

**LAB MANNUAL**  
**COMPILER LAB (KCS-552)**  
**B.Tech. – CSE (5<sup>th</sup> Sem.)**



**Department of Computer Science and Engineering**  
**United College of Engineering & Research, Prayagaraj**  
**A-31, UPSIDC, Industrial Area, Naini Allahabad**  
**Website: [www.united.ac.in](http://www.united.ac.in)**

**STUDY & EVALUATION SCHEME**

**Compiler Design Lab (KCS-552)**

**Total No. of Periods : 2 Periods per week**

**Sessional Evaluation:**

**INTERNAL EVALUATION**

**CLASS TEST 1 : 15 MARKS**

**CLASS TEST 2 : 15 MARKS**

**TEACHER'S ASSESMENT : 10 MARKS**

**ATTENDANCE : 10 MARKS**

**TOTAL INTERNAL MARKS: 50 MARKS (Converted to 25)**

**END SEMESTER EXAM : 25 MARKS**

**TOTAL MARKS : 50 MARKS**

**CREDIT : 1**

**SYLLABUS**

1. Design and implement a lexical analyzer for given language using C and the lexical analyzer should ignore redundant spaces, tabs and new lines.
2. Implementation of Lexical Analyzer using Lex Tool
3. Generate YACC specification for a few syntactic categories.
  - a) Program to recognize a valid arithmetic expression that uses operator +, −, \* and /.
  - b) Program to recognize a valid variable which starts with a letter followed by any number of letters or digits.
  - c) Implementation of Calculator using LEX and YACC
  - d) Convert the BNF rules into YACC form and write code to generate abstract syntax tree
4. Write program to find  $\epsilon$  – closure of all states of any given NFA with  $\epsilon$  transition.
5. Write program to convert NFA with  $\epsilon$  transition to NFA without  $\epsilon$  transition.
6. Write program to convert NFA to DFA
7. Write program to minimize any given DFA.
8. Develop an operator precedence parser for a given language.
9. Write program to find Simulate First and Follow of any given grammar.
10. Construct a recursive descent parser for an expression.
11. Construct a Shift Reduce Parser for a given language.
12. Write a program to perform loop unrolling.
13. Write a program to perform constant propagation.
14. Implement Intermediate code generation for simple expressions.
15. Implement the back end of the compiler which takes the three address code and produces the 8086 assembly language instructions that can be assembled and run using an 8086 assembler. The target assembly instructions can be simple move, add, sub, jump etc.

**Note: The Instructor may add/delete/modify/tune experiments, wherever he/she feels in a justified manner**

**It is also suggested that open source tools should be preferred to conduct the lab ( C, C++ , Lex or Flex and YACC tools ( Unix/Linux utilities )etc)**

List of Experiments			
Serial No.	Experiment	CO	Bloom's Level
1	WRITE A C PROGRAM TO RECOGNIZE STRING UNDER 'a*', 'a*b+', 'abb'.	CO1	K3
2	WRITE A LEX PROGRAM TO CHECK WHETHER INPUT STRING IS VERB OR NOT.(LIST OF VERB ARE GIVEN)	CO2	K3
3	WRITE A PROGRAM TO CHECK INPUT IDENTIFIER IS VALID OR NOT.	CO2	K3
4	WRITE A LEX PROGRAM TO COUNT NUMBER OF CHARACTER AND NUMBER OF LINES.	CO2	K3
5	WRITE A YACC PROGRAM TO IMPLEMENT GRAMMER FOR A SIMPLE CALCULATOR.	CO3	K3
6	WRITE A YACC PROGRAM TO IMPLEMENT IF-THEN-ELSE STATEMENT.	CO3	K3
7	WRITE A YACC PROGRAM TO IMPLEMENT FOR LOOP.	CO3	K3
8	WRITE A C PROGRAM TO IMPLEMENT RECURSIVE DESCENT PARSER FOR GRAMMAR:- E->E+T T T->T*F F F->id	CO3	K3
9	WRITE A PROGRAM TO DESIGN LALR BOTTOM UP PARSER.	CO3	K3
10	WRITE A PROGRAM TO CONVERT NFA INTO DFA.	CO3	K3
11	WRITE A PROGRAM FOR IMPLEMENTATION OF	CO3	K3

	SHIFT REDUCE PARSING ALGORITHM		
12	WRITE A C PROGRAM TO GENERATE MACHINE CODE FROM ABSTRACT SYNTAX TREE GENERATED BY THE PARSER. THE INSTRUCTION SET SPECIFIED MAY BE CONSIDERED AS THE TARGET CODE.	CO5	K3

# **COURSE OUTCOMES**

**Course Name:** Compiler Design Lab

**AKTU Course Code:** KCS552

<b>CO No.</b>	<b>Statement</b>
CO1	Basic understanding about tokens & regular expressions for lexical analysis.
CO2	Design Lexical analyser for given language using C and LEX and YACC programming language.
CO3	Design and analyze top down and bottom up parsers.
CO4	Generate the intermediate code
CO5	Generate machine code from the intermediate code forms

