COMPILER DESIGN

(SUBJECT CODE: 3170701)

LAB MANUAL

Enrollment No: 180060107042

Class: 4th year 7th Semester (CE)

Session: Jun 2021 – Oct 2021

Faculty Name: Ms. Nensi panchal



**Department of Computer Science & Engineering**

**BHAGWAN MAHAVIR COLLEGE OF ENGINEERING AND TECHNOLOGY**

**Surat - 395 017, Gujarat.**

**CERTIFICATE**

**Bhagwan Mahavir collage of Engineering and Technology, Surat**

****

This is to certify that ~~Mr.~~ / Ms. **Riya Patel** of class **Computer Science and Engineering (7th semester).** Enrollment No. **180060107042** has satisfactorily submitted ~~his~~ / her work in subject **COMPILER DESIGN** for the term ending in **2021.**

Date:

Sign of teacher Sign of Head of department

**INDEX**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **No.** | **PRACTICAL** | **PAGE NO.** | **DATE** | **SIGN** |
| 1 | Implementation of Finite Automata and String Validation | 5 |  |  |
| 2 | Introduction to Lex Tool. | 9 |  |  |
| 3 | Implement following Programs Using Lex  a. Generate Histogram of words  b. Caesar Cypher  c. Extract single and multiline comments from C Program | 13 |  |  |
| 4 | Implement following Programs Using Lex  a. Convert Roman to Decimal  b. Check weather given statement is compound or simple  c. Extract html tags from .html file | 17 |  |  |
| 5 | Implementation of Recursive Descent Parser without backtracking  Input: The string to be parsed.  Output: Whether string parsed successfully or not. Explanation:  Students have to implement the recursive procedure for RDP for a typical grammar. The  production no.is displayed as they are used to derive the string. | 23 |  |  |
| 6 | Finding “First” set  Input: The string consists of grammar symbols.  Output: The First set for a given string. | 28 |  |  |
| 7 | Generate 3-tuple intermediate code for given infix expression | 37 |  |  |
| 8 | Extract Predecessor and Successor from given Control Flow Graph | 41 |  |  |
| 9 | Introduction to YACC and generate Calculator Program | 44 |  |  |
| 10 | Finding “Follow” set  Input: The string consists of grammar symbols.  Output: The Follow set for a given string. | 46 |  |  |
| 11 | Implement a C program for constructing LL (1) parsing | 50 |  |  |
| 12 | Implement a C program to implement LALR parsing. | 59 |  |  |
| 13 | Implement a C program to implement operator precedence parsing. | 68 |  |  |

**Practical - 1**

**Aim**: Implementation of Finite Automata and String Validation.

**Code:**

#include <stdio.h>

#include <string.h>

int dfa = 0;

void start(char c)

{

if (c == 'a')

{

dfa = 1;

}

else if (c == 'b')

{

dfa = 3;

}

else

{

dfa = -1;

}

}

void state1(char c)

{

if (c == 'a')

{

dfa = 2;

}

else if (c == 'b')

{

dfa = 4;

}

else

{

dfa = -1;

}

}

void state2(char c)

{

if (c == 'b')

{

dfa = 3;

}

else if (c == 'a')

{

dfa = 1;

}

else

{

dfa = -1;

}

}

void state3(char c)

{

if (c == 'b')

{

dfa = 3;

}

else if (c == 'a')

{

dfa = 4;

}

else

{

dfa = -1;

}

}

void state4(char c)

{

dfa = -1;

}

int isAccepted(char str[])

{

int i, len = strlen(str);

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

{

if (dfa == 0) start(str[i]);

else if (dfa == 1) state1(str[i]);

else if (dfa == 2) state2(str[i]);

else if (dfa == 3) state3(str[i]);

else if (dfa == 4) state4(str[i]);

else

return 0;

}

if (dfa == 3) return 1;

else

return 0;

}

int main()

{

int i;

char str[] = "aaaaaabbbb";

printf("String: %s\n", str);

if (isAccepted(str))

printf("ACCEPTED");

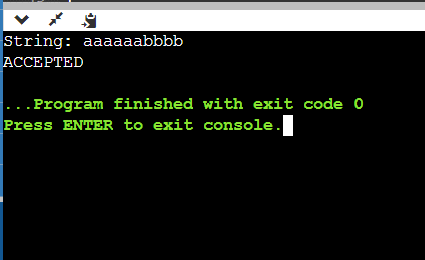
else

printf("NOT ACCEPTED");

return 0;

}

**Output:**

****

**Practical - 2**

**Aim:** Introduction to Lex Tool.

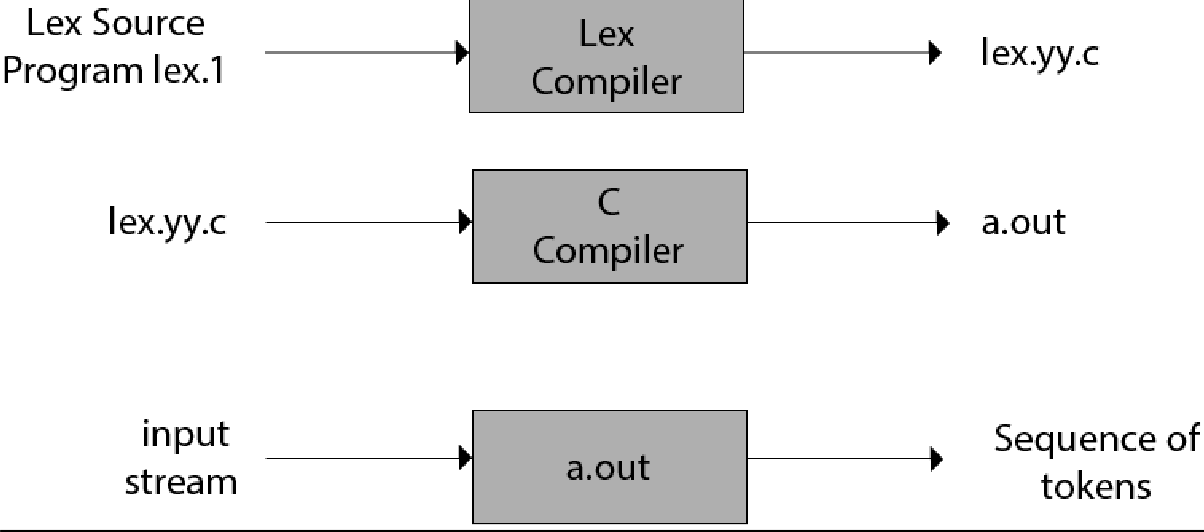
**Code:**

**Introduction:**

* Lex is a program that generates lexical analyzer. It is used with YACC parser generator.
* The lexical analyzer is a program that transforms an input stream into a Sequence of tokens.
* It reads the input stream and produces the source code as output through implementing the lexical analyzer in the C program.

