Department of Computer Science Engineering

PREFACE

This book entitled **"Compiler Design Lab Manual"** is intended for the use of First semester (i.e, III-I) B.Tech (CSE) students of Marri Laxman Reddy Institute of Technology and Management, Dundigal, Hyderabad. The main objective of the Compiler Design Lab is to intended to make the students experiment on the basic techniques of compiler construction and tools that can used to perform syntax-directed translation of a high-level programming language into an executable code. Students will design and implement language processors in C by using tools to automate parts of the implementation process. This will provide deeper insights into the more advanced semantics aspects of programming languages, code generation, machine independent optimizations, dynamic memory allocation, and object orientation.

By
Ch Sravani ,
Assistant Professor,
CSE Department.

Department of Computer Science Engineering

Institute Vision and Mission

Institute Vision:

To be as an ideal academic institution by graduating talented engineers to be ethically strong, competent with quality research and technologies.

Institute Mission:

- Utilize rigorous educational experiences to produce talented engineers.
- Create an atmosphere that facilitates the success of students.
- Programs that integrate global awareness, communication skills and Leadership qualities.
- Education and Research partnership with institutions and industries to prepare the students for interdisciplinary research.

Department of Computer Science Engineering

Department Vision and Mission

Department Vision:

To empower the students to be technologically adept, innovative, self-motivated and responsible global citizen possessing human values and contribute significantly towards high quality technical education with ever changing world.

Department Mission:

- To offer high-quality education in the computing fields by providing an environment where the knowledge is gained and applied to participate in research, for both students and faculty.
- To develop the problem solving skills in the students to be ready to deal with cutting edge technologies of the industry.
- To make the students and faculty excel in their professional fields by inculcating the communication skills, leadership skills, team building skills with the organization of various co-curricular and extra-curricular programmes.
- To provide the students with theoretical and applied knowledge, and adopt an education approach that promotes lifelong learning and ethical growth.

Department of Computer Science Engineering

Program Educational Objectivies:

PEO1: Establish a successful professional career in industry, government or academia.

PEO2: Gain multidisciplinary knowledge providing a sustainable competitive edge in higher studies or Research.

PEO3: Promote design, analyze, and exhibit of products, through strong communication, leadership and ethical skills, to succeed an entrepreneurial.

Program Outcomes

The Program Outcomes (POs) of the department are defined in a way that the Graduate Attributes are included, which can be seen in the Program Outcomes (POs) defined. The Program Outcomes (POs) of the department are as stated below:

a : An ability to apply knowledge of Science, Mathematics, Engineering & Computing fundamentals for the solutions of Complex Engineering problems.

b : An ability to identify, formulates, research literature and analyze complex engineering problems using first principles of mathematics and engineering sciences.

c : An ability to design solutions to complex process or program to meet desired needs.

d : Ability to use research-based knowledge and research methods including design of experiments to provide valid conclusions.

e : An ability to use appropriate techniques, skills and tools necessary for computing practice.

f : Ability to apply reasoning informed by the contextual knowledge to assess social issues, consequences & responsibilities relevant to the professional engineering practice.

g : Ability to understand the impact of engineering solutions in a global, economic, environmental, and societal context with sustainability.

h : An understanding of professional, ethical, Social issues and responsibilities.

i : An ability to function as an individual, and as a member or leader in diverse teams and in multidisciplinary settings.

j : An ability to communicate effectively on complex engineering activities within the engineering community.

k : Ability to demonstrate and understanding of the engineering and management principles as a member and leader in a team.

l : Ability to engage in independent and lifelong learning in the context of technological change.

Program Specific Outcomes

PSO1: Applications of Computing: Ability to use knowledge in various domains to provide solution to new ideas and innovations.

PSO2: Programming Skills: Identify required data structures, design suitable algorithms, develop and maintain software for real world problems.

PROGRAM OUTCOMES	
PO1	**Engineering knowledge**: Apply the knowledge of mathematics, science, engineering fundamentals, and an engineering specialization to the solution of complex engineering problems.
PO2	**Problem analysis**: Identify, formulate, review research literature, and analyze complex engineering problems reaching substantiated conclusions using first principles of mathematics, natural sciences, and engineering sciences.
PO3	**Design/development of solutions**: Design solutions for complex engineering problems and design system components or processes that meet the specified needs with appropriate consideration for the public health and safety, and the cultural, societal, and environmental considerations.
PO4	**Conduct investigations of complex problems**: Use research-based knowledge and research methods including design of experiments, analysis and interpretation of data, and synthesis of the information to provide valid
PO5	**Modern tool usage**: Create, select, and apply appropriate techniques, resources, and modern engineering and IT tools including prediction and modeling to complex engineering activities with an understanding of the
PO6	**The engineer and society**: Apply reasoning informed by the contextual knowledge to assess societal, health, safety, legal and cultural issues and the consequent responsibilities relevant to the professional engineering practice.
PO7	**Environment and sustainability**: Understand the impact of the professional engineering solutions in societal and environmental contexts, and demonstrate the knowledge of, and need for sustainable development.

