txt

```
1 +12 4.4.4
-1242 12.214
15 +14.214
-134.143 24.24 2523
```

Expected Output:

4 positive integers
1 negative integers
3 positive fractions
1 negative fractions

Actual Output:

```
dasarada@dasarada-Inspiron-5558:~/dachi/3-2/cc$ flex 8positive_negative.l
dasarada@dasarada-Inspiron-5558:~/dachi/3-2/cc$ gcc lex.yy.c -ll
dasarada@dasarada-Inspiron-5558:~/dachi/3-2/cc$ ./a.out




4 positive integers
1 negative integers
3 positive fractions
1 negative fractions
```

Result:

Successfully executed the program.

Program:
//Dasarada Ram Reddy – 160114733092
//Program to count the no of ‘scanf’ and ‘printf’ statements in a C program. Replace them with ‘readf’ and ‘writef’ statements respectively using lex.

```
%{
#include<stdio.h>
#include<string.h>
FILE *out;
char line[50];
int count=0;
%}

%%

"printf" yytext="writef";
"scanf" yytext="readf";
['\n'] { fprintf(out,"%s\n",line);}
(.*) {strcpy(line,yytext);}

%%

int yywrap()
{
	return 1;
}
int main()
{
	yyin=fopen("in.txt","r");
	out=fopen("psout.txt","w");
	yylex();
}
```

Testing:

Input:
//in.txt

```
#include<stdio.h>//Header file
int main()//Start of execution
{
	int a;//Declaration
	scanf("%d",a);//Input
```

```
        printf("Hello CBIT");//Output}
```

Expected Output:

psout.txt

```
#include<stdio.h>//Header file
int main()//Start of execution
{
        int a;//Declaration
        readf("%d",a);//Input
        writef("Hello CBIT");//Output
}
```

Actual Output:

psout.txt

```
include<stdio.h>/*Header file*/
int main()//Start of execution
{
        int a;//Declaration
        readf("%d",a);//Input
        writef("Hello CBIT");//Output
}
```

Result:

Successfully executed the program.

Program:
//Dasarada Ram Reddy – 160114733092
//Program to recognize nested IF control statements and display the levels of nesting.

nif.l

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

nif.y

```
%{
#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("the statement is invalid\n");
```

```
exit(0);
}

main()
{
printf("enter the statement\n");
yyparse();
}
```

Testing:

Input:

```
if(a>b){s}
```

Expected Output:

```
enter a statement
if(a>b){s}
No. of nested if statements=1
```

Actual Output:

```
dasarada@dasarada-Inspiron-5558:~/dachi/3-2/cc$ flex nif.l
dasarada@dasarada-Inspiron-5558:~/dachi/3-2/cc$ yacc -d nif.y
dasarada@dasarada-Inspiron-5558:~/dachi/3-2/cc$ gcc lex.yy.c y.tab.c -ll
dasarada@dasarada-Inspiron-5558:~/dachi/3-2/cc$ ./a.out
enter the statement
if(a>b){s}
No. of nested if        statements=1
```

Result:

Successfully executed the program.

Program:
//Dasarada Ram Reddy – 160114733092
//Program to recognize strings ‘aaab’, ‘abbb’, ‘ab’ and ‘a’ using grammar ($a^n b^n$, n>=0)

anbn.l

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

%%
[aA] {return A;}
[bB] {return B;}
\n {return NL;}
. {return yytext[0];}
%%
```

anbn.y

```
%{
#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();
}
```

Testing:

Input:

```
ab
aabb
aaabb
```

Expected Output:

```
enter a string
ab
valid string
```

Actual Output:

```
dasarada@dasarada-Inspiron-5558:~/dachi/3-2/cc$ flex anbn.l
dasarada@dasarada-Inspiron-5558:~/dachi/3-2/cc$ yacc -d anbn.y
dasarada@dasarada-Inspiron-5558:~/dachi/3-2/cc$ gcc lex.yy.c y.tab.c -ll
dasarada@dasarada-Inspiron-5558:~/dachi/3-2/cc$ ./a.out
enter the string
ab
valid string
dasarada@dasarada-Inspiron-5558:~/dachi/3-2/cc$ ./a.out
enter the string
aabb
valid string
dasarada@dasarada-Inspiron-5558:~/dachi/3-2/cc$ ./a.out
enter the string
aaabb
invalid string
```

Result:

Successfully executed the program.

Program:

//Dasarada Ram Reddy - 160114733092

//Program to find FIRST elements for the given grammar.

first.c

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

int main()
{
        int i,n,j,k;
        char str[10][10],f;
        printf("Enter the number of productions\n");
        scanf("%d",&n);
        printf("Enter grammar\n");
        for(i=0;i<n;i++)
        scanf("%s",&str[i]);
        for(i=0;i<n;i++)
        {
            f= str[i][0];
            int temp=i;
            if(isupper(str[i][3]))
            {
                repeat:
                for(k=0;k<n;k++)
                {
                    if(str[k][0]==str[i][3])
                    {
                        if(isupper(str[k][3]))
                        {
                            i=k;
                            goto repeat;
                        }
                        else
                            printf("First(%c)=%c\n",f,str[k][3]);
                        }
                }
            }
            else
                printf("First(%c)=%c\n",f,str[i][3]);
            i=temp;
        }
}
```