## **CO-PO MATRIX**

**Course Name:** COMPILER DESIGN LAB (KCS-552)

<b>C O</b>	<b>PO 1</b>	<b>PO 2</b>	<b>PO 3</b>	<b>PO 4</b>	<b>PO 5</b>	<b>PO 6</b>	<b>PO 7</b>	<b>PO 8</b>	<b>PO 9</b>	<b>PO1 0</b>	<b>PO1 1</b>	<b>PO1 2</b>
1	3	1	3	2	3	-	-	-	-	-	-	-
2	3	2	3	2	3	-	-	-	-	-	-	-
3	3	2	3	2	3	-	-	-	-	-	-	-
4	3	1	3	1	3	-	-	-	-	-	-	-
5	3	1	3	1	3	-	-	-	-	-	-	-

## **CO-PSO MATRIX**

<b>CO</b>	<b>PSO1</b>	<b>PSO2</b>	<b>PSO3</b>
1	1	1	-
2	1	2	-
3	1	2	-
4	1	2	-
5	1	2	-

Enter attainment levels 1, 2 or 3 as defined below:

1: Slight (Low)      2: Moderate (Medium)      3: Substantial (High)

If there is no attainment, put “-”

## PROGRAM OUTCOME

<b>PO -1</b>	<b>Engineering Knowledge:</b> Apply the knowledge of mathematics, science, engineering fundamentals, and an engineering specialization to the solution of complex engineering problems.
<b>PO -2</b>	<b>Problem Analysis:</b> Identify, formulate, review research literature, and analyse complex engineering problems reaching substantiated conclusions using first principles of mathematics, natural sciences and engineering sciences.
<b>PO -3</b>	<b>Design/Development of Solutions:</b> 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.
<b>PO -4</b>	<b>Conduct Investigations of Complex Problems:</b> 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 for complex problems: <ul style="list-style-type: none"><li>• that cannot be solved by straightforward application of knowledge, theories and techniques applicable to the engineering discipline as against problems given at the end of chapters in a typical text book that can be solved using simple engineering theories and techniques;</li><li>• that may not have a unique solution. For example, a design problem can be solved in many ways and lead to multiple possible solutions;</li><li>• that require consideration of appropriate constraints / requirements not explicitly given in the problem statement such as cost, power requirement, durability, product life, etc.;</li><li>• which need to be defined (modelled) within appropriate mathematical framework; and</li><li>• that often require use of modern computational concepts and tools, for example, in the design of an antenna or a DSP filter.</li></ul>
<b>PO -5</b>	<b>Modern Tool Usage:</b> Create, select, and apply appropriate techniques, resources, and modern engineering and IT tools including prediction and modelling to complex engineering activities with an understanding of the limitations.
<b>PO -6</b>	<b>The Engineer and Society:</b> 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.
<b>PO -7</b>	<b>Environment and Sustainability:</b> Understand the impact of the professional engineering solutions in societal and environmental contexts, and demonstrate the knowledge of, and need for sustainable development.
<b>PO -8</b>	<b>Ethics:</b> Apply ethical principles and commit to professional ethics and responsibilities and norms of the engineering practice.



<b>PO -9</b>	<b>Individual and Team Work:</b> Function effectively as an individual, and as a member or leader in diverse teams, and in multidisciplinary settings.
<b>PO -10</b>	<b>Communication:</b> 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.
<b>PO -11</b>	<b>Project Management and Finance:</b> 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 in multidisciplinary environments.
<b>PO -12</b>	<b>Life-long Learning:</b> Recognize the need for, and have the preparation and ability to engage in independent and lifelong learning in the broadest context of technological change.

<b>PSO-1</b>	An ability to demonstrate basic knowledge of Database System, Software Engineering, Computer Hardware, Networking and Operating System.
<b>PSO-2</b>	An ability to Design & Develop Program, Algorithms and Projects using efficient Data Structure.
<b>PSO-3</b>	An ability to apply their skills in the field of web designing, cloud computing, machine learning, artificial intelligence, IOT and data analytics.

