NRI INSTITUTE OF TECHNOLOGY

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING

COMPILER DESIGN LAB MANUAL

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING

Vision:

To achieve Academic Excellence in Computer Science and Engineering by imparting knowledge to the students towards fulfilling the ever changing Industrial demands and Societal needs

Mission:

M1: To produce professionally competitive Computer Science Engineers with Excellent skill set.
M2: To fulfill Global demands, State of the Art Laboratories and Research facilities are developed to impart knowledge based education.
M3: Through Career Development Training, skills required for Employability and Societal needs are developed.

Program Specific Outcomes:

PSO1:Professional Skills: The ability to understand, analyze and develop computer programs in the areas related to Algorithms, System Software, Multimedia, Web design, Big Data Analytics, and networking for efficient 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: 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.

Program Educational Objectives:

PEO 1: To Produce Graduates with a Strong foundation in Mathematics, Science and Computer Engineering fundamentals to solve Engineering problems and also pursue Higher studies.

PEO 2: Graduates with Ability to Analyze, Design and Synthesize Data, Create Novel Products to satisfy Industry needs.

PEO 3: Ability to Understand and Analyze Engineering issues in a broader perspective with Ethical responsibility towards Sustainable Development and Societal needs.

PEO 4: Graduates with Managerial, Soft Skills, Entrepreneurship and Leadership Qualities in order to be Competent Professional.

Programme Outcomes:

After completion of the Computer Science and Engineering program students will have:

PO1: Engineering Knowledge: Apply knowledge of mathematics and science, with fundamentals of Computer Science & Engineering to be able to solve complex engineering problems related to CSE.

PO2: Problem Analysis: Identify, formulate, review research literature and analyze complex engineering problems related to CSE and reaching substantiated conclusions using first principles of mathematics, natural sciences and engineering sciences.

PO3: Design/Development of Solutions: Design solutions for complex engineering problems related to CSE 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 conclusions.

PO5: Modern Tool Usage: Create, select and apply appropriate techniques, resources and modern Engineering and IT tools including prediction and modeling to computer science related complex Engineering activities with an understanding of the limitations.

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 CSE professional engineering practice.

PO7: Environment and Sustainability: Understand the impact of the CSE 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 society at large such as , able to comprehend, write effective reports, design documentation, and to make effective presentations .

PO11: Project Management and Finance: Demonstrate knowledge, understanding of the engineering management principles and apply these to one's own work, as a member and leader in a team to manage projects and in multi disciplinary environments.

PO12: Life-long Learning: Recognize the need for preparation and ability to engage in independent, lifelong learning the broadest context of technological change.

NRI INSTITUTE OF TECHNOLOGY

(AUTONOMOUS)

Approved by AICTE, New Delhi: Permanently Affiliated to JNTUK, Kakinada
Accredited by NAAC with "A" GRADE, Accredited by NBA (CSE, ECE&EEE)
An ISO 9001:2015 Certified Institution
Pothavarappadu (V), Agiripalli (M), Eluru District, A.P., India, Pin: 521 212
URL: www.nriit.edu.in, email: principal@nriit.edu.in, Mobile: + 91 8333882444

Name of the Course: COMPILER DESIGN **Course Code: 20A3205493**

Regulation:NRIA20 **Academic Year:2023-24** **Year/ Semester:III/II**

1. PRE-REQUISITES: FLAT, C- Programming, Data Structures, Statistics fundamentals

2. COURSE OBJECTIVES: The students will be able to learn

1. To describe the design of a compiler including its phases and components and basic understanding of Grammars and language definition.
2. To Identify the similarities and differences among various parsing techniques and grammar transformation Techniques.
3. To Understand the syntax analysis, intermediate code generation, type checking, the role of symbol table and its organization.
4. To Understand, design code generation and optimization schemes.

3. COURSE OUTCOMES: At the end of the course, students will be able to

CO1: To use the knowledge of patterns, tokens & regular expressions for solving a problem.

CO2: To apply the knowledge of lex tool & yacc tool to develop a scanner & parser.

CO3: To write the new code optimization techniques to improve the performance of a program in terms of speed & space.

C04: To employ the knowledge of modern compiler & its features.