PO8	**Ethics**: Apply ethical principles and commit to professional ethics and responsibilities and norms of the engineering practice.
PO9	**Individual and team work**: Function effectively as an individual, and as a member or leader in diverse teams, and in multidisciplinary settings.
PO10	**Communication**: Communicate effectively on complex engineering activities with the engineering community and with society at large, such as, being able to comprehend and write effective reports and design documentation, make effective presentations, and give and receive clear instructions.
PO11	**Project management and finance**: Demonstrate knowledge and understanding of the engineering and management principles and apply these to one's own work, as a member and leader in a team, to manage projects and
PO12	**Life-long learning**: Recognize the need for, and have the preparation and ability to engage in independent and life-long learning in the broadest context of technological change.
PROGRAM SPECIFIC OUTCOMES	
PSO1	**Professional Skills:** The ability to research, understand and implement computer programs in the areas related to algorithms, system software, multimedia, web design, big data analytics, and networking for efficient analysis and design of computer-based systems of varying complexity.
PSO2	**Problem-Solving Skills:** The ability to apply standard practices and strategies in software project development using open-ended programming environments to deliver a quality product for business success.
PSO3	**Successful Career and Entrepreneurship:** The ability to employ modern computer languages, environments, and platforms in creating innovative career paths, to be an entrepreneur, and a zest for higher studies.

COMPILER DESIGN LAB SYLLABUS

S. No.	List of Experiments	Page No.
1	Write a LEX Program to scan reserved word & Identifiers of C Language language.	
2	Implement Predictive Parsing algorithm	
3	Write a C program to generate three address code.	
4	Implement SLR(1) Parsing algorithm	
5	Design LALR bottom up parser for the given language	
	ADDITIONAL PROGRAMS	
6	Write a C program for implementing the functionalities of predictive parser for the mini language specified in Note 1.	
7	a) *Write a C program for constructing of LL (1) parsing. b) *Write a C program for constructing recursive descent parsing.	

*Content beyond the University prescribed syllabi

Department of Computer Science Engineering

COURSE OBJECTIVE

1. To provide hands-on experience on web technologies
2. To develop client-server application using web technologies
3. To introduce server-side programming with Java servlets and JSP
4. To understand the various phases in the design of a compiler.
5. To understand the design of top-down and bottom-up parsers.
6. To understand syntax directed translation schemes.
7. To introduce lex and yacc tools.

OUTCOMES

1. Design and develop interactive and dynamic web applications using HTML, CSS, JavaScript and XML

2. Apply client-server principles to develop scalable and enterprise web applications.

3. Ability to design, develop, and implement a compiler for any language.

4. Able to use lex and yacc tools for developing a scanner and a parser.

5. Able to design and implement LL and LR parsers

WEEK-1

OBJECTIVE:

Implement the lexical analyzer using JLex, flex or other lexical analyzer generating tools.

AIM:

To analyze JLex, flex or other lexical analyzer generating tools.

RESOURCE:

- Linux using Putty

PROGRAM LOGIC:

- Read the input string.
- Check whether the string is identifier/ keyword /symbol by using the rules of identifier and keywords using LEX Tool

PROCEDURE:

- Go to terminal
- Open vi editor,
- Compile using Lex lex.l , cc lex.yy.c , ./a.out

PROGRAM:

```
/* program name is lexp.l */
%{
 /* program to recognize a c program */
  int COMMENT=0;
%}
identifier [a-zA-Z][a-zA-Z0-9]*
%%
#.* { printf("\n%s is a PREPROCESSOR DIRECTIVE",yytext);}
int | float | char | double | while | for | do | if | break | continue | void | switch |
case | long | struct | const | typedef | return |else |
goto{printf("\n\t%s is a KEYWORD",yytext);}
"/*" {COMMENT = 1;}
```

```
 /*{printf("\n\n\t%s is a COMMENT\n",yytext);}*/
"*/" {COMMENT = 0;}


 /* printf("\n\n\t%s is a COMMENT\n",yytext);}*/
{identifier}\( {if(!COMMENT)printf("\n\nFUNCTION\n\t%s",yytext);}
\{ {if(!COMMENT) printf("\n BLOCK BEGINS");}
\} {if(!COMMENT) printf("\n BLOCK ENDS");}
{identifier}(\[[0-9]*\])? {if(!COMMENT) printf("\n %s IDENTIFIER",yytext);}
\".*\" {if(!COMMENT) printf("\n\t%s is a STRING",yytext);}
[0-9]+ {if(!COMMENT) printf("\n\t%s is a NUMBER",yytext);}
\)(\;)? {if(!COMMENT) printf("\n\t");ECHO;printf("\n");}
\(
\(  ECHO;
= {if(!COMMENT)printf("\n\t%s is an ASSIGNMENT OPERATOR",yytext);}
\<= | \>= | \< | == |
\> {if(!COMMENT) printf("\n\t%s is a RELATIONAL OPERATOR",yytext);}
%%
int main(int argc,char **argv)
{
 if (argc > 1)
 {
 FILE *file;
 file = fopen(argv[1],"r");
 if(!file)
{
 printf("could not open %s \n",argv[1]);
exit(0);
}
 yyin = file;
 }
 yylex();
 printf("\n\n");
 return 0;
```

```
}
int yywrap()
{
 return 0;
}
```

INPUT & OUTPUT:

Input

$vi var.c

```
#include<stdio.h> main()
{
int a,b;
}
```

Output

$lex lex.l

$cc lex.yy.c

$./a.out var.c

#include<stdio.h> is a PREPROCESSOR DIRECTIVE FUNCTION

main ( )

BLOCK BEGINS

int is a KEYWORD

a IDENTIFIER

b IDENTIFIER

BLOCK ENDS

PRE LAB QUESTIONS:

1. List the different sections available in LEX compiler?
2. What is an auxiliary definition?
3. How can we define the translation rules?
4. What is regular expression?
5. What is finite automaton?