Testing:

Input:

S->AB
A->a
B->b

Expected Output:

Enter the number of productions
3
Enter grammar
S->AB
A->a
B->b
First(S)=a
First(A)=a
First(B)=b

Actual Output:

```
dasarada@dasarada-Inspiron-5558:~/dachi/3-2/cc/FIRSTFOLLOWS$ ./a.out
Enter the number of productions
3
Enter grammar
S->AB
A->a
B->b
First(S)=a
First(A)=a
First(B)=b
```

Result:

Successfully executed the program.

Program:

//Dasarada Ram Reddy – 160114733092

//Program to find FOLLOW elements for the given grammar.

follow.c

```
#include<stdio.h>
#include<string.h>
main()
{
 int np,i,j,k;
 char prods[10][10],follow[10][10],Imad[10][10];
 printf("enter no. of productions\n");
 scanf("%d",&np);
 printf("enter grammar\n");
 for(i=0;i<np;i++)
  scanf("%s",&prods[i]);
 for(i=0; i<np; i++)
 {
 if(i==0)
  printf("Follow(%c) = $\n",prods[0][0]);//Rule1
 for(j=3;prods[i][j]!='\0';j++)
 {
  int temp2=j;
  //Rule-2: production A->xBb then everything in first(b) is in follow(B)
  if(prods[i][j] >= 'A' && prods[i][j] <= 'Z')
  {
   if((strlen(prods[i])-1)==j)
        printf("Follow(%c) = Follow(%c)\n",prods[i][j],prods[i][0]);
   int temp=i;
   char f=prods[i][j];
   if(!isupper(prods[i][j+1])&&(prods[i][j+1]!='\0'))
   printf("Follow(%c) = %c\n",f,prods[i][j+1]);
   if(isupper(prods[i][j+1]))
   {
        repeat:
         for(k=0;k<np;k++)
         {
          if(prods[k][0]==prods[i][j+1])
          {
           if(!isupper(prods[k][3]))
                printf("Follow(%c) = %c\n",f,prods[k][3]);
           else
           {
```

```
                    i=k;
                    j=2;
                    goto repeat;
                }
               }
              }
             }i=temp;
          } j=temp2;
        }
       }
      }
```

Testing:

Input:

S->AB
A->a
B->b

Expected Output:

enter no. of productions
3
enter grammar
S->AB
A->a
B->b
Follow(S) = $
Follow(A)=b
Follow(B)=Follow(S)

Actual Output:

```
dasarada@dasarada-Inspiron-5558:~/dachi/3-2/cc/FIRSTFOLLOWS$ ./a.out
enter no. of productions
3
enter grammar
S->AB
A->a
B->b
Follow(S) = $
Follow(A)=b
Follow(B)=Follow(S)
```

Result:
Successfully executed the program.

Program:
//Dasarada Ram Reddy – 160114733092
//Program to construct predictive LL1 parsing table.