## **GENERAL LABORATORY INSTRUCTIONS**

1. Students are advised to come to the laboratory at least 5 minutes before (to starting time), those who come after 5 minutes will not be allowed into the lab.
2. Plan your task properly much before to the commencement, come prepared to the lab with the synopsis / program / experiment details.
3. Student should enter into the laboratory with:
  - a. Laboratory observation notes with all the details (Problem statement, Aim, Algorithm, Procedure, Program, Expected Output, etc.,) filled in for the lab session.
  - b. Laboratory Record updated up to the last session experiments and other utensils (if any) needed in the lab.
  - c. Proper Dress code and Identity card.
4. Sign in the laboratory login register, write the TIME-IN, and occupy the computer system allotted to you by the faculty.
5. Execute your task in the laboratory, and record the results / output in the lab observation note book, and get certified by the concerned faculty.
6. All the students should be polite and cooperative with the laboratory staff, must maintain the discipline and decency in the laboratory.
7. Computer labs are established with sophisticated and high end branded systems, which should be utilized properly.
8. Students / Faculty must keep their mobile phones in SWITCHED OFF mode during the lab sessions. Misuse of the equipment, misbehaviors with the staff and systems etc., will attract severe punishment.
9. Students must take the permission of the faculty in case of any urgency to go

out ; if anybody found loitering outside the lab / class without permission during working hours will be treated seriously and punished appropriately.

10. Students should LOG OFF/ SHUT DOWN the computer system before he/she leaves the lab after completing the task (experiment) in all aspects. He/she must ensure the system / seat is kept properly.

\

## **SOFTWARE REQUIREMENTS**

### **SOFTWARE**

1. Operating System (Windows / Linux )
2. IDE for C language (Turbo C / C++, Borland C / C++ or any)
3. Flex & Bison

## PROGRAM NO:-1

**OBJECT:**WRITE A C PROGRAM TO RECOGNIZE STRING UNDER  
'a\*', 'a\*b+', 'abb'.

```
#include<stdio.h>
#include<string.h>
#include<stdlib.h>
void main()
{
    char s[20],c;
    int state=0,i=0;
    printf("\n enter a string:");
    scanf("\n%s",s);
    while(s[i]!='\0')
    {
        switch(state)
        {
            case 0:
                c=s[i++];
                if(c=='a')
                    state =1;
                else if(c=='b')
                    state=2;
                else
                    state=6;
                break;
            case 1:
                c=s[i++];
                if(c=='a')
                    state=3;
                else if(c=='b')
                    state=4;
                else
                    state=6;
                break;
            case 2:
                c=s[i++];
                if(c=='a')
                    state=6;
                else if(c=='b')
```

```

        state=2;
        else
        state=6;
        break;
case 3:
    c=s[i++];
    if(c=='a')
        state=3;
        else if(c=='b')
        state=2;
        else
        state=6;
        break;
case 4:
    c=s[i++];
    if(c=='a')
        state=6;
        else if(c=='b')
        state=5;
        else
        state=6;
        break;
case 5:
    c=s[i++];
    if(c=='a')
        state=6;
        else if(c=='b')
        state=2;
        else
        state=6;
        break;
case 6:
    printf("%s is not recognised.",s);
    exit(0);
}
}
if((state==1)||(state==3)||(state==0))
printf("\n %s is accepted under rule 'a*'",s);
else if((state==2)||(state==4))
printf("\n %s is accepted under rule 'a*b+'",s);
else if(state==5)

```

```
        printf("\n %s is accepted under rule 'abb'",s);
    }
}
```

### **PROGRAM N0:02**

**OBJECT:**WRITE A LEX PROGRAM TO CHEAK WETHER INPUT STRING IS VERB OR NOT.(LIST OF VERB ARE GIVEN)

```
%%
    [\t]+
    is |
    am |
    are|
    was|
    were{printf("%s: is a verb",yytext);}
    [a-zA-Z]+{printf("%s: is not a verb,yytext);}
%%
main()
{
    yylex();
}
```

**OUTPUT:** \$ ./a.out  
are  
are:is a verb

### **PROGRAM N0:03**

**OBJECT:**WRITE A PROGRAM TO CHEAK INPUT IDENTEFIER IS VALID OR NOT.

```
% {  
#include<stdio.h>  
% }  
digit[0-9]  
letter[a-zA-Z]  
%%  
{ letter } ( { letter } | { digit } ) * printf("id: %s\n",yytext);  
\n printf("new line\n");  
%%  
main()  
{  
yylex();
```

**OUTPUT:** > ./a.out

**abhishek  
id:abhishek**

**new line**

**555**

**555new line**

## **PROGRAM NO:04**

**Object:** WRITE A LEX PROGRAM TO COUNT NUMBER OF CHARACTER AND NUMBER OF LINES.

```
% {  
#include<stdio.h>  
int num_lines=0,num_chars=0;  
% }  
%%  
\n {++num_lines; ++num_chars;}  
. {++num_chars;}  
%%  
int main()  
{  
yylex();  
printf("There are %d lines and %d characters. \n", num_lines,num_chars);  
return 0;  
}
```