**The Function of Lex:**

* Firstly lexical analyzer creates a program lex.1 in the Lex language. Then Lex compiler runs the lex.1 program and produces a C program lex.yy.c.
* Finally C compiler runs the lex.yy.c program and produces an object program a.out.

a.out is lexical analyzer that transforms an input stream into a sequence of tokens.

The structure of LEX programs:

A LEX program consists of three sections: Declarations, Rules and Auxiliary functions.

Declarations

%%

Translation rules

%%

Auxiliary functions

* Declarations: This section includes declaration of variables, constants.
* Translation rules: It contains regular expressions and code segments.
* Form: Pattern {Action}
* Pattern is a regular expression or regular definition.
* Action refers to segments of code.

How to run an LEX program

• To run the program, it should be first saved with the extension .l or .lex.

• Run the below commands on terminal in order to run the program file.

Step 1: lex filename.l or lex filename.lex depending on the extension file is saved with

Step 2: gcc lex.yy.c Step 3: ./a.out or a.exe

Step 4: Provide the input to program in case it is required

Built in Function and Variables

BEGIN:

Switch start condition.

ECHO:

ECHO is a macro that writes code matched by the pattern.

This is the default action for any unmatched strings.

Syntax: [a-z]+ ECHO;

• yytext:

o A buffer that holds the input characters that actually match the pattern (i.e. lexeme) or say a pointer to the matched string.

o Syntax: [a-z]+ printf(“%s”, yytext);

• yylex():

o Implies the main entry point for lex, reads the input stream generates tokens, returns zero at the end of input stream.

o It is called to invoke the lexer (or scanner) and each time yylex() is called, the scanner continues processing the input from where it last left off.

o Example:

/\* Declarations \*/

%%

{number} {return atoi(yytext);}

%%

int main()

{

}

• yywrap():

int num = yylex(); printf("Found: %d", num); return 1;

o It is called by lex when input is exhausted (or at EOF).

o Default yywrap always return 1.

o Syntax:

int yywrap(void)

{

• yyin:

return 1;

}

o yyin is a variable of the type FILE\* and points to the input

file. yyin is defined by LEX automatically. If the programmer assigns an input file to yyin in the auxiliary functions section, then yyin is set to point to that file. Otherwise LEX assigns yyin to stdin (console input).

**Practical - 3**

**Aim:** Implement following Programs Using Lex.

a. Generate Histogram of words

b. Ceasor Cypher

c. Extract single and multiline comments from C Program

**code:**

a. Generate Histogram of words

%{

#include<stdio.h>

#include<string.h>

int i = 0;

%}

%%

([a-zA-Z0-9])\* {i++;}

"\n" { printf("%d\n", i); i = 0; }

%%

int yywrap()

{

return 1;

}

int main()

{

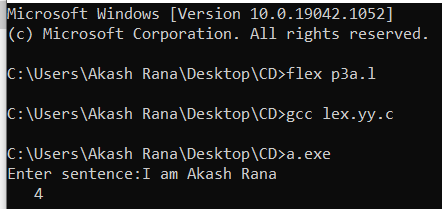
printf("Etner sentence:");

yylex();

return 0;

}

**output:**

****

b. Ceasor Cypher

**code:**

%{

#include<stdio.h>

%}

%%

[a-z]

{

char c = yytext[0];

c +=4;

if(c > 'z')

c -= ('z'+1- 'a');

printf("%c",c);

}

[A-Z]

{

char c = yytext[0];

c +=4;

if(c>'Z')

c -= ('Z'+1- 'A');

printf("%c" ,c);

}

%%

int main()

{

printf("Enter the sentence\n");

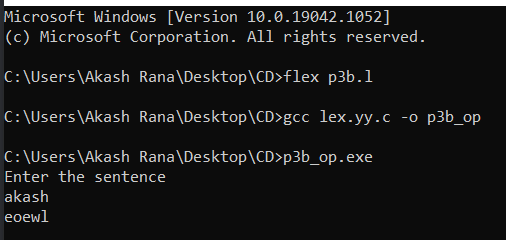
yylex();

}

int yywrap()

{ return1;}

**output:**



c. Extract single and multiline comments from C Program

**code:**

%{

#include<stdio.h>

%}

%%

\/\/(.\*) { fprintf(yyout,"%s\n",yytext); printf("%s\n",yytext); };

\/\\*(.\*\n)\*.\*\\*\/ { fprintf(yyout,"%s\n",yytext); printf("%s\n",yytext); };

%%

int yywrap()

{return 1;}

int main()

{

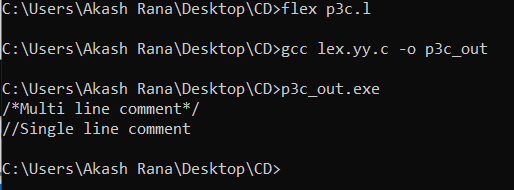
FILE \*fp, \*fo; fp=fopen("p3c.c","r");

fo=fopen("p3c\_out.txt","w"); yyin=fp;

yyout=fo; yylex(); return0;