POST LAB QUESTIONS:

1. What is Jlex?
2. What is Flex?
3. What is lexical analyzer generator?
4. What is the input for LEX Compiler?
5. What is the output of LEX compiler?

WEEK-2

OBJECTIVE:

Write a C program for implementing the functionalities of predictive parser .

Understanding the functionalities of predictive parser for a given language.

RESOURCE:

- Turbo C++

PROGRAM LOGIC:

- Read the input string.
- By using the FIRST AND FOLLOW values.
- Verify the FIRST of non terminal and insert the production in the FIRST value
- If we have any @ terms in FIRST then insert the productions in FOLLOW values
- Constructing the predictive parser table

PROCEDURE:

- Go to debug -> run or press CTRL + F9 to run the program.

PROGRAM:

```
#include<stdio.h>
#include<conio.h>
#include<string.h>
char prol[7][10]={"s","A","A","B","B","C","C"};
char pror[7][10]={"Aa","Bb","Cd","aB","@","Cc","@"};
char prod[7][10]={"s-->A","A-->Bb","A-->Cd","B-->aB","B-->@","C-->Cc","C-->@"};
char first[7][10]={"abcd","ab",cd","a@","@","c@","@"};
char follow[7][10]={"$","$","$","a$","b$","c$","d$"};
```

```
char table[5][6][10];
{
switch(c)
{
case 'S':return0;
case 'A':return1;

case 'B':return2;
case 'C':return3;
case 'a':return0;
case 'b':return1;
case 'c':return2;
case 'd':return3;
case '$':return4;
}
retun(2);
}
void main()
{
int i,j,k;
clrscr();
for(i=0;i<5;i++)
for(j=0;j<6;j++)
strcpy(table[i][j]," ");
printf("\n The following is the predictive parsing table for the following grammar:\n");
for(i=0;i<7;i++)
printf("%s\n",prod[i]);
printf("\n Predictive parsing table is:\n ");
fflush(stdin);
for(i=0;i<7;i++)
{
k=strlen(first[i]);
for(j=0;j<10;j++)
if(first[i][j]!='@')
```

```
strcpy(table[numr(prol[i][0])+1][numr(first[i][j])+1],prod[i]);
}
for(i=0;i<7;i++)
{
if(strlen(pror[i])==1)
{
if(pror[i][0]=='@')
{
k=strlen(follow[i]);

for(j=0;j<k;j++)
strcpy(table[numr(prol[i][0])+1][numr(follow[i][j])+1]prod[i]);
}
}
}
strcpy(table[0][0]," ");
strcpy(table[0][1],"a");
strcpy(table[0][2],"b");
strcpy(table[0][3],"c");
strcpy(table[0][4],"d");
strcpy(table[0][5],"$");
strcpy(table[1][0],"S");
strcpy(table[2][0],"A");
strcpy(table[3][0],"B");
strcpy(table[4][0],"C");
printf("\n--------------------------------------------------------------------------\n");
for(i-0;i<5;i++)
for(j=0;j<6;j++)
{
printf("%s_10S",table[i][j]);
if(j==5)
printf("\n--------------------------------------------------------------------------\n");
}
getch();
```

}

INPUT & OUTPUT:

The following is the predictive parsing table for the following grammar:

S->A
A->Bb
A->Cd
B->aB
B->@
C->Cc
C->@

Predictive parsing table is

	a	b	c	d	$
S	S->A	S->A	S->A	S->A	
A	A->Bb	A->Bb	A->Cd	A->Cd	
B	B->aB	B->@	B->@		B->@
C			C->@	C->@	C->@

PRE LAB QUESTIONS:

1. What is top-down parsing?
2. What are the disadvantages of brute force method?
3. What is context free grammar?
4. What is parse tree?
5. What is ambiguous grammar?

6. What are the derivation methods to generate a string for the given grammar?
7. What is the output of parse tree?

POST LAB QUESTIONS

1. What is Predictive parser?
2. How many types of analysis can we do using Parser?
3. What is Recursive Decent Parser?
4. How many types of Parsers are there?
5. What is LR Parser?

WEEK-3

OBJECTIVE: Write a C program to generate three address code.

RESOURCE:

Turbo C++

ALGORITHM:

Step1: Begin the program

Step2 : The expression is read from the file using a file pointer

Step3 : Each string is read and the total no. of strings in the file is calculated.

Step4: Each string is compared with an operator; if any operator is seen then the previous string and next string are concatenated and stored in a first temporary value and the three address code expression is printed

Step5 : Suppose if another operand is seen then the first temporary value is concatenated to the next string using the operator and the expression is printed.

Step6 : The final temporary value is replaced to the left operand value.