**OUTPUT:** > ./a.out

**Compiler Design.**

**There are 1 lines and 17 characters.**



## PROGRAM NO:05

**Object:** WRITE A YACC PROGRAM TO IMPLEMENT GRAMMER FOR A SIMPLE CALCULATOR.

```
% {
#define YYSTYPE double
#include "cal.tab.h"
#include <stdlib.h>
% }

white [ \t]+
digit [0-9]
integer {digit}+
exponent [eE][+-]?{integer}
real {integer}("."{integer})?{exponent}?

%%

{ white } { }
{ real } { yylval=atof(yytext);
return NUMBER;
}

"+" return PLUS;
"-" return MINUS;
"*" return TIMES;
"/" return DIVIDE;
"^" return POWER;
"(" return LEFT;
")" return RIGHT;
"\n" return END;

% {
#include <math.h>
#include <stdio.h>
#include <stdlib.h>
#define YYSTYPE double
% }
```

```
%token NUMBER
%token PLUS MINUS TIMES DIVIDE POWER
%token LEFT RIGHT
%token END
```

```
%left PLUS MINUS
%left TIMES DIVIDE
%left NEG
%right POWER
```

```
%start Input
%%
```

Input:

```
| Input Line;
```

Line:

END

```
| Expression END { printf("Result: %f\n", $1); }
;
```

Expression:

```
NUMBER { $$=$1; }
```

```
| Expression PLUS Expression { $$=$1+$3; }
```

```
| Expression MINUS Expression { $$=$1-$3; }
```

```
| Expression TIMES Expression { $$=$1*$3; }
```

```
| Expression DIVIDE Expression { $$=$1/$3; }
```

```
| MINUS Expression %prec NEG { $$=-$2; }
```

```
| Expression POWER Expression { $$=pow($1,$3); }
```

```
| LEFT Expression RIGHT { $$=$2; }
```

```
;
```

```
%%
```

```
int yyerror(char *s) {
    printf("%s\n", s);
}
```

```
int main() {
    if (yyparse())
```

```
fprintf(stderr, "Successful parsing.\n");
else
fprintf(stderr, "error found.\n");
}
```

**OUTPUT:** > ./a.out  
12\*34-67+23

**Result:364.000000**

### **PROGRAM NO:06**

**Object:** WRITE A YACC PROGRAM TO IMPLEMENT IF-THEN-ELSE STATEMENT.

#### **CREATE IFTE.L FILE-**

```
alpha [A-Za-z]
digit [0-9]
%%
[\\t\\n]
if      return IF;
then return THEN;
else   return ELSE;
{ digit }+({ alpha }|{ digit })* return ID;
"<=" return LE;
">=" return GE;
"==" return EQ;
"!=" return NE;
"||"  return OR;
"&&" return AND;
.     return yytext[0];
%%
```

#### **CREATE IFTE.Y FILE-**

```
% {
#include<stdio.h>
#include<stdlib.h>
% }
```

```

%token ID NUM IF THEN LE GE EQ NE OR AND ELSE
%right '='
%left AND OR
%left '<'>' LE GE EQ NE
%left '+' '-'
%left '*' '/'
%right UMINUS
%left '|'
%%

```

```

S : ST{printf("Input Accepted.\n");exit(0);};
ST : IF '(' E2 ')' THEN ST1 ';' ELSE ST1 ';'
    | IF '(' E2 ')' THEN ST1';'
    ;
ST1 : ST
      | E
      ;
E :   ID '=' E
      | E '+' E
      | E '-' E
      | E '*' E
      | E '/' E
      | E '<' E
      | E '>' E
      | E LE E
      | E GE E
      | E EQ E
      | E NE E
      | E OR E
      | E AND E
      | ID|NUM;
E2:  E '<' E
      | E '>' E
      | E LE E
      | E GE E
      | E EQ E
      | E NE E
      | E OR E
      | E AND E
      | ID
      | NUM

```

```

;
%%
#include "lex.yy.c"
main()
{
    printf("Enter The Expression: ");
    yyparse();
}

```

### **OUTPUT:**

**> ./a.out**

**Enter Expression:if(i==0) then x=2; else y=2;  
Input Accepted.**

**> ./a.out**

**Enter Expression:if(i==0) then x=2 else y=2;  
Syntax error.**

## **PROGRAM NO:07**

**Object:** WRITE A YACC PROGRAM TO IMPLEMENT FOR LOOP.

### **CREATE FORLP.L FILE-**

```

alpha [A-Za-z]
digit [0-9]
%%
[ \t\n]
for return FOR;
{ digit }+ return NUM;
{ alpha } ( { alpha } | { digit } ) * return ID;
"<=" return LE;
">=" return GE;
"==" return EQ;
"!=" return NE;
"||" return OR;
"&&" return AND;
. return yytext[0];

```

%%

## **CREATE FORLP.Y FILE-**

% {

#include<stdio.h>

#include<stdlib.h>

% }

%token ID NUM FOR LE GE EQ NE OR AND

%right '=' UMINUS

%left AND OR '<'>' LE GE EQ NE '+' '-' '\*' '/' '!'