}

**output:**



**Practical - 4**

**Aim:** Implement following Programs Using Lex

a. Convert Roman to Decimal

b. Check weather given statement is compound or simple

c. Extract html tags from .html file

**Code:**

a. Convert Roman to Decimal

%{

#include<stdio.h>

%}

%%

"I" {printf(" Answer : %s = 1 \n",yytext);} "II" {printf(" Answer : %s = 2 \n",yytext);} "III" {printf(" Answer : %s = 3 \n",yytext);} "IV" {printf(" Answer : %s = 4 \n",yytext);} "V" {printf(" Answer : %s = 5 \n",yytext);} "VI" {printf(" Answer : %s = 6 \n",yytext);} "VII" {printf(" Answer : %s = 7\n",yytext);} "VIII" {printf(" Answer : %s = 8\n",yytext);} "IX" {printf(" Answer : %s = 9 \n",yytext);} "X" {printf(" Answer : %s = 10 \n",yytext);}

. {;}

%%

int main()

{

printf("\nEnter Roman Number to convert it into Decimal:"); yylex();

return 0;

}

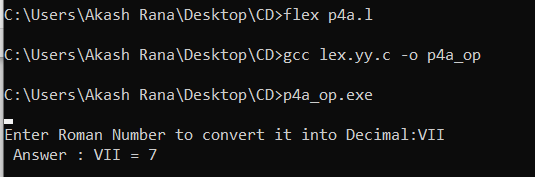
int yywrap()

{

return 0;

}

# Output:



b. Check weather given statement is compound or simple

**Code:**

%{

int flag=0;

%}

%%

(""[aA][nN][dD]"") {flag=1;}

(""[oO][rR]"") {flag=1;}

(""[bB][uU][tT]"") {flag=1;}

%%

int main()

{

printf("Enter the sentence\n");yylex(); if(flag==1) printf("\nCompound sentence\n");else printf("\nSimple sentence\n");

return 0;

}

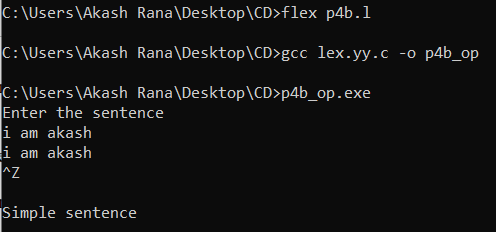
int yywrap()

{

return1;

}

# Output:



c. Extract html tags from .html file

**Code:**

%{

#include <stdio.h>

%}

%%

"<"[^>]\*">" { fprintf(yyout,"%s\n",yytext); printf("%s\n",yytext);}

.;

\n

%%

int yywrap()

{

return 1;

}

main()

{ FILE \*fp, \*fo; fp=fopen("p4c.html","r"); fo=fopen("p4c\_out.txt","w")

;yyin=fp; yyout=f o; yylex(); return

0;

}

p4c.html:

<html>

<head></head>

<title></title>

<hr>

<body>

<h1>H1 tag</h1><hr>

<h2> H2tag</h2><hr>

<h3>H3 tag </h3><hr>

<h4></h4>

<br>

<h5></h5><hr>

<h6></h6><hr>

<p>Praegraph tag</p>

</body>

<!...Comment section...>

</html>

p4c\_out.txt:

<html>

<head>

</head>

<title>

</title>

<hr>

<body>

<h1>

</h1>

<hr>

<h2>

</h2>

<hr>

<h3>

</h3>

<hr>

<h4>

</h4>

<br>

<h5>

</h5>

<hr>

h6>

</h6>

<hr>

<p>

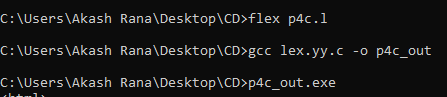
</p>

</body>

<!...Comment section...>

</html>

**Output:**



**Practical - 5**

**Aim:** Implementation of Recursive Descent Parser without backtracking

Input: The string to be parsed.

Output: Whether string parsed successfully or not.

Explanation: Students have to implement the recursive procedure for RDP for a typical grammar. The production no. are displayed as they are used to derive the string.

**Code:**

#include<stdio.h> #include<conio.h> #include<string.h> int E();//E->TE'

int ED(); //E'->+TE'|@ int T();//T->FT'

int TD(); //T'->\*FT'|@ int F(); //F->(E)|ID charinput[100];

int i,l;

void main()

{

printf("\nRecursive Descent Parsing for the following Grammar\n");

printf("\nE->TE' \nE'->+TE'|@ \nT->FT' \nT'->\*FT'|@ \nF->(E)|ID\n");

printf("\nProduction Number considered:\n");

printf("\n1. E->TE' \n2. E'->+TE' \n3. E'->@ \n4. T->FT' \n5. T'->\*FT'\n6. T'->@ \n7. F- >(E)\n8. F->ID \n");

printf("\nEnter the String to be checked: ");

gets(input);

if(E())

{

if(input[i+1] == '\0')

printf("\n\nString is parsed successfully");

else

printf("\n\nString is not parsed successfully");

}

else

printf("\n\nString not parsed successfully");

getch();

}

intE()

{

if(T())

{

if(ED())

{

printf("\nProduction No. 1 used:E->TE' ");

return1;

}

else return0;

}

else return0;

}

intED()

{

if(input[i]=='+')

{

i++;

if(T())

{

if(ED())

{

printf("\nProduction No. 2 used:T'->+TE'");

return1;

}

else return0;

}

else return0;

}

else

{

printf("\nProduction No. 3 used:E'->@");

return 1;

}

}

intT()

{

if(F())

{

if(TD())

{

printf("\nProduction No. 4 used:T->FT'");

return1;

}

else return0;

}

else return0;

}

intTD()

{

if(input[i]=='\*')

{

i++;

if(F())

{

if(TD())

{

printf("\nProduction No. 5 used:T'->\*FT'");

return1;

}

else return0;

}

else return0;

}

else

{

printf("\nProduction No. 6 used:T'->@");

}

}

F()

{

if(input[i]=='(')

{

i++;

if(E())

{

if(input[i]==')')

{

i++;

printf("\nProduction no. 7 used:F->(E)");

return(1);

}

else return(0);

}

else return(0);

}

else if(input[i]>='a'&&input[i]<='z'||input[i]>='A'&&input[i]<='Z')

{

i++;