ll1.c

```
#include<stdio.h>
#include<conio.h>
#include<string.h>
#include<process.h>
char prod[10][20],start[2];
char nonterm[10],term[10];
char input[10],stack[50];
int table[10][10];
int te,nte;
int n;
void main()
{
	clrscr();
	init();
	parse();
	getch();
}
init()
{
	int i,j;
	/*The terminals should be entered in single lower case
	letters,special symbol and non-terminals should be entered in
	single upper case letters extends to symbol is '->' and epsilon
	symbol is '@'*/
	printf("\nEnter the no. of terminals:");
	scanf("%d",&te);
	for(i=0;i<te;i++)
	{
		fflush(stdin);
		printf("Enter the terminal %d:",i+1);
		scanf("%c",&term[i]);
	}
	term[i]='$';
	printf("\nEnter the no. of non terminals:");
	scanf("%d",&nte);
	for(i=0;i<nte;i++)
	{
```

```
		fflush(stdin);
		printf("Enter the non-terminal %d:",i+1);
		scanf("%c",&nonterm[i]);
	}
	printf("\nEnter the no. of productions:");
	scanf("%d",&n);
	for(i=0;i<n;i++)
	{
		printf("Enter the production %d:",i+1);
		scanf("%s",prod[i]);
	}
	fflush(stdin);
	printf("\nEnter the start symbol:");
	scanf("%c",&start[0]);
	printf("\nEnter the input string:");
	scanf("%s",input);
	input[strlen(input)]='$';
	printf("\n\nThe productions are:");
	printf("\nProductionNo.	Production");
	for(i=0;i<n;i++)
		printf("\n %d			%s",i+1,prod[i]);
	printf("\n\nEnter the parsing table:");
	printf("\n Enter the production number in the required entry as
	mentioned above.");
	printf("\n Enter the undefined entry or error of table as '0'\n\n");
	for(i=0;i<nte;i++)
	{
		for(j=0;j<=te;j++)
		{
			fflush(stdin);
			printf("Entry of table[%c,%c]:",nonterm[i],term[j]);
			scanf("%d",&table[i][j]);
		}
	}
}
parse()
{
	int i,j,prodno;
	int top=-1,current=0;
	stack[++top]='$';
	stack[++top]=start[0];
	do
	{
```

```
                if((stack[top]==input[current])&&(input[current]=='$'))
                {
                        printf("\nThe given input string is parsed");
                        getch();
                        exit(0);
                }
                else if(stack[top]==input[current])
                {
                        top--;
                        current++;
                }
                else if(stack[top]>='A'&&stack[top]<='Z')
                {
                        for(i=0;i<nte;i++)
                                if(nonterm[i]==stack[top]) break;
                        for(j=0;j<=te;j++)
                                if(term[j]==input[current]) break;
                        prodno=table[i][j];
                        if(prodno==0)
                        {
                                printf("\nThe given input string is not parsed");
                                getch();
                                exit(0);
                        }
                        else
                        {
                                for(i=strlen(prod[prodno-1])-1;i>=3;i--)
                                {
                                        if(prod[prodno-1][i]!='@')
                                        stack[top++]=prod[prodno-1][i];
                                }
                                top--;
                        }
                }
                else
                {
                        printf("\nThe given input string is not parsed");
                        getch();
                        exit(0);
                }
        }while(1);
}
```

Testing:

Input:

```
Enter the no. of terminals:2
Enter the terminal 1:a
Enter the terminal 2:b

Enter the no. of non terminals:3
Enter the non-terminal 1:S
Enter the non-terminal 2:A
Enter the non-terminal 3:B

Enter the no. of productions:6
Enter the production 1:S->aAB
Enter the production 2:S->@
Enter the production 3:A->aA
Enter the production 4:A->@
Enter the production 5:B->bB
Enter the production 6:B->@

Enter the parsing table:
Entry of table[S,a]:1
Entry of table[S,b]:2
Entry of table[S,$]:3
Entry of table[A,a]:4
Entry of table[A,b]:5
Entry of table[A,$]:5
Entry of table[B,a]:0
Entry of table[B,b]:6
Entry of table[B,$]:7
```

Expected Output:

```
The productions are:
Production No.    Production
      1             S->aAB
      2             S->@
      3             A->aA
      4             A->@
      5             B->bB
```

6 B->@

The given input string is parsed

Actual Output:

```
Enter the no. of terminals:2
Enter the terminal 1:a
Enter the terminal 2:b

Enter the no. of non terminals:3
Enter the non-terminal 1:S
Enter the non-terminal 2:A
Enter the non-terminal 3:B

Enter the no. of productions:6
Enter the production 1:S->aAB
Enter the production 2:S->@
Enter the production 3:A->aA
Enter the production 4:A->@
Enter the production 5:B->bB
Enter the production 6:B->@

Enter the start symbol:S

Enter the input string:ab$


The productions are:
ProductionNo.   Production
 1              S->aAB
 2              S->@
 3              A->aA
 4              A->@
 5              B->bB
 6              B->@

Enter the parsing table:
 Enter the production number in the required entry
 as mentioned above.
 Enter the undefined entry or error of table as '0'

Entry of table[S,a]:1
Entry of table[S,b]:0
Entry of table[S,$]:2
Entry of table[A,a]:3
Entry of table[A,b]:4
Entry of table[A,$]:0
Entry of table[B,a]:0
Entry of table[B,b]:5
Entry of table[B,$]:6

The given input string is parsed
```

Result:

Successfully executed the program.