%%

S:ST {printf("Input Accepted.\n"); exit(0);};

ST:FOR '(' E ';' E2 ';' E ')' DEF;

DEF:'{' BODY '}'

|E ';'

|ST

|

;

BODY:BODY BODY

|E';'

|ST

|

;

E:ID '=' E

|E '+' E

|E '-' E

|E '\*' E

|E '/' E

|E '<' E

|E '>' E

|E LE E

|E GE E

|E EQ E

|E NE E

|E OR E

|E AND E

|E '+' '+'

|E '-' '-'

|ID

|NUM

;

E2:E '<' E

|E '>' E

|E LE E

|E GE E

|E EQ E

|E NE E

|E OR E

|E AND E

|ID

|NUM

;

%%

#include "lex.yy.c"

main()

{

printf("Enter Expression: ");

yyvsparse();

}

### **OUTPUT:**

> ./a.out

```
Enter Expression: for(i=10;i<20;i++)
{
    x=x*i;
}
```

**Input Accepted.**

> ./a.out

**Enter Expression: for(i=10;i<20,i++)**

**syntax error**

**abhishek@abhishek-HP-Notebook:~\$ ./a.out**

**Enter Expression: for(i=10;i<20;i++)**

```
{
    for(j=2;j<5;j++)
    {
        x=i+j;
    }
}
```



}  
**Input Accepted.**

### **PROGRAM NO:08**

**Object:** WRITE A C PROGRAM TO IMPLEMENT RECURSIVE DECENT  
PARSER FOR GRAMMAR:-  $E \rightarrow E+T \mid T$

$T \rightarrow T * F \mid F$

$F \rightarrow id$

```
#include<stdio.h>
#include<string.h>
#include<ctype.h>
int i,err;
char inp[10];
void E();
void E1();
void T();
void T1();
void F();
void main()
{
    i=0,err=0;
    printf("Enter String: ");
    scanf("%s",inp);
    E();
    if(strlen(inp)==i && err==0)
        printf("\t\t*****STRING ACCEPTED!!!!!!!!!!!!*****\n\n");
    else
        printf("\t\t*****STRING NOT ACCEPTED!!!!*****\n\n");
}
void E()
{
    T();
    E1();
}
void E1()
{
    if(inp[i]=='+' )
```

```

        {
            i++;
            T();
            E1();
        }
    }
void T()
{
    F();
    T1();
}
void T1()
{
    if(inp[i]=='*')
    {
        i++;
        F();
        T1();
    }
}
void F()
{
    if(isalnum(inp[i]))
        i++;
    else if(inp[i]=='(')
    {
        i++;
        E();
        if(inp[i]==')')
            i++;
        else
            err=1;
    }
    else
        err=1;
}

```

**OUTPUT:** > ./a.out

**Enter Expression: 5+3\*7**

**\*\*\*\*\*STRING ACCEPTED!!!!!!!!!!\*\*\*\*\***

**> ./a.out**

**Enter Expression: 5\*(6+8)**

**\*\*\*\*\*STRING ACCEPTED!!!!!!!!!!\*\*\*\*\***

**> ./a.out**

**Enter Expression: 5\*(6+\*)**

**\*\*\*\*\*STRING NOT ACCEPTED!!!!\*\*\*\*\***

**>:~\$ ./a.out**

**Enter Expression: 5\*(6+8(**

**\*\*\*\*\*STRING NOT ACCEPTED!!!!\*\*\*\*\***

### **PROGRAM NO:9**

**Object** : Write a program to Design LALR Bottom up Parser.

**/\*LALR PARSER**

**E->E+T**

**E->T**

**T->T\*F**

**T->F**

**F->(E)**

**F->i**

**\*/**

**#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"},
{"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;
clrscr();
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");
printr(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++;
27
}
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,char inp[],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;  
    }  
}
```

## PROGRAM NO:10

Object: Write a program to convert NFA to DFA

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

```
int Fa[10][10][10],states[2][10],row=0,col=0,sr=0,sc=0,th=0,
in,stat,new_state[10][10],max_inp=-1,no_stat;
FILE *fp;
```

```
int search(int search_var)
```

```
{
```

```
    int i;
```

```
    for(i=0;i<no_stat;i++)
```

```
        if(search_var == states[1][i])
```

```
            return 1;
```

```
    return 0;
```

```
}
```

```
int sort(int *arr,int count)
```

```
{
```

```
    int temp,i,j;
```

```
    for(i=0;i<count-1;i++)
```

```
    {
```

```
        for(j=i+1;j<count;j++)
```

```
        {
```

```
            if(arr[i]>=arr[j])
```

```
            {
```

```
                temp=arr[i];
```

```
                arr[i]=arr[j];
```

```
                arr[j]=temp;
```

```
            }
```

```
        }
```

```
    }
```

```
    return 0;
```

```
}
```

```
int checkcon(int *arr,int *count) //for doing this {4,1}={1,2,1}=={1,2}
```

```
{
```

```

int i,temp,j,k,c,t,m;
for(i=0;i<*count;i++)
{
    if(arr[i]>row)
    {
        temp =arr[i];
        c=0;
        t=0;
        while(new_state[arr[i]][t]!=-1)
        {
            t++;
            c++;
        }
        //right shift from ith postion (c-2) th time
        for(k=0;k<=c-2;k++)
        {
            for(j=9;j>=i+1+k;j--)
            {
                arr[j]=arr[j-1];
            }
        }