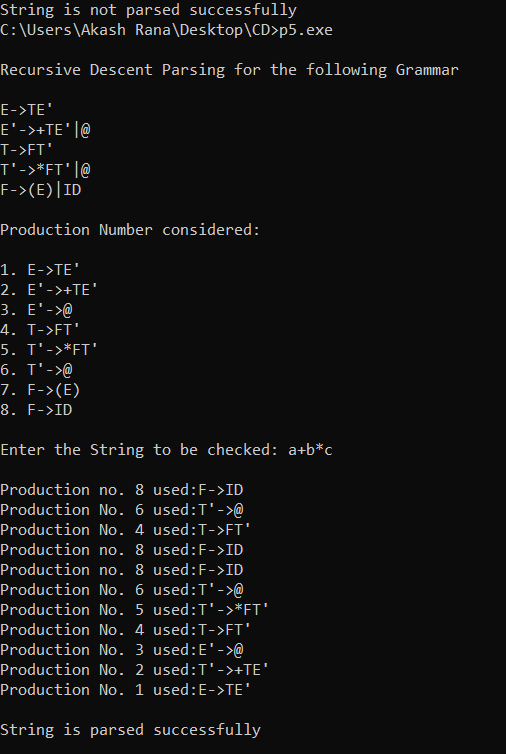
printf("\nProduction no. 8 used:F->ID");

return(1);

}

else return(0);

}

**Output:**

**Practical - 6**

**Aim:** Finding “First” set

Input: The string consists of grammar symbols.

Output: The First set for a given string.

Explanation: The student has to assume a typical grammar. The program when run will ask for the string to be entered. The program will find the First set of the given string.

**Code:**

#include<stdio.h>

#include<string.h>

#define TSIZE 128

struct product

{

char str[100]; int len;

}pro[20];

char first[26][TSIZE]; char follow[26][TSIZE];

isNT(char c) {

return c >= 'A' && c <= 'Z';

}

readFromFile() {

FILE\* fptr;

fptr = fopen("LL1.txt", "r");

char buffer[255]; int i;

int j;

while (fgets(buffer, sizeof(buffer), fptr)) { printf("%s", buffer);

j = 0;

nonterminal[buffer[0] - 'A'] = 1;

for (i = 0; i < strlen(buffer) - 1; ++i) { if (buffer[i] == '|') {

++no\_pro;

pro[no\_pro - 1].str[j] = '\0'; pro[no\_pro - 1].len = j;

pro[no\_pro].str[0] = pro[no\_pro - 1].str[0]; pro[no\_pro].str[1] = pro[no\_pro - 1].str[1]; pro[no\_pro].str[2] = pro[no\_pro - 1].str[2]; j = 3;

}

else {

pro[no\_pro].str[j] = buffer[i];

++j;

if (!isNT(buffer[i]) && buffer[i] != '-' && buffer[i] != '>') { terminal[buffer[i]] = 1;

}

}

}

pro[no\_pro].len = j;

++no\_pro;

}

}

void add\_FIRST\_A\_to\_FOLLOW\_B(char A, char B) { int i;

for (i = 0; i < TSIZE; ++i) {

if (i != '^')

follow[B - 'A'][i] = follow[B - 'A'][i] || first[A - 'A'][i];

}

}

void add\_FOLLOW\_A\_to\_FOLLOW\_B(char A, char B) { int i;

for (i = 0; i < TSIZE; ++i) {

if (i != '^')

follow[B - 'A'][i] = follow[B - 'A'][i] || follow[A - 'A'][i];

}

}

void FOLLOW() { int t = 0;

int i, j, k, x;

while (t++ < no\_pro)

{

for (k = 0; k < 26; ++k) {

if (!nonterminal[k]) continue; char nt = k + 'A';

for (i = 0; i < no\_pro; ++i) { for (j = 3; j < pro[i].len; ++j) {

if (nt == pro[i].str[j]) {

for (x = j + 1; x < pro[i].len; ++x) { char sc = pro[i].str[x];

if (isNT(sc)) { add\_FIRST\_A\_to\_FOLLOW\_B(sc, nt); if (first[sc - 'A']['^'])

continue;

}

else {

follow[nt - 'A'][sc] = 1;

}

break;

}

if (x == pro[i].len) add\_FOLLOW\_A\_to\_FOLLOW\_B(pro[i].str[0], nt);

}

}

}

}

}

}

void add\_FIRST\_A\_to\_FIRST\_B(char A, char B) { int i;

for (i = 0; i < TSIZE; ++i) {

if (i != '^') {

first[B - 'A'][i] = first[A - 'A'][i] || first[B - 'A'][i];

}

}

}

void FIRST() { int i, j;

int t = 0;

while (t < no\_pro) {

for (i = 0; i < no\_pro; ++i) { for (j = 3; j < pro[i].len; ++j) { char sc = pro[i].str[j];

if (isNT(sc)) {

add\_FIRST\_A\_to\_FIRST\_B(sc, pro[i].str[0]); if (first[sc - 'A']['^'])

continue;

}

else {

first[pro[i].str[0] - 'A'][sc] = 1;

}

break;

}

if (j == pro[i].len) first[pro[i].str[0] - 'A']['^'] = 1;

}

++t;

}

}

void add\_FIRST\_A\_to\_FIRST\_RHS

B(char A, int B) { int i;

for (i = 0; i < TSIZE; ++i) {

if (i != '^')

first\_rhs[B][i] = first[A - 'A'][i] || first\_rhs[B][i];

}

}

FIRST(ß) for each A->ß void FIRST\_RHS() {

int i, j; int t = 0;

while (t < no\_pro) {

for (i = 0; i < no\_pro; ++i) { for (j = 3; j < pro[i].len; ++j) { char sc = pro[i].str[j];

if (isNT(sc)) { add\_FIRST\_A\_to\_FIRST\_RHS B(sc, i); if (first[sc - 'A']['^'])

continue;

}

else { first\_rhs[i][sc] = 1;

}

break;

}

if (j == pro[i].len) first\_rhs[i]['^'] = 1;

}

++t;

}

}

int main() { readFromFile();

follow[pro[0].str[0] - 'A']['$'] = 1; FIRST();

FOLLOW();