Step7 : End the program

PROGRAM:

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

struct three
```

```
{
char data[10],temp[7];

}s[30];
void main()
{
char d1[7],d2[7]="t";
int i=0,j=1,len=0;
FILE *f1,*f2;
clrscr();
f1=fopen("sum.txt","r");
f2=fopen("out.txt","w");
while(fscanf(f1,"%s",s[len].data)!=EOF)
len++;
itoa(j,d1,7);
strcat(d2,d1);
strcpy(s[j].temp,d2);
strcpy(d1,"");
strcpy(d2,"t");
if(!strcmp(s[3].data,"+"))
{
fprintf(f2,"%s=%s+%s",s[j].temp,s[i+2].data,s[i+4].data);
j++;
}
else if(!strcmp(s[3].data,"-"))
{
fprintf(f2,"%s=%s-%s",s[j].temp,s[i+2].data,s[i+4].data);
j++;
}
for(i=4;i<len-2;i+=2)
{
itoa(j,d1,7);
strcat(d2,d1);
strcpy(s[j].temp,d2);
if(!strcmp(s[i+1].data,"+"))
fprintf(f2,"\n%s=%s+%s",s[j].temp,s[j-1].temp,s[i+2].data);
else if(!strcmp(s[i+1].data,"-"))
fprintf(f2,"\n%s=%s-%s",s[j].temp,s[j-1].temp,s[i+2].data);
strcpy(d1,"");
strcpy(d2,"t");
j++;
}
fprintf(f2,"\n%s=%s",s[0].data,s[j-1].temp);
```

```
fclose(f1);
fclose(f2);
getch();


}
```

Input: sum.txt

out = in1 + in2 + in3 - in4

Output : out.txt

```
t1=in1+in2
t2=t1+in3
t3=t2-in4
out=t3
```

RESULT:

Thus a C program to generate a three address code for a given expression is written, executed and the output is verified.

WEEK-4

OBJECTIVE: Implement SLR(1) Parsing algorithm

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

int i,j,k,m,n=0,o,p,ns=0,tn=0,rr=0,ch=0;
char cread[15][10],gl[15],gr[15][10],temp,templ[15],tempr[15][10],*ptr,temp2[5];
char dfa[15][10];

struct states
{
   char lhs[15],rhs[15][10];
   int n;//state number
}I[15];

int compstruct(struct states s1,struct states s2)
{
   int t;
if(s1.n!=s2.n)
       return 0;
if( strcmp(s1.lhs,s2.lhs)!=0 )
       return 0;
   for(t=0;t<s1.n;t++)
if( strcmp(s1.rhs[t],s2.rhs[t])!=0 )
           return 0;
   return 1;
}

void moreprod()
{
   int r,s,t,l1=0,rr1=0;
   char *ptr1,read1[15][10];

   for(r=0;r<I[ns].n;r++)
   {
      ptr1=strchr(I[ns].rhs[l1],'.');
      t=ptr1-I[ns].rhs[l1];
if( t+1==strlen(I[ns].rhs[l1]) )
       {
```

```
            l1++;
            continue;
        }
        temp=I[ns].rhs[l1][t+1];
        l1++;
        for(s=0;s<rr1;s++)
if( temp==read1[s][0] )
                break;
        if(s==rr1)
        {
            read1[rr1][0]=temp;
            rr1++;
        }
        else
            continue;

        for(s=0;s<n;s++)
        {
            if(gl[s]==temp)
            {
                I[ns].rhs[I[ns].n][0]='.';
                I[ns].rhs[I[ns].n][1]='\0';
strcat(I[ns].rhs[I[ns].n],gr[s]);
                I[ns].lhs[I[ns].n]=gl[s];
                I[ns].lhs[I[ns].n+1]='\0';
                I[ns].n++;
            }
        }
    }
}

void canonical(int l)
{
    int t1;
    char read1[15][10],rr1=0,*ptr1;
    for(i=0;i<I[l].n;i++)
    {
        temp2[0]='.';
        ptr1=strchr(I[l].rhs[i],'.');
        t1=ptr1-I[l].rhs[i];
if( t1+1==strlen(I[l].rhs[i]) )
            continue;

        temp2[1]=I[l].rhs[i][t1+1];
        temp2[2]='\0';

        for(j=0;j<rr1;j++)
```