CO5: To participate in GATE, PGECET and other competitive examinations

4. Course Articulation Matrix with PO & PSO:

Course Outcomes vs POs Mapping

Courses Outcomes	PO 1	PO 2	PO 3	PO 4	PO 5	PO 6	PO 7	PO 8	PO 9	PO 10	PO 11	PO 12
CO1	3	3	2	--	3	--	--	--	--	--	--	
CO2	3	2	3	--	2	--	--	--	--	--	--	--
CO3	2	2	3	--	--	--	--	--	--	--	--	
CO4	3	-3	2	2	2	--	--	--	--	--	--	--
CO5	2	3	3	3	2	--	--	--	--	--	--	

Course Outcomes vs PSOs Mapping

Courses Outcomes	PSO1	PSO2	PSO3
CO1	3	3	--
CO2	2	3	2
CO3	3	3	--
CO4	2	3	--
CO5	3	3	3
CO6	2	3	3

Note: Slight: 1 Moderate: 2 High: 3 No relevance: --

5. List of Programs

S.NO	Programs to be Covered	Mapping CO's
1	Design a lexical analyzer for given language and the lexical analyzer should ignore redundant spaces, tabs and new lines. It should also ignore comments. Although the syntax specification states that identifiers can be arbitrarily long, you may restrict the length to some reasonable value. Simulate the same in C language	CO1
2	Write a LEX program to scan reserved word & identifiers of C language	CO2
3	Implementation of Predictive Parsing Algorithm	CO2
4	Write a C Program to generate three address code	CO3
5	Implementation of SLR(1) Parsing	CO4
6	Write a program to Design LALR Bottom up Parser.	CO4

EXPERIMENT 1:

AIM: Design a lexical analyzer for given language and the lexical analyzer should ignore redundant spaces,tabs and new lines. It should also ignore comments. Although the syntax specification states that identifiers can be arbitrarily long, you may restrict the length to some reasonable value. Simulate the same in C language.

LOGIC:

1. Read the input Expression

2. Check whether input is alphabet or digits then store it as identifier

3. If the input is is operator store it as symbol

4. Check the input for keywords

```
#include<string.h>
#include<ctype.h>
#include<stdio.h>
void keyword(char str[10])
{
if(strcmp("for",str)==0||strcmp("while",str)==0||strcmp("do",str)==0||
strcmp("int",str)==0||strcmp("float",str)==0||strcmp("char",str)==0||strcmp("double",str)==0||
strcmp("static",str)==0||strcmp("switch",str)==0||strcmp("case",str)==0)
printf("\n%s is a keyword",str);
else
printf("\n%s is an identifier",str);
}
main()
{
FILE *f1,*f2,*f3;
char c,str[10],st1[10];
int num[100],lineno=0,tokenvalue=0,i=0,j=0,k=0;
printf("\nEnter the c program");/*gets(st1);*/
f1=fopen("input","w");
while((c=getchar())!=EOF)
putc(c,f1);
fclose(f1);
f1=fopen("input","r");
f2=fopen("identifier","w");
f3=fopen("specialchar","w");
while((c=getc(f1))!=EOF){
if(isdigit(c))
{
tokenvalue=c-'0';
c=getc(f1);
while(isdigit(c)){
tokenvalue*=10+c-'0';
c=getc(f1);
}
num[i++]=tokenvalue;
```

```
ungetc(c,f1);
}
else if(isalpha(c))
{
putc(c,f2);
c=getc(f1);
while(isdigit(c)||isalpha(c)||c=='_'||c=='$')
{
putc(c,f2);
c=getc(f1);
}
putc(' ',f2);
ungetc(c,f1);
}
else if(c==' '||c=='\t')
printf(" ");
else
if(c=='\n')
lineno++;
else
putc(c,f3);
}
fclose(f2);
fclose(f3);
fclose(f1);
printf("\nThe no's in the program are");
for(j=0;j<i;j++)
printf("%d",num[j]);
printf("\n");
f2=fopen("identifier","r");
k=0;
printf("The keywords and identifiersare:");
while((c=getc(f2))!=EOF){
if(c!=' ')
str[k++]=c;
else
{
str[k]='\0';
keyword(str);
k=0;
}
}
fclose(f2);
f3=fopen("specialchar","r");
printf("\nSpecial characters are");
while((c=getc(f3))!=EOF)
printf("%c",c);
printf("\n");
fclose(f3);
printf("Total no. of lines are:%d",lineno);
```

```
}
```