        t=0;
        for(j=i;j<c;j++)
        {
            arr[j]=new_state[temp][t];
            t++;
        }
    }
    c=0;
    for(i=0;arr[i]!=-1;i++)
        c++;
    *count=c;
    return 0;
}

```

```

int remove_duplicate(int *arr,int *count)
{
    int i,j=0;
    for(i=1;i<*count;i++)

```

```

{
    if(arr[i]!=arr[j])
    {
        j++;
        arr[j]=arr[i];
    }
}
*count=j+1;
return 0;
}

```

int check(int i ,int j,int c,int \*name)*//for checking is this a new state?*

```

{
    int t,l,f;
    for(l=0;l<=stat;l++)
    {
        t=0; f=0;
        while(Fa[i][j][t]!=-1)
        {
            if(Fa[i][j][t]==new_state[l][t])
                t++;
            else
            {
                f=1;
                break;
            }
        }
        if((t==c)&&!f)
        {
            *name=l;
            return 1;
        }
    }
    return 0;
}

```

int trans(int i ,int j,int t,int c,int \*count,int \*arr)*//transition o/p for particular i/p on states*

```

{
    int k=0,co,temp;
    *count=0;

```



```

        for(k=0;k<10;k++)
            arr[k]=-1;
        trans(i,j,t,c,&count,arr);

        checkcon(arr,&count);

        sort(arr,count);
        remove_duplicate(arr,&count);

        for(k=0;k<count;k++)
            Fa[stat][t][k]=arr[k];
    }
    Fa[i][j][0]=stat++;
    for(t=1;t<c;t++)
        Fa[i][j][t]=-1;
    }
    else
    {
        Fa[i][j][0]=name ;
        for(t=1;t<c;t++)
            Fa[i][j][t]=-1;
    }
}

}

return 0;
}
int main()
{

    int i,j,k,flag=0,start,end;
    char c,ch;
    fp=fopen("Nfa_ip.txt","r+");

    for(i=0;i<2;i++)
        for(j=0;j<10;j++)
            states[i][j]=-1;

```

```

for(i=0;i<10;i++)
    for(j=0;j<10;j++)
        new_state[i][j]=-1;

for(i=0;i<10;i++)
    for(j=0;j<10;j++)
        for(k=0;k<10;k++)
            Fa[i][j][k]=-1;

while(fscanf(fp,"%d",&in)!=EOF)
{
    fscanf(fp,"%c",&c);

    if(flag)
    {
        states[sr][sc++]=in;
        if(c=='\n')
        {
            sr++;
            sc=0;
        }
    }
    else if(c=='#')
    {
        flag=1;
        Fa[row][col][th]=in;

    }
    else if(!flag)
    {
        Fa[row][col][th]=in;
        if(c==',')
        {
            th++;
        }
        else if(c=='\n')
        {
            if(max_inp<col)
                max_inp=col;
            col=0;
            row++;
        }
    }
}

```

```

        th=0;
    }
    else if(c!='')
    {
        col++;
        th=0;
    }
}

}
no_stat=0;
i=0;
while(states[1][i++]!=-1)
    no_stat++;
    stat=row+1;
    start=0;end=row;
    while(1)
    {
        nfa2dfa(start,end);
        start=end+1;
        end=row;
        if(start>end)
            break;
    }

printf("\n\nDFA IS : \n\n");
for(i=0;i<=max_inp;i++)
    printf("\t%d",i);
printf("\n");
printf("-----\n");

for(i=0;i<stat;i++)
    {
        printf("%d-> |",i);
        for(j=0;j<=max_inp;j++)
        {
            printf("%2d      ",Fa[i][j][0]);
        }
        printf("\n");
    }
printf("\n\n");

```



```

printf("Total Number Of State Is : %d \n\n",stat);
printf("Final States Are : ");
    for(i=0;states[1][i]!=-1;i++)
        printf("%d ",states[1][i]);

printf("\n\n");
getch();
    return 0;
}

```

### **PROGRAM NO:11**

**OBJECT:**To write a C program to implement the shift-reduce parsing algorithm