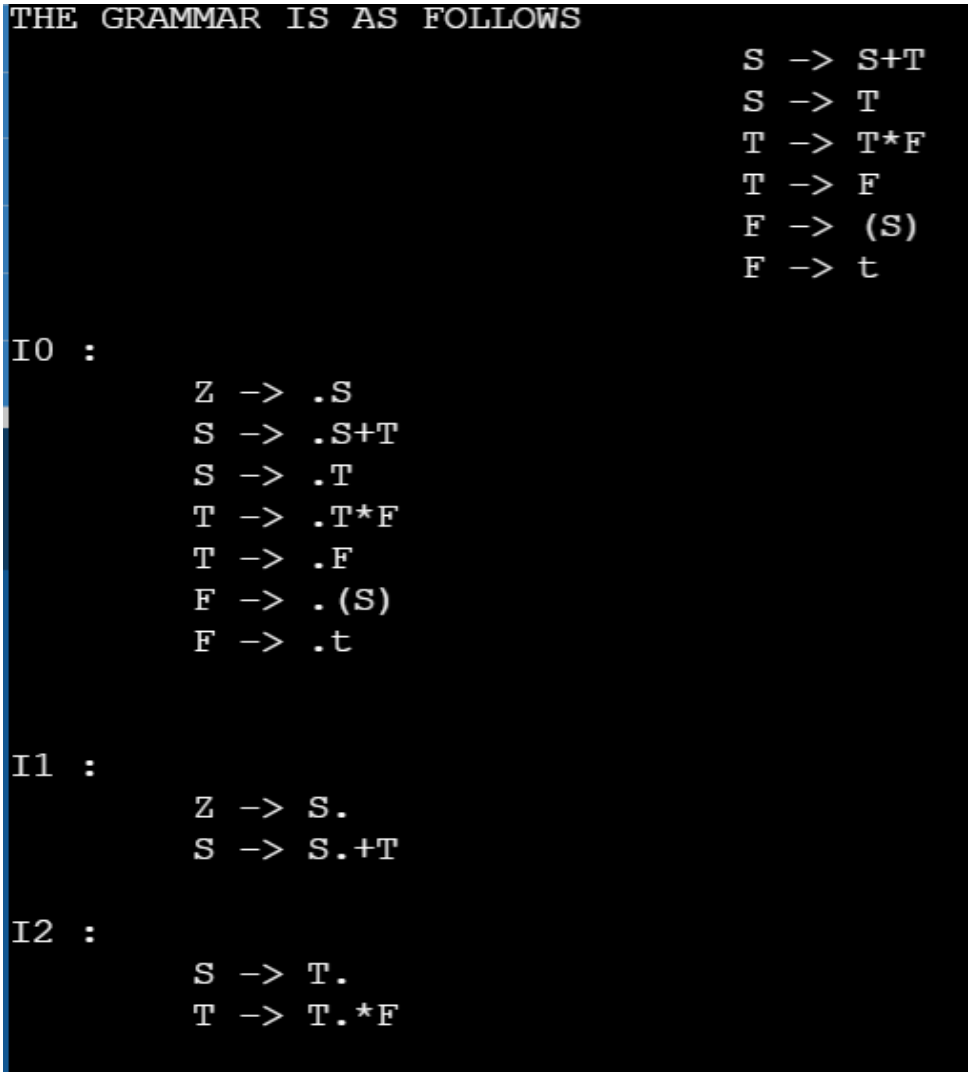
```
if( strcmp(temp2,read1[j])==0 )
            break;
      if(j==rr1)
      {
strcpy(read1[rr1],temp2);
         read1[rr1][2]='\0';
         rr1++;
      }
      else
         continue;

      for(j=0;j<I[0].n;j++)
      {
ptr=strstr(I[l].rhs[j],temp2);
if( ptr )
         {
templ[tn]=I[l].lhs[j];
templ[tn+1]='\0';
strcpy(tempr[tn],I[l].rhs[j]);
tn++;
         }
      }

      for(j=0;j<tn;j++)
      {
ptr=strchr(tempr[j],'.');
         p=ptr-tempr[j];
tempr[j][p]=tempr[j][p+1];
tempr[j][p+1]='.';
         I[ns].lhs[I[ns].n]=templ[j];
         I[ns].lhs[I[ns].n+1]='\0';
strcpy(I[ns].rhs[I[ns].n],tempr[j]);
         I[ns].n++;
      }

moreprod();
      for(j=0;j<ns;j++)
      {
         //if ( memcmp(&I[ns],&I[j],sizeof(struct states))==1 )
if( compstruct(I[ns],I[j])==1 )
         {
            I[ns].lhs[0]='\0';
            for(k=0;k<I[ns].n;k++)
               I[ns].rhs[k][0]='\0';
            I[ns].n=0;
dfa[l][j]=temp2[1];
            break;
```

```
            }
        }
        if(j<ns)
        {
tn=0;
            for(j=0;j<15;j++)
            {
templ[j]='\0';
tempr[j][0]='\0';
            }
            continue;
        }

dfa[l][j]=temp2[1];
printf("\n\nI%d :",ns);
        for(j=0;j<I[ns].n;j++)
printf("\n\t%c -> %s",I[ns].lhs[j],I[ns].rhs[j]);
        //getch();
        ns++;
tn=0;
        for(j=0;j<15;j++)
        {
templ[j]='\0';
tempr[j][0]='\0';
        }
    }
}

void main()
{
    FILE *f;
    int l;
    //clrscr();

    for(i=0;i<15;i++)
    {
        I[i].n=0;
        I[i].lhs[0]='\0';
        I[i].rhs[0][0]='\0';
dfa[i][0]= '\0';
    }

    f=fopen("tab6.txt","r");
    while(!feof(f))
    {
fscanf(f,"%c",&gl[n]);
fscanf(f,"%s\n",gr[n]);
```

```
        n++;
    }

printf("THE GRAMMAR IS AS FOLLOWS\n");
    for(i=0;i<n;i++)
printf("\t\t\t\t%c -> %s\n",gl[i],gr[i]);

    I[0].lhs[0]='Z';
strcpy(I[0].rhs[0],".S");
I[0].n++;
    l=0;
    for(i=0;i<n;i++)
    {
        temp=I[0].rhs[l][1];
        l++;
        for(j=0;j<rr;j++)
if( temp==cread[j][0] )
                break;
        if(j==rr)
        {
cread[rr][0]=temp;
rr++;
        }
        else
            continue;
        for(j=0;j<n;j++)
        {
            if(gl[j]==temp)
            {
                I[0].rhs[I[0].n][0]='.';
strcat(I[0].rhs[I[0].n],gr[j]);
                I[0].lhs[I[0].n]=gl[j];
I[0].n++;
            }
        }
    }
    ns++;

printf("\nI%d :\n",ns-1);
    for(i=0;i<I[0].n;i++)
printf("\t%c -> %s\n",I[0].lhs[i],I[0].rhs[i]);

    for(l=0;l<ns;l++)
        canonical(l);

printf("\n\n\t\tPRESS ANY KEY FOR TABLE");
    //getch();
```

```
    //clrscr();

printf("\t\t\t\nDFA TABLE IS AS FOLLOWS\n\n\n");
    for(i=0;i<ns;i++)
    {
printf("I%d : ",i);
       for(j=0;j<ns;j++)
          if(dfa[i][j]!='\0')
printf("'%c'->I%d | ",dfa[i][j],j);
printf("\n\n\n");
    }
printf("\n\n\n\t\tPRESS ANY KEY TO EXIT");
    //getch();
}
```