FIRST\_RHS();

int i, j, k;

for (i = 0; i < no\_pro; ++i) {

if (i == 0 || (pro[i - 1].str[0] != pro[i].str[0])) { char c = pro[i].str[0];

printf("FIRST OF %c: ", c); for (j = 0; j < TSIZE; ++j) {

if (first[c - 'A'][j]) { printf("%c ", j);

}

}

printf("\n");

}

}

for (i = 0; i < no\_pro; ++i) {

if (i == 0 || (pro[i - 1].str[0] != pro[i].str[0])) { char c = pro[i].str[0];

printf("FOLLOW OF %c: ", c); for (j = 0; j < TSIZE; ++j) {

if (follow[c - 'A'][j]) { printf("%c ", j);

}

}

printf("\n");

}

}

for (i = 0; i < no\_pro; ++i) { printf("FIRST OF %s: ", pro[i].str); for (j = 0; j < TSIZE; ++j) {

if (first\_rhs[i][j]) {

printf("%c ", j);

}

}

printf("\n");

}

terminal['$'] = 1;

printf("\n\t\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* LL(1) PARSING TABLE \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\n"); printf("\t- \n");

printf("%-10s", "");

for (i = 0; i < TSIZE; ++i) {

if (terminal[i]) printf("%-10c", i);

}

printf("\n"); int p = 0;

for (i = 0; i < no\_pro; ++i) {

if (i != 0 && (pro[i].str[0] != pro[i - 1].str[0])) p = p + 1;

for (j = 0; j < TSIZE; ++j) {

if (first\_rhs[i][j] && j != '^') { table[p][j] = i + 1;

}

else if (first\_rhs[i]['^']) { for (k = 0; k < TSIZE; ++k) {

if (follow[pro[i].str[0] - 'A'][k]) { table[p][k] = i + 1;

}

}

}

}

}

k = 0;

for (i = 0; i < no\_pro; ++i) {

if (i == 0 || (pro[i - 1].str[0] != pro[i].str[0])) {

printf("%-10c", pro[i].str[0]); for (j = 0; j < TSIZE; ++j) {

if (table[k][j]) {

printf("%-10s", pro[table[k][j] - 1].str);

}

else if (terminal[j]) { printf("%-10s", "");

}

}

++k;

printf("\n");

}

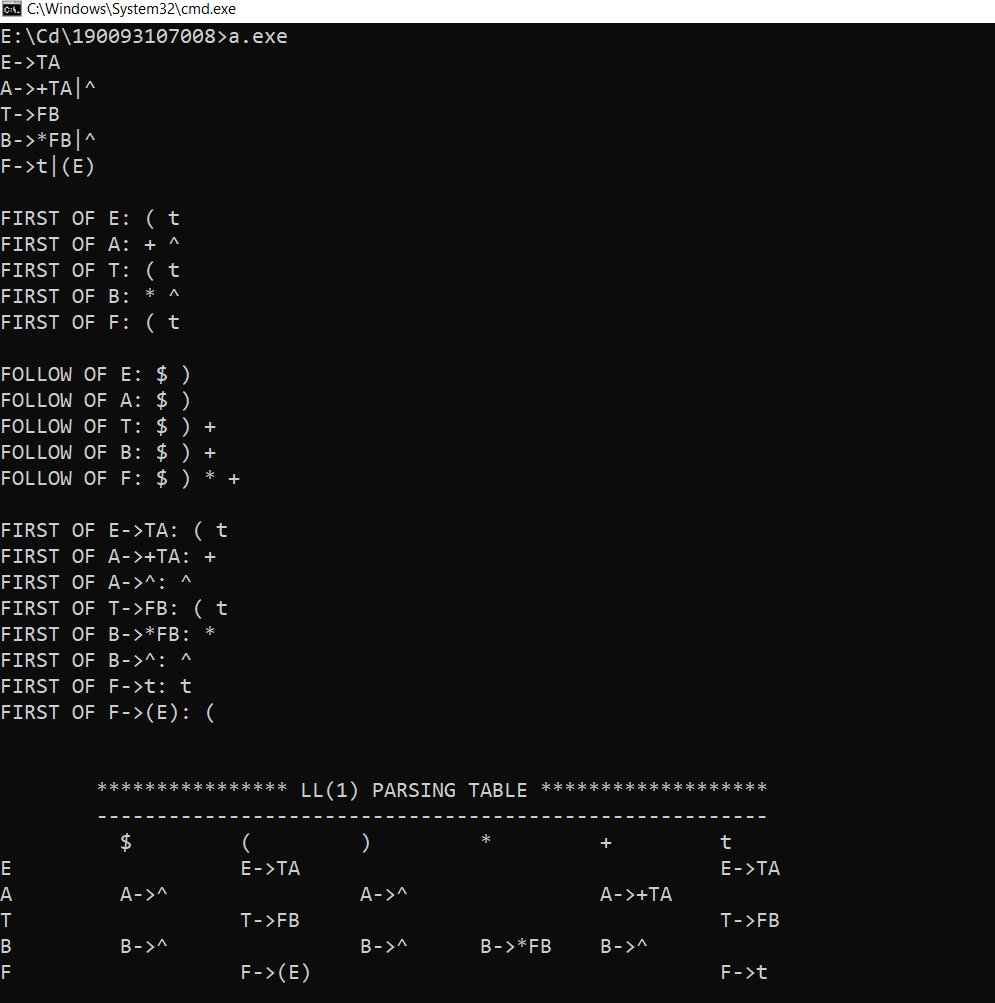
}

}

Inputfile:-LL1.txt E->TA

A->+TA|^ T->FB

B->\*FB|^ F->t|(E)

**Output:**

**Practical - 7**

**Aim**: Generate 3-tuple intermediate code for given infix expression.

**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) || (strcmp(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]);