Input:

```
Enter Program $ for termination:
{
int a[3],t1,t2;
t1=2; a[0]=1; a[1]=2; a[t1]=3;
t2=-(a[2]+t1*6)/(a[2]-t1);
if t2>5 then
print(t2);
else {
int t3;
t3=99;
t2=-25;
print(-t1+t2*t3); /* this is a comment on 2 lines */
} endif
}
(cntrl+z)
```

Output:

```
 Variables : a[3] t1 t2 t3
Operator : - + * / >
Constants : 2 1 3 6 5 99 -25
Keywords : int if then else endif
Special Symbols : , ; ( ) { }
Comments : this is a comment on 2 lines
```

EXPERIMENT 2:

AIM:Write a LEX Program to scan reserved word & Identifiers of C Language

```
#include<string.h>
#include<ctype.h>
#include<stdio.h>
void keyword(char str[10])
{
if(strcmp("for",str)==0||strcmp("while",str)==0||strcmp("do",str)==0||strcmp("int",
str
)==0||strcmp("float",str)==0||strcmp("char",str)==0||strcmp("double",str)==0||str
cmp("static",str)==0||strcmp("switch",str)==0||strcmp("case",str)==0)
printf("\n%s is a keyword",str);
else
printf("\n%s is an identifier",str);
}
main()
{
FILE *f1,*f2,*f3;
char c, str[10], st1[10];
int num[100], lineno=0, tokenvalue=0,i=0,j=0,k=0;
printf("\n Enter the c program : ");/*gets(st1);*/
f1=fopen("input","w");
while((c=getchar())!=EOF)
 putc(c,f1);
fclose(f1);
f1=fopen("input","r");
f2=fopen("identifier","w");
f3=fopen("specialchar","w");
while((c=getc(f1))!=EOF)
{
if(isdigit(c))
{
tokenvalue=c-'0';
c=getc(f1);
while(isdigit(c))
{
tokenvalue*=10+c-'0';
c=getc(f1);
}
num[i++]=tokenvalue;
ungetc(c,f1);
}
else
if(isalpha(c))
{
putc(c,f2);
c=getc(f1);
while(isdigit(c)||isalpha(c)||c=='_'||c=='$')
```

```
{
putc(c,f2);
c=getc(f1);
}
putc(' ',f2);
ungetc(c,f1);
}
else
if(c==' '||c=='\t')
printf(" ");
else
if(c=='\n')
lineno++;
else
putc(c,f3);
}
fclose(f2);
fclose(f3);
fclose(f1);
printf("\n The no's in the program are :");
for(j=0; j<i; j++)
printf("%d", num[j]);
printf("\n");
f2=fopen("identifier", "r");
k=0;
printf("The keywords and identifiers are:");
while((c=getc(f2))!=EOF)
{
if(c!=' ')
str[k++]=c;
else
{
str[k]='\0';
keyword(str);
k=0;
}
 }
 fclose(f2);
f3=fopen("specialchar","r");
printf("\n Special characters are : ");
while((c=getc(f3))!=EOF)
printf("%c",c);
printf("\n");
fclose(f3);
printf("Total no. of lines are:%d", lineno);
}
```

Output :

Enter the C program: a+b*c

Ctrl-D

The no's in the program are:

The keywords and identifiers are:

a is an identifier and terminal

b is an identifier and terminal

c is an identifier and terminal

Special characters are:

+ *

EXPERIMENT 3:
Aim:Implement Predictive Parsing algorithm

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

char prol[7][10] = { "S", "A", "A", "B", "B", "C", "C" };
char pror[7][10] = { "A", "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];

int numr(char c)
{
  switch (c)
  {
    case 'S':
      return 0;

    case 'A':
      return 1;

    case 'B':
      return 2;

    case 'C':
      return 3;

    case 'a':
      return 0;

    case 'b':
      return 1;
```

```
    case 'c':
      return 2;

    case 'd':
      return 3;

    case '$':
      return 4;
  }

  return (2);
}

int main()
{
  int i, j, k;

  for (i = 0; i < 5; i++)
    for (j = 0; j < 6; j++)
      strcpy(table[i][j], " ");

  printf("The following grammar is used for Parsing Table:\n");

  for (i = 0; i < 7; i++)
    printf("%s\n", prod[i]);

  printf("\nPredictive parsing table:\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("%-10s", table[i][j]);
      if (j == 5)
        printf("\n----------------------------------------------------\n");
    }
}
```

Output:

The following grammar is used for Parsing Table:

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

Predictive parsing table:

```
----------------------------------------------------
        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->@

EXPERIMENT 4:
Write a C program to generate three address code.

```
#include<stdio.h>
#include<string.h>
void pm();
void plus();
void div();
int i,ch,j,l,addr=100;
char ex[10], exp[10] ,exp1[10],exp2[10],id1[5],op[5],id2[5];
void main()
{
clrscr();
while(1)
{
printf("\n1.assignment\n2.arithmetic\n3.relational\n4.Exit\nEnter the choice:");
scanf("%d",&ch);
switch(ch)
{
case 1:
printf("\nEnter the expression with assignment operator:");
scanf("%s",exp);
l=strlen(exp);
exp2[0]='\0';
i=0;
while(exp[i]!='=')
{
i++;
}
strncat(exp2,exp,i);
strrev(exp);
exp1[0]='\0';
strncat(exp1,exp,l-(i+1));
strrev(exp1);
printf("Three address code:\ntemp=%s\n%s=temp\n",exp1,exp2);
break;

case 2:
printf("\nEnter the expression with arithmetic operator:");
scanf("%s",ex);
strcpy(exp,ex);
l=strlen(exp);
exp1[0]='\0';
```

```
for(i=0;i<l;i++)
{
if(exp[i]=='+'||exp[i]=='-')
{
if(exp[i+2]=='/'||exp[i+2]=='*')
{
pm();
break;
}
else
{
plus();
break;
}
}
else if(exp[i]=='/'||exp[i]=='*')
{
div();
break;
}
}
break;

case 3:
printf("Enter the expression with relational operator");
scanf("%s%s%s",&id1,&op,&id2);
if(((strcmp(op,"<")==0)||(strcmp(op,">")==0)||(strcmp(op,"<=")==0)||(strcmp(op,">=")==0)||(str
cmp(op,"==")==0)||(strcmp(op,"!=")==0))==0)
printf("Expression is error");
else
{
printf("\n%d\tif %s%s%s goto %d",addr,id1,op,id2,addr+3);
addr++;
printf("\n%d\t T:=0",addr);
addr++;
printf("\n%d\t goto %d",addr,addr+2);
addr++;
printf("\n%d\t T:=1",addr);
}
break;
case 4:
exit(0);
}
}
}
void pm()
{
strrev(exp);
j=l-i-1;
```

```
strncat(exp1,exp,j);
strrev(exp1);
printf("Three address code:\ntemp=%s\ntemp1=%c%ctemp\n",exp1,exp[j+1],exp[j]);
}
void div()
{
strncat(exp1,exp,i+2);
printf("Three address code:\ntemp=%s\ntemp1=temp%c%c\n",exp1,exp[i+2],exp[i+3]);
}
void plus()
{
strncat(exp1,exp,i+2);
printf("Three address code:\ntemp=%s\ntemp1=temp%c%c\n",exp1,exp[i+2],exp[i+3]);
}
```

Example Generation of Three Address Project Output Result

```
1. assignment
2. arithmetic
3. relational
4. Exit
Enter the choice:1
Enter the expression with assignment operator:
a=b
Three address code:
temp=b
a=temp

1.assignment
2.arithmetic
3.relational
4.Exit
Enter the choice:2
Enter the expression with arithmetic operator:
a+b-c
Three address code:
temp=a+b
temp1=temp-c

1.assignment
2.arithmetic
3.relational
4.Exit
Enter the choice:2
Enter the expression with arithmetic operator:
a-b/c
Three address code:
temp=b/c
temp1=a-temp
```