```

#include<conio.h>
#include<stdio.h>
#include<stdlib.h>
#include<string.h>
char ip_sym[15],stack[15];
int ip_ptr=0,st_ptr=0,len,i;
char temp[2],temp2[2];
char act[15];
void check();
void main()
{
clrscr();
printf("\n\n\t Shift Reduce Parser\n");
printf("\n\t***** ***** *****");
printf("\n Grammar\n\n");
printf("E->E+E\nE->E/E\n");
printf("E->E*E\nE->a/b");
printf("\n Enter the Input Symbol:\t");
gets(ip_sym);
printf("\n\n\t Stack Implementation Table");
printf("\n Stack\t\t Input Symbol\t\t Action");
printf("\n $\t\t %s$\t\t --",ip_sym);
strcpy(act,"shift");
temp[0]=ip_sym[ip_ptr];
temp[1]='\0';
strcat(act,temp);
len=strlen(ip_sym);

```

```

for(i=0;i<=len-1;i++)
{
stack[st_ptr]=ip_sym[ip_ptr];
stack[st_ptr+1]='\0';
ip_sym[ip_ptr]=' ';
ip_ptr++;
printf("\n$%s\t\t%s$\t\t\t%s",stack,ip_sym,act);
strcpy(act,"shift");
temp[0]=ip_sym[ip_ptr];
temp[1]='\0';
strcat(act,temp);
check();
st_ptr++;
}
st_ptr++;
check();
getch();
}

```

```

void check()
{
int flag=0;
temp2[0]=stack[st_ptr];
temp[1]='\0';
if((!strcmpi(temp2,"a"))||(!strcmpi(temp2,"b")))
{
stack[st_ptr]='E';
if(!strcmpi(temp2,"a"))
printf("\n$%s\t\t%s$\t\t\tE->a",stack,ip_sym);
else
printf("\n$%s\t\t%s$\t\t\tE->a",stack,ip_sym);
flag=1;
}
if((!strcmpi(temp2,"+"))||(strcmpi(temp2,"*"))||(!strcmpi(temp2,"/")))
{
flag=1;
}
if((!strcmpi(stack,"E+E"))||(!strcmpi(stack,"E/E"))||(!strcmpi(stack,"E*E")))
{
strcpy(stack,"E");
st_ptr=0;
}

```

```

if(!strcmpi(stack,"E+E"))
printf("\n%s\t\t%s$\t\tE->E+E",stack,ip_sym);
else
if(!strcmpi(stack,"E/E"))
printf("\n%s\t\t%s$\t\tE->E/E",stack,ip_sym);
else
printf("\n%s\t\t%s$\t\tE->E*E",stack,ip_sym);
flag=1;
}
if(!strcmpi(stack,"E")&&ip_ptr==len)
{
printf("\n%s\t\t%s$\t\tAccept",ip_sym);
getch();
exit(0);
}
if(flag==0)
{
printf("\n %s \t\t %s \t\t Reject",stack,ip_sym);
}
return;
}

```

### Sample Input & Output:

Shift Reduce Parser

\*\*\*\*\*

Grammar

E->E+E

E->E/E

E->E\*E

E->a/b

Enter the input symbol: if(a\*b)

Stack Implementation Table

Stack	Input Symbol	Action
\$	if(a*b)\$	--
\$i	f(a*b)\$	shift i
\$if	(a*b)\$	shift f
\$if(	a*b)\$	shift (
\$if(a	*b)\$	shift a
\$if(E	*b)\$	E->a
\$if(E*	b)\$	shift *
if(E*	b)	reject

## **PROGRAM NO:12**

**Object** : Write a C program to generate machine code from abstract syntax tree generated by the parser. The instruction set specified may be considered as the target code.

Consider the following mini language, a simple procedural high –level language, only operating on integer data, with a syntax looking vaguely like a simple C crossed with pascal. The syntax of the language is defined by the following grammar.

```
<program>::=<block>
<block>::={ <variable definition><slist> }
|{ <slist> }
<variabledefinition>::=int <vardeflist>
<vardec>::=<identifier>|<identifier>[<constant>]
<slist>::=<statement>|<statement>;<slist>
<statement>::=<assignment>|<ifstatement>|<whilestatement>
|<block>|<printstatement>|<empty>
<assignment>::=<identifier>=<expression>
|<identifier>[<expression>]=<expression>
<if statement>::=if<bexpression>then<slist>else<slist>endif
|if<bexpression>then<slisi>endif
<whilestatement>::=while<bexpression>do<slisi>enddo
<printstatement>::=print(<expression>)
<expression>::=<expression>::=<expression><addingop><term>|<term>|<addingop>
p>
<term>
<bexprssion>::=<expression><relop><expression>
<relop>::=<|<=<|==<|>=<|>|!=
<addingop>::=+|-
<term>::=<term><multop><factor>|<factor>
<Multop>::=*/
<factor>::=<constant>|<identifier>|<identifier>[<expression>]
|(<expression>)
<constant>::=<digit>|<digit><constant>
<identifier>::=<identifier><letter or digit>|<letter>
<letter or digit>::=<letter>|<digit>
```

<letter>::=a|b|c|d|e|f|g|h|i|j|k|l|m|n|o|p|q|r|s|t|u|v|w|x|y|z  
<digit>::=0|1|2|3|4|5|^|7|8|9  
<empty>::=has the obvious meaning