3 OUTPUT



4

```
I3 :
        T -> F.

I4 :
        F -> (.S)
        S -> .S+T
        S -> .T
        T -> .T*F
        T -> .F
        F -> .(S)
        F -> .t

I5 :
        F -> t.

I6 :
        S -> S+.T
        T -> .T*F
        T -> .F
        F -> .(S)
        F -> .t

I7 :
        T -> T*.F
        F -> .(S)
        F -> .t
```

5

```
I8 :
      F -> (S.)
      S -> S.+T

I9 :
      S -> S+T.
      T -> T.*F

I10 :
      T -> T*F.

I11 :
      F -> (S).

              PRESS ANY KEY FOR TABLE
DFA TABLE IS AS FOLLOWS


I0 : 'S'->I1 | 'T'->I2 | 'F'->I3 | '('->I4 | 't'->I5 |


I1 : '+'->I6 |


I2 : '*'->I7 |
```

6

```
I3 :

I4 : 'T'->I2 | 'F'->I3 | '('->I4 | 't'->I5 | 'S'->I8 |

I5 :

I6 : 'F'->I3 | '('->I4 | 't'->I5 | 'T'->I9 |

I7 : '('->I4 | 't'->I5 | 'F'->I10 |

I8 : 'F'->I0 | '+'->I6 | ')'->I11 |

I9 : ')'->I1 | '*'->I7 |

I10 :

I11 :
```

```
PRESS ANY KEY TO EXIT
```