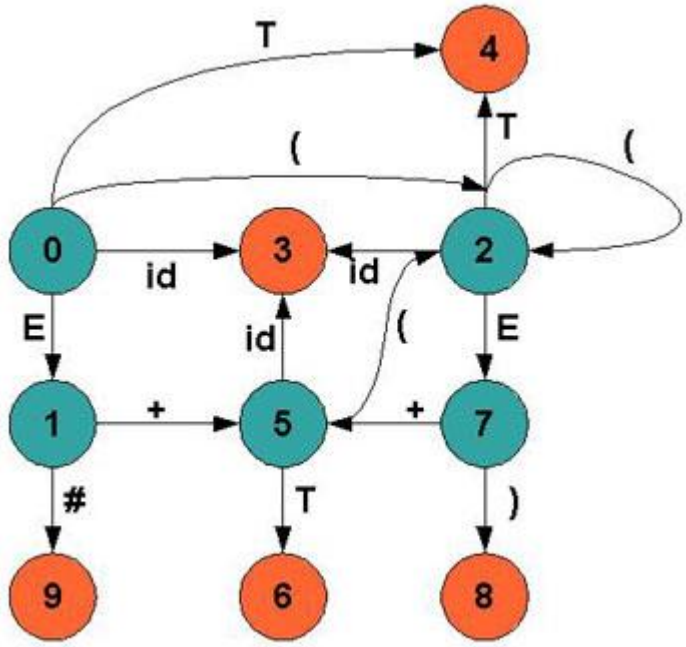
```
1.assignment
2.arithmetic
3.relational
4.Exit
Enter the choice:2
Enter the expression with arithmetic operator:
a*b-c
Three address code:
temp=a*b
temp1=temp-c

1.assignment
2.arithmetic
3.relational
4.Exit
Enter the choice:2
Enter the expression with arithmetic operator:a/b*c
Three address code:
temp=a/b
temp1=temp*c
1.assignment
2.arithmetic
3.relational
4.Exit
Enter the choice:3
Enter the expression with relational operator
a
<=
b

100 if a<=b goto 103
101 T:=0
102 goto 104
103 T:=1

1.assignment
2.arithmetic
3.relational
4.Exit
Enter the choice:4
```