```
#include<stdio.h>
#include<stdlib.h>
#include<string.h>
int label[20];
int no=0;
int main()
{
FILE *fp1,*fp2;
char fname[10],op[10],ch;
char operand1[8],operand2[8],result[8];
int i=0,j=0;
printf("\n Enter filename of the intermediate code");
scanf("%s",&fname);
fp1=fopen(fname,"r");
fp2=fopen("target.txt","w");
if(fp1==NULL || fp2==NULL)
{
printf("\n Error opening the file");
exit(0);
} w
while(!feof(fp1))
{
45
fprintf(fp2,"\n"); fscanf(fp1,"%s",op);
i++; if(check_label(i))
fprintf(fp2,"\nlabel#%d",i);
if(strcmp(op,"print")==0)
{
fscanf(fp1,"%s",result);
fprintf(fp2,"\n\t OUT %s",result);
}
if(strcmp(op,"goto")==0)
{
fscanf(fp1,"%s %s",operand1,operand2);
fprintf(fp2,"\n\t JMP %s,label#%s",operand1,operand2);
label[no++]=atoi(operand2);
} i
```

```

f(strcmp(op,"[]")==0)
{
fscanf(fp1,"%s %s %s",operand1,operand2,result);
fprintf(fp2,"\n\t STORE %s[%s],%s",operand1,operand2,result);
}
if(strcmp(op,"uminus")==0)
{ fs
canf(fp1,"%s %s",operand1,result);
fprintf(fp2,"\n\t LOAD -%s,R1",operand1);
fprintf(fp2,"\n\t STORE R1,%s",result);
}
switch(op[0])
{
case '*': fscanf(fp1,"%s %s %s",operand1,operand2,result);
fprintf(fp2,"\n\t
LOAD",operand1);
fprintf(fp2,"\n\t LOAD
%s,R1",operand2);
fprintf(fp2,"\n\t MUL R1,R0");
fprintf(fp2,"\n\t STORE
R0,%s",result);
break;
case '+': fscanf(fp1,"%s %s %s",operand1,operand2,result);
fprintf(fp2,"\n\t LOAD %s,R0",operand1);
fprintf(fp2,"\n\t LOAD %s,R1",operand2);
fprintf(fp2,"\n\t ADD R1,R0");
fprintf(fp2,"\n\t STORE R0,%s",result);
break;
case '-': fscanf(fp1,"%s %s %s",operand1,operand2,result); fprintf(fp2,"\n\t LOAD
%s,R0",operand1);
fprintf(fp2,"\n\t LOAD %s,R1",operand2); fprintf(fp2,"\n\t SUB R1,R0");
fprintf(fp2,"\n\t STORE R0,%s",result);
break;
case '/': fscanf(fp1,"%s %s %s",operand1,operand2,result);
fprintf(fp2,"\n\t LOAD %s,R0",operand1);
fprintf(fp2,"\n\t LOAD %s,R1",operand2);
fprintf(fp2,"\n\t DIV R1,R0");
fprintf(fp2,"\n\t STORE R0,%s",result);
break;
case '%': fscanf(fp1,"%s %s %s",operand1,operand2,result);
fprintf(fp2,"\n\t LOAD %s,R0",operand1);

```

```

fprintf(fp2, "\n \t LOAD %s,R1",operand2);
fprintf(fp2, "\n \t DIV R1,R0");
fprintf(fp2, "\n \t STORE R0,%s",result);
break;
case '=': fscanf(fp1,"%s %s",operand1,result);
fprintf(fp2, "\n\t STORE %s %s",operand1,result);
break;
case '>': j++;
fscanf(fp1,"%s %s %s",operand1,operand2,result);
fprintf(fp2, "\n \t LOAD %s,R0",operand1);
fprintf(fp2, "\n\t JGT %s,label#%s",operand2,result);
label[no++]=atoi(result);
break;
case '<': fscanf(fp1,"%s %s %s",operand1,operand2,result);
fprintf(fp2, "\n \t LOAD %s,R0",operand1); fprintf(fp2, "\n\t
JLT %s,label#%d",operand2,result);
label[no++]=atoi(result);
break;
}
}
fclose(fp2); fclose(fp1);
fp2=fopen("target.txt","r");
if(fp2==NULL)
{
printf("Error opening the file\n");
exit(0);
}
do
{
ch=fgetc(fp2);
printf("%c",ch);
}while(ch!=EOF);
fclose(fp1);
return 0;
47
} i
nt check_label(int k)
{ i
nt i;
for(i=0;i<no;i++)
{

```

```
if(k==label[i])  
    return 1;  
}  
return 0;  
}
```