WEEK-5

AIM:C program to Design LALR Bottom up Parser

PROGRAM

```
#include<stdio.h>

#include<conio.h>

#include<stdlib.h>

#include<string.h>

void push(char *,int *,char);

char stacktop(char *);

void isproduct(char,char);

int ister(char);

int isnter(char);

int isstate(char);

void error();

void isreduce(char,char);

char pop(char *,int *);

void printt(char *,int *,char [],int);

void rep(char [],int);

struct action

{

char row[6][5];

};
```

```
const struct action A[12]={
{"sf","emp","emp","se","emp","emp"},
{"emp","sg","emp","emp","emp","acc"},
{"emp","rc","sh","emp","rc","rc"},
{"emp","re","re","emp","re","re"},
{"sf","emp","emp","se","emp","emp"},
{"emp","rg","rg","emp","rg","rg"},
{"sf","emp","emp","se","emp","emp"},
{"sf","emp","emp","se","emp","emp"},
{"emp","sg","emp","emp","sl","emp"},
{"emp","rb","sh","emp","rb","rb"},
{"emp","rb","rd","emp","rd","rd"},
{"emp","rf","rf","emp","rf","rf"}
};
struct gotol
{
char r[3][4];
};
const struct gotol G[12]={
{"b","c","d"},
{"emp","emp","emp"},
{"emp","emp","emp"},
```

```
{"emp","emp","emp"},

{"i","c","d"},

{"emp","emp","emp"},

{"emp","j","d"},

{"emp","emp","k"},

{"emp","emp","emp"},

{"emp","emp","emp"},

};

char ter[6]={'i','+','*',')','(','$'};

char nter[3]={'E','T','F'};

char states[12]={'a','b','c','d','e','f','g','h','m','j','k','l'};

char stack[100];

int top=-1;

char temp[10];

struct grammar

{
char left;

char right[5];

};

const struct grammar rl[6]={

{'E',"e+T"},

{'E',"T"},

{'T',"T*F"},
```

```
{'T',"F"},

{'F',"(E)"},

{'F',"i"},

};

void main()

{

char inp[80],x,p,dl[80],y,bl='a';

int i=0,j,k,l,n,m,c,len;


printf(" Enter the input :");

scanf("%s",inp);

len=strlen(inp);

inp[len]='$';

inp[len+1]='\0';

push(stack,&top,bl);

printf("\n stack \t\t\t input");

printt(stack,&top,inp,i);

do

{

x=inp[i];

p=stacktop(stack);


isproduct(x,p);

if(strcmp(temp,"emp")==0)
```

```
error();
if(strcmp(temp,"acc")==0)
break;
else
{
if(temp[0]=='s')
{
push(stack,&top,inp[i]);
push(stack,&top,temp[1]);
i++;
}
else
{
if(temp[0]=='r')
{
j=isstate(temp[1]);
strcpy(temp,rl[j-2].right);
dl[0]=rl[j-2].left;
dl[1]='\0';
n=strlen(temp);
for(k=0;k<2*n;k++)
pop(stack,&top);
for(m=0;dl[m]!='\0';m++)
```

```
push(stack,&top,dl[m]);
l=top;
y=stack[l-1];
isreduce(y,dl[0]);
for(m=0;temp[m]!='\0';m++)
push(stack,&top,temp[m]);
}
}
}
printt(stack,&top,inp,i);
}while(inp[i]!='\0');
if(strcmp(temp,"acc")==0)
printf(" \n accept the input ");
else
printf(" \n do not accept the input ");
getch();
}
void push(char *s,int *sp,char item)
{
if(*sp==100)
printf(" stack is full ");
else
```

```
{

*sp=*sp+1;
s[*sp]=item;

}

}

char stacktop(char *s)

{

char i;

i=s[top];

return i;

}

void isproduct(char x,char p)

{

int k,l;

k=ister(x);

l=isstate(p);

strcpy(temp,A[l-1].row[k-1]);

}

int ister(char x)

{

int i;

for(i=0;i<6;i++)

if(x==ter[i])
```

```
return i+1;

return 0;

}

int isnter(char x)

{

int i;

for(i=0;i<3;i++)

if(x==nter[i])

return i+1;

return 0;

}

int isstate(char p)

{

int i;

for(i=0;i<12;i++)

if(p==states[i])
return i+1;

return 0;

}

void error()

{

printf(" error in the input ");

exit(0);
```

```
}
void isreduce(char x,char p)
{
int k,l;
k=isstate(x);
l=isnter(p);
strcpy(temp,G[k-1].r[l-1]);
}

char pop(char *s,int *sp)
{
char item;
if(*sp==-1)
printf(" stack is empty ");
else
{
item=s[*sp];
*sp=*sp-1;
}
return item;
}
void printt(char *t,int *p,charinp[],int i)
{
```

```
int r;

printf("\n");

for(r=0;r<=*p;r++)

rep(t,r);

printf("\t\t\t");

for(r=i;inp[r]!='\0';r++)
printf("%c",inp[r]);

}

void rep(char t[],int r)

{

char c;

c=t[r];

switch(c)

{

case 'a': printf("0");

break;

case 'b': printf("1");

break;

case 'c': printf("2");

break;

case 'd': printf("3");

break;

case 'e': printf("4");
```

```
break;
case 'f': printf("5");
break;
case 'g': printf("6");
break;
case 'h': printf("7");
break;
case 'm': printf("8");
break;
case 'j': printf("9");
break;
case 'k': printf("10");
break;
case 'l': printf("11");
break;
default :printf("%c",t[r]);
break;
}
}
```

OUTPUT:

```
 Enter the input :i*i+i*i

 stack                          input
0                              i*i+i*i$
0i5                            *i+i*i$
0F3                            *i+i*i$
0T2                            *i+i*i$
0T2*7                          i+i*i$
0T2*7i5                        +i*i$
0T2*7F10                                +i*i$
0E1                            +i*i$
0E1+6                          i*i$
0E1+6i5                        *i$
0E1+6F3                        *i$
0E1+6T9                        *i$
0E1+6T9*7                               i$
0E1+6T9*7i5                             $
0E1+6T9*7F10                            $
0E1+6T9                        $
0E1                            $
 accept the input
```

ADDITIONAL PROGRAMS

OBJECTIVE:

*Write a C program for constructing of LL (1) parsing.

AIM:

Analyzing the constructing of LL (1) parser.

RESOURCE:

urbo C++

PROGRAM LOGIC:

- the input string.
- predictive parsing table parse the given input using stack .
- stack [i] matches with token input string pop the token else shift it repeat the process until it reaches to $.

PROCEDURE:

- to debug -> run or press CTRL + F9 to run the program.

PROGRAM

```
#include<stdio.h>
#include<conio.h>
#include<string.h>
char s[20],stack[20];
void main()
{
 char m[5][6][3]={"tb"," "," ","tb"," "," "," ","+tb"," "," ","n","n","fc"," "," ","fc"," "," ","
```

```
","n","*fc"," ","n","n","i"," "," ","(e)"," "," "};
 int size[5][6]={2,0,0,2,0,0,0,3,0,0,1,1,2,0,0,2,0,0,0,1,3,0,1,1,1,0,0,3,0,0};
 int i,j,k,n,str1,str2;
 clrscr();
 printf("\n Enter the input string: ");
scanf("%s",s);
strcat(s,"$");
n=strlen(s);
stack[0]='$';
stack[1]='e';
i=1;
j=0;
printf("\nStack     Input\n");
printf("_____\n");
while((stack[i]!='$')&&(s[j]!='$'))
{
 if(stack[i]==s[j])
{
 i--;
 j++;
 }
 switch(stack[i])
{
 case 'e': str1=0;
break;
 case 'b': str1=1;
break;
 case 't': str1=2;
break;
 case 'c': str1=3;
break;
 case 'f': str1=4;
break;
}
```

```
 switch(s[j])
{
case 'i': str2=0;
break;
 case '+': str2=1;
break;
 case '*': str2=2;
break;
 case '(': str2=3;
break;
 case ')': str2=4;
break;
 case '$': str2=5;
break;
}
 if(m[str1][str2][0]=='\0')
{
 printf("\nERROR");
 exit(0);
 }
else if(m[str1][str2][0]=='n')
 i--;
else if(m[str1][str2][0]=='i')
 stack[i]='i';
else
{
for(k=size[str1][str2]-1;k>=0;k--)
{
stack[i]=m[str1][str2][k];
 i++;
}
 i--;
}
for(k=0;k<=i;k++)
```

```
 printf(" %c",stack[k]);
 printf("       ");
for(k=j;k<=n;k++)
 printf("%c",s[k]);
 printf(" \n ");
}
printf("\n SUCCESS");
getch();
}
```