}

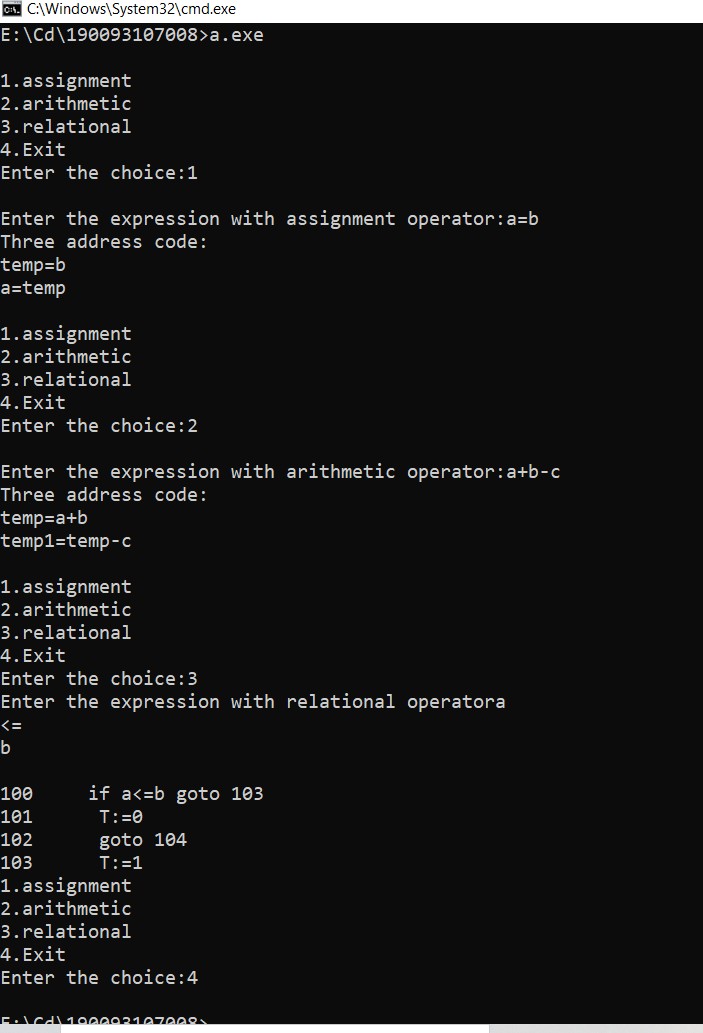
void plus()

{

strncat(exp1, exp, i + 2);

printf("Three address code:\ntemp=%s\ntemp1=temp%c%c\n", exp1, exp[i + 2], exp[i + 3]);

}

**Output**

**Practical - 8**

**Aim**: Extract Predecessor and Successor from given Control Flow Graph

**Code:**

class Node:

def \_init (self, key):

self.key = key self.left = None self.right = None

of key in BST # It sets pre and suc as predecessor and successor respectively

def findPreSuc(root, key): # Base Case

if root is None: return

If key is present at root if root.key == key:

if root.left is not None:

tmp = root.left while(tmp.right):

tmp = tmp.right findPreSuc.pre = tmp

if root.right is not None:

tmp = root.right while(temp.left): tmp = tmp.left

findPreSuc.suc = tmp

return

if root.key > key:

findPreSuc.suc = root findPreSuc(root.left, key)

else

findPreSuc.pre = root findPreSuc(root.right, key)

def insert(node, key): if node is None:

return Node(key)

if key < node.key:

node.left = insert(node.left, key)

else:

node.right = insert(node.right, key)

return node

root = None

root = insert(root, 50) insert(root, 30)

insert(root, 20)

insert(root, 40)

insert(root, 70)

insert(root, 60)

insert(root, 80)

findPreSuc.pre = None

findPreSuc.suc = None findPreSuc(root, key)

if findPreSuc.pre is not None: print("Predecessor is", findPreSuc.pre.key)

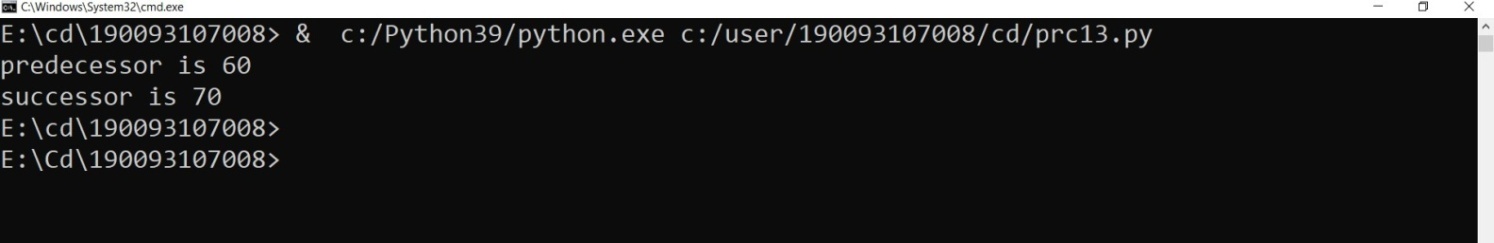
else:

print("No Predecessor")

if findPreSuc.suc is not None: print("Successor is", findPreSuc.suc.key)

else:

print("No Successor")

**Output:**

**Practical - 9**

**Aim:** Introduction to YACC and generate Calculator Program

● YACC stands for Yet Another Compiler Compiler.

● YACC provides a tool to produce a parser for a given grammar.

● YACC is a program designed to compile a LALR (1) grammar.

● It is used to produce the source code of the syntactic analyzer of the language produced by LALR (1) grammar.

● The input of YACC is the rule or grammar and the output is a C program.

Input File :-

/\* definitions \*/

....

%%

/\* rules \*/

....

%%

/\* auxiliary routines \*/

....

Definition Part:

The definition part includes information about the tokens used in the syntax definition. Yacc also recognizes single characters as tokens. The definition part can include C code external to the definition of the parser and variable declarations, within %{ and %} in the first column.

Rule Part:

The rules part contains grammar definition in a modified BNF form. Actions is C code in { } and can be embedded inside (Translation schemes).

Auxiliary Routines Part:

The auxiliary routines part is only C code. It includes function definitions for every function needed in rules part. It can also contain the main() function definition if the parser is going to be run as a program. The main() function must call the function yyparse().

YACC input file generally finishes with: .y

Output Files : -

The output of YACC is a file named y.tab.c

If it contains the main() definition, it must be compiled to be executable. Otherwise, the code can be an external function definition for the function int yyparse() .

If called with the –d option in the command line, Yacc produces as output a header file y.tab.h with all its specific definitions.