EXPERIMENT 5:
Implement SLR(1) Parsing algorithm




```
#include<stdio.h>

#include<string.h>


int axn[][6][2]={

    {{100,5},{-1,-1},{-1,-1},{100,4},{-1,-1},{-1,-1}},

    {{-1,-1},{100,6},{-1,-1},{-1,-1},{-1,-1},{102,102}},

    {{-1,-1},{101,2},{100,7},{-1,-1},{101,2},{101,2}},

    {{-1,-1},{101,4},{101,4},{-1,-1},{101,4},{101,4}},

    {{100,5},{-1,-1},{-1,-1},{100,4},{-1,-1},{-1,-1}},

    {{-1,-1},{101,6},{101,6},{-1,-1},{101,6},{101,6}},

    {{100,5},{-1,-1},{-1,-1},{100,4},{-1,-1},{-1,-1}},

    {{100,5},{-1,-1},{-1,-1},{100,4},{-1,-1},{-1,-1}},

    {{-1,-1},{100,6},{-1,-1},{-1,-1},{100,1},{-1,-1}},
```

```
        {{-1,-1},{101,1},{100,7},{-1,-1},{101,1},{101,1}},
        {{-1,-1},{101,3},{101,3},{-1,-1},{101,3},{101,3}},
        {{-1,-1},{101,5},{101,5},{-1,-1},{101,5},{101,5}}
        };//Axn Table
int gotot[12][3]={1,2,3,-1,-1,-1,-1,-1,-1,-1,-1,-1,8,2,3,-1,-1,-1,
   -1,9,3,-1,-1,10,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1};  //GoTo table
int a[10];
char b[10];
int top=-1,btop=-1,i;
void push(int k)
{
  if(top<9)
    a[++top]=k;
}

void pushb(char k)
{
  if(btop<9)
    b[++btop]=k;
}

char TOS()
{
   return a[top];
}
```

```
void pop()
{
  if(top>=0)
    top--;
}

void popb()
{
  if(btop>=0)
    b[btop--]='\0';
}

void display()
{
 for(i=0;i<=top;i++)
   printf("%d%c",a[i],b[i]);
}

void display1(char p[],int m) //Displays The Present Input String
{
 int l;
 printf("\t\t");
 for(l=m;p[l]!='\0';l++)
   printf("%c",p[l]);
```

```
  printf("\n");
}

void error()
{
  printf("Syntax Error");
}

void reduce(int p)
{
   int len,k,ad;
   char src,*dest;
   switch(p)
   {
 case 1:dest="E+T";
        src='E';
        break;
 case 2:dest="T";
        src='E';
        break;
 case 3:dest="T*F";
        src='T';
        break;
 case 4:dest="F";
        src='T';
```

```
        break;
case 5:dest="(E)";
        src='F';
        break;
case 6:dest="i";
        src='F';
        break;
default:dest="\0";
 src='\0';
 break;
  }
  for(k=0;k<strlen(dest);k++)
  {
    pop();
    popb();
  }
  pushb(src);
  switch(src)
  {
case 'E':ad=0;
  break;
case 'T':ad=1;
  break;
case 'F':ad=2;
  break;
```

```
 default: ad=-1;
   break;
   }
  push(gotot[TOS()][ad]);
}
int main()
{
   int j,st,ic;
   char ip[20]="\0",an;
  // clrscr();
   printf("Enter any String\n");
+
   scanf("%s",ip);
   push(0);
   display();
   printf("\t%s\n",ip);
   for(j=0;ip[j]!='\0';)
   {
 st=TOS();
 an=ip[j];
 if(an>='a'&&an<='z') ic=0;
 else if(an=='+') ic=1;
 else if(an=='*') ic=2;
 else if(an=='(') ic=3;
 else if(an==')') ic=4;
```

```
else if(an=='$') ic=5;
else {
   error();
   break;
}
  if(axn[st][ic][0]==100)
   {
     pushb(an);
     push(axn[st][ic][1]);
     display();
     j++;
     display1(ip,j);
   }
  if(axn[st][ic][0]==101)
   {
     reduce(axn[st][ic][1]);
     display();
     display1(ip,j);
   }
  if(axn[st][ic][1]==102)
   {
  printf("Given String is accepted \n");
//  getch();
     break;
   }
```

```
 /*  else
   {
   printf("Given String is rejected \n");
      break;
    }*/
  }
return 0;
}
/*
----------------------OUTPUT----------------------------
deepti@Inspiron-3542:~$ gcc slr.c
deepti@Inspiron-3542:~$ ./a.out
Enter any String
a+a*a$
0 a+a*a$
0a5  +a*a$
0F3  +a*a$
0T2  +a*a$
0E1  +a*a$
0E1+6  a*a$
0E1+6a5  *a$
0E1+6F3  *a$
0E1+6T9  *a$
0E1+6T9*7  a$
0E1+6T9*7a5  $
```

0E1+6T9*7F10 $

0E1+6T9 $

0E1 $

Given String is accepted

*/

EXPERIMENT 6:

Design LALR bottom up parser for the given language

<parser.l>

```
%{
#include<stdio.h>
#include "y.tab.h"
%}
%%
[0-9]+ {yylval.dval=atof(yytext);
return DIGIT;
}
\n|. return yytext[0];
%%
```

<parser.y>

```
%{
/*This YACC specification file generates the LALR parser for the program
considered in experiment 4.*/
#include<stdio.h>
%}
%union
{
double dval;
}
```

```
%token <dval> DIGIT
%type <dval> expr
%type <dval> term
%type <dval> factor
%%
line : expr '\n' {
;
printf("%g\n", $1);
}
expr : expr '+' term { $$=$1 + $3 ; }
| term
;
term: term '*' factor {$$=$1 * $3 ;}
| factor
;
factor: '(' expr ')' {$$=$2 ;}
| DIGIT
;
%%
39 | P a g e
int main()
{
Print(“ Enter AE:”);
yyparse();
}
yyerror(char *s)
{
printf("%s",s);
}
```

Output:

$ lex parser.l

$ yacc –d parser.y

$cc lex.yy.c y.tab.c –ll –lm

$./a.out

2+3

5.0000