INPUT & OUTPUT:

Enter the input string:i*i+i

STACK	**INPUT**
$bt	i*i+i$
$bcf	i*i+i$
$bci	i*i+i$
$bc	*i+i$
$bcf*	*i+i$
$bcf	i+i$
$bci	i+i$
$bc	+i$
$b	+i$
$bt+	+i$
$bt	i$
$bcf	i$
$ bci	i$
$bc	$
$b	$
$	$

success

PRE LAB QUESTIONS:

1. What is LL(1) parsing?
2. What are the disadvantages of brute force method?
3. How to parse LL(1) parser

LAB ASSIGNMENT:

1. Write a program to compute i+i*I using LL(1) parser

POST LAB QUESTIONS

1. What is Predictive parser?
2. What is Recursive Decent Parser?
3. How many types of Parsers are there?

OBJECTIVE:

Construction of recursive descent parsing for the following grammar

E->TE'

E'->+TE/@ "@ represents null character"

T->FT'

T`->*FT'/@

F->(E)/ID

AIM:

Analyzing recursive descent parsing grammar.

RESOURCE:

- T urbo C++

PROGRAM LOGIC:

- R ead the input string.
- W rite procedures for the non terminals
- V erify the next token equals to non terminals if it satisfies match the non terminal. If the input string does not match print error.

PROCEDURE:

- G o to debug -> run or press CTRL + F9 to run the program.

PROGRAM:

```
#include<stdio.h>
#include<conio.h>
#include<string.h>
char input[100];
int i,l;
void main()
{
 clrscr();
 printf("\nRecursive descent parsing for the following grammar\n");
 printf("\nE->TE'\nE'->+TE'/@\nT->FT'\nT'->*FT'/@\nF->(E)/ID\n");
 printf("\nEnter the string to be checked:");
 gets(input);
 if(E())
{
 if(input[i+1]=='\0')
 printf("\nString is accepted");
else
 printf("\nString is not accepted");
}
else
printf("\nString not accepted");
getch();
}
E()
{
 if(T())
{
 if(EP())
 return(1);
 else
 return(0);
}
else
 return(0);
```

```
}
EP()
{
 if(input[i]=='+')
{
 i++;
 if(T())
 {
 if(EP())
return(1);
      else
           return(0);
     }
    else
      return(0);
   }
  else
    return(1);
 }
T()
 {
  if(F())
   {
    if(TP())
     return(1);
    else
     return(0);
   }
  else
   return(0);
 }
TP()
 {
  if(input[i]=='*')
```

```
  {
   i++;
   if(F())
    {
     if(TP())
      return(1);
     else
      return(0);
    }
   else
    return(0);
  }
 else
  return(1);
}
F()
{
 if(input[i]=='(')
  {
   i++;
   if(E())
    {
     if(input[i]==')')
      {
          i++;
          return(1);
      }
    else
      return(0);
    }
   else
    return(0);
  }
 else if(input[i]>='a'&&input[i]<='z'||input[i]>='A'&&input[i]<='Z')
```

```
        {
         i++;
         return(1);
        }
    else
        return(0);
}
```

INPUT & OUTPUT:

INPUT:

Recursive descent parsing for the following grammar

E->TE'

E'->+TE'/@

T->FT'

T'->*FT'/@

F->(E)/ID

Enter the string to be checked:(a+b)*c

OUTPUT:

String is accepted

INPUT:

Recursive descent parsing for the following grammar

E->TE'

E'->+TE'/@

T->FT'

T'->*FT'/@

F->(E)/ID

Enter the string to be checked:a/c+d

OUTPUT:

String is not accepted

PRE LAB QUESTIONS:

1. What is parse tree?
2. What is LL(1) parser?
3. What are the derivation methods to generate a string for the given grammar?
4. What is the output of parse tree?

LAB ASSIGNMENT:

1. Write a program to compute recursive descent parsing for the following grammar?

   E□ TE'

   E'□ +TE'/î

   T□ FT'

   T'□ *FT'/î

   F□ (E)/i
2. Write a program to compute recursive descent parsing for the following grammar?

   S□ iCtSS'

   S'□ eS/ î
3. Write a program to compute recursive descent parsing for the following grammar?

   S□ iCtSS'

   S'□ eS/ î

POST LAB QUESTIONS

1. What is Predictive parser?
2. How many types of analysis can we do using Parser?
3. What is Recursive Decent Parser?
4. How many types of Parsers are there?
5. What is LR Parser?