If called with the –v option, Yacc produces as output a file y.output containing a textual description of the LALR(1) parsing table used by the parser. This is useful for tracking down how the parser solves conflicts.

For Compiling YACC Program : -

1. Write lex program in a file file.l and yacc in a file file.y

2. Open Terminal and Navigate to the Directory where you have saved the files.

3. type lex file.l

4. type yacc file.y

5. type cc lex.yy.c y.tab.h -ll

6. type ./a.out

**Practical - 10**

**Aim:** Finding “Follow” set

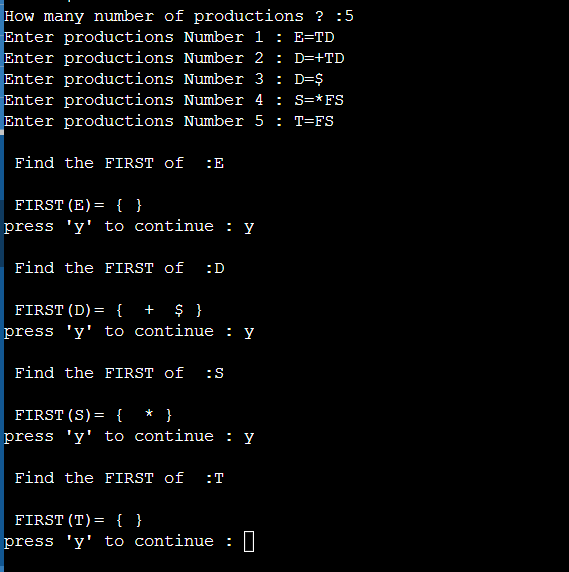
Input: The string consists of grammar symbols.

Output: The Follow set for a given string.

**Code:**

#include<stdio.h>  
#include<ctype.h>  
void FIRST(char[],char );  
void addToResultSet(char[],char);  
int numOfProductions;  
char productionSet[10][10];  
main()  
{  
    int i;  
    char choice;  
    char c;  
    char result[20];  
    printf("How many number of productions ? :");  
    scanf(" %d",&numOfProductions);  
    for(i=0;i<numOfProductions;i++)//read production string eg: E=E+T  
    {  
        printf("Enter productions Number %d : ",i+1);  
        scanf(" %s",productionSet[i]);  
    }  
    do  
    {  
        printf("\n Find the FIRST of  :");  
        scanf(" %c",&c);  
        FIRST(result,c); //Compute FIRST; Get Answer in 'result' array  
        printf("\n FIRST(%c)= { ",c);  
        for(i=0;result[i]!='\0';i++)  
        printf(" %c ",result[i]);       //Display result  
        printf("}\n");  
         printf("press 'y' to continue : ");  
        scanf(" %c",&choice);  
    }  
    while(choice=='y'||choice =='Y');  
}  
/\*  
 \*Function FIRST:  
 \*Compute the elements in FIRST(c) and write them  
 \*in Result Array.  
 \*/  
void FIRST(char\* Result,char c)  
{  
    int i,j,k;  
    char subResult[20];  
    int foundEpsilon;  
    subResult[0]='\0';  
    Result[0]='\0';  
    //If X is terminal, FIRST(X) = {X}.  
    if(!(isupper(c)))  
    {  
        addToResultSet(Result,c);  
               return ;  
    }  
    //If X is non terminal  
    //Read each production  
    for(i=0;i<numOfProductions;i++)  
    {  
//Find production with X as LHS  
        if(productionSet[i][0]==c)  
        {  
//If X → ε is a production, then add ε to FIRST(X).  
 if(productionSet[i][2]=='$') addToResultSet(Result,'$');  
            //If X is a non-terminal, and X → Y1 Y2 … Yk  
            //is a production, then add a to FIRST(X)  
            //if for some i, a is in FIRST(Yi),  
            //and ε is in all of FIRST(Y1), …, FIRST(Yi-1).  
      else  
            {  
                j=2;  
                while(productionSet[i][j]!='\0')  
                {  
                foundEpsilon=0;  
                FIRST(subResult,productionSet[i][j]);  
                for(k=0;subResult[k]!='\0';k++)  
                    addToResultSet(Result,subResult[k]);  
                 for(k=0;subResult[k]!='\0';k++)  
                     if(subResult[k]=='$')  
                     {  
                         foundEpsilon=1;  
                         break;  
                     }  
                 //No ε found, no need to check next element  
                 if(!foundEpsilon)  
                     break;  
                 j++;  
                }  
            }  
    }  
}  
    return ;  
}  
/\* addToResultSet adds the computed  
 \*element to result set.  
 \*This code avoids multiple inclusion of elements  
  \*/  
void addToResultSet(char Result[],char val)  
{  
    int k;  
    for(k=0 ;Result[k]!='\0';k++)  
        if(Result[k]==val)  
            return;  
    Result[k]=val;  
    Result[k+1]='\0';  
}

**OUTPUT: -**



**Practical - 11**

**Aim:** Implement a C program for constructing LL (1) parsing.

**Code:**

#include<stdio.h>

#include<string.h>

#define TSIZE 128

struct product {

char str[100]; int len;

}pro[20];

// no of productions in form A->ß int no\_pro;

char first[26][TSIZE]; char follow[26][TSIZE];

// stores first of each production in form A->ß char first\_rhs[100][TSIZE];

// check if the symbol is nonterminal int isNT(char c) {

return c >= 'A' && c <= 'Z';

}

// reading data from the file void readFromFile() {

FILE\* fptr;

fptr = fopen("LL1.txt", "r");

char buffer[255]; int i;

int j;

while (fgets(buffer, sizeof(buffer), fptr)) { printf("%s", buffer);

j = 0;

nonterminal[buffer[0] - 'A'] = 1;

for (i = 0; i < strlen(buffer) - 1; ++i) { if (buffer[i] == '|') {

++no\_pro;

pro[no\_pro - 1].str[j] = '\0'; pro[no\_pro - 1].len = j;

pro[no\_pro].str[0] = pro[no\_pro - 1].str[0]; pro[no\_pro].str[1] = pro[no\_pro - 1].str[1]; pro[no\_pro].str[2] = pro[no\_pro - 1].str[2]; j = 3;

}

else {

pro[no\_pro].str[j] = buffer[i];

++j;

if (!isNT(buffer[i]) && buffer[i] != '-' && buffer[i] != '>') { terminal[buffer[i]] = 1;

}

}

}

pro[no\_pro].len = j;

++no\_pro;

}

}

void add\_FIRST\_A\_to\_FOLLOW\_B(char A, char B) { int i;

for (i = 0; i < TSIZE; ++i) {

if (i != '^')

follow[B - 'A'][i] = follow[B - 'A'][i] || first[A - 'A'][i];

}

}

void add\_FOLLOW\_A\_to\_FOLLOW\_B(char A, char B) { int i;

for (i = 0; i < TSIZE; ++i) {

if (i != '^')

follow[B - 'A'][i] = follow[B - 'A'][i] || follow[A - 'A'][i];

}

}

void FOLLOW() { int t = 0;

int i, j, k, x;

while (t++ < no\_pro)

{ for (k = 0; k < 26; ++k) {

if (!nonterminal[k]) continue; char nt = k + 'A';

for (i = 0; i < no\_pro; ++i) { for (j = 3; j < pro[i].len; ++j) {

if (nt == pro[i].str[j]) {

for (x = j + 1; x < pro[i].len; ++x) { char sc = pro[i].str[x];

if (isNT(sc)) { add\_FIRST\_A\_to\_FOLLOW\_B(sc, nt); if (first[sc - 'A']['^'])

continue;

}

else {

follow[nt - 'A'][sc] = 1;

}

break;

}

if (x == pro[i].len) add\_FOLLOW\_A\_to\_FOLLOW\_B(pro[i].str[0], nt);

}

}

}

}

}

}

void add\_FIRST\_A\_to\_FIRST\_B(char A, char B) { int i;

for (i = 0; i < TSIZE; ++i) {

if (i != '^') {

first[B - 'A'][i] = first[A - 'A'][i] || first[B - 'A'][i];

}

}

}

void FIRST() { int i, j;

int t = 0;

while (t < no\_pro) {

for (i = 0; i < no\_pro; ++i) { for (j = 3; j < pro[i].len; ++j) { char sc = pro[i].str[j];

if (isNT(sc)) {

add\_FIRST\_A\_to\_FIRST\_B(sc, pro[i].str[0]); if (first[sc - 'A']['^'])

continue;

}

else {

first[pro[i].str[0] - 'A'][sc] = 1;

}

break;

}

if (j == pro[i].len) first[pro[i].str[0] - 'A']['^'] = 1;

}

++t;

}

}

void add\_FIRST\_A\_to\_FIRST\_RHS B(char A, int B) { int i;

for (i = 0; i < TSIZE; ++i) {

if (i != '^')

first\_rhs[B][i] = first[A - 'A'][i] || first\_rhs[B][i];

}

}

// Calculates FIRST(ß) for each A->ß void FIRST\_RHS() {

int i, j; int t = 0;

while (t < no\_pro) {

for (i = 0; i < no\_pro; ++i) { for (j = 3; j < pro[i].len; ++j) { char sc = pro[i].str[j];

if (isNT(sc)) { add\_FIRST\_A\_to\_FIRST\_RHS B(sc, i); if (first[sc - 'A']['^'])

continue;

}

else { first\_rhs[i][sc] = 1;

}

break;

}

if (j == pro[i].len) first\_rhs[i]['^'] = 1;

}

++t;

}

}

int main() { readFromFile();

follow[pro[0].str[0] - 'A']['$'] = 1; FIRST();

FOLLOW();

FIRST\_RHS();

int i, j, k;

// display first of each variable printf("\n");

for (i = 0; i < no\_pro; ++i) {

if (i == 0 || (pro[i - 1].str[0] != pro[i].str[0])) { char c = pro[i].str[0];

printf("FIRST OF %c: ", c); for (j = 0; j < TSIZE; ++j) {

if (first[c - 'A'][j]) { printf("%c ", j);

}

}

printf("\n");

}

}

// display follow of each variable printf("\n");

for (i = 0; i < no\_pro; ++i) {

if (i == 0 || (pro[i - 1].str[0] != pro[i].str[0])) { char c = pro[i].str[0];

printf("FOLLOW OF %c: ", c); for (j = 0; j < TSIZE; ++j) {

if (follow[c - 'A'][j]) { printf("%c ", j);

}

}

printf("\n");

}

}

// display first of each variable ß

// in form A->ß printf("\n");

for (i = 0; i < no\_pro; ++i) { printf("FIRST OF %s: ", pro[i].str); for (j = 0; j < TSIZE; ++j) {

if (first\_rhs[i][j]) {

printf("%c ", j);

}

}

printf("\n");

}

terminal['$'] = 1;

// the parse table do not read '^'

// as input

// so we set terminal['^'] = 0

// to remove '^' from terminals terminal['^'] = 0;

// printing parse table printf("\n");

printf("\n\t\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* LL(1) PARSING TABLE \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\n"); printf("\t- \n");

printf("%-10s", "");

for (i = 0; i < TSIZE; ++i) {

if (terminal[i]) printf("%-10c", i);

}

printf("\n"); int p = 0;

for (i = 0; i < no\_pro; ++i) {

if (i != 0 && (pro[i].str[0] != pro[i - 1].str[0])) p = p + 1;

for (j = 0; j < TSIZE; ++j) {

if (first\_rhs[i][j] && j != '^') { table[p][j] = i + 1;

}

else if (first\_rhs[i]['^']) { for (k = 0; k < TSIZE; ++k) {

if (follow[pro[i].str[0] - 'A'][k]) { table[p][k] = i + 1;

}

}

}

}

}

k = 0;

for (i = 0; i < no\_pro; ++i) {

if (i == 0 || (pro[i - 1].str[0] != pro[i].str[0])) {

printf("%-10c", pro[i].str[0]); for (j = 0; j < TSIZE; ++j) {

if (table[k][j]) {

printf("%-10s", pro[table[k][j] - 1].str);

}

else if (terminal[j]) { printf("%-10s", "");

}

}

++k;

printf("\n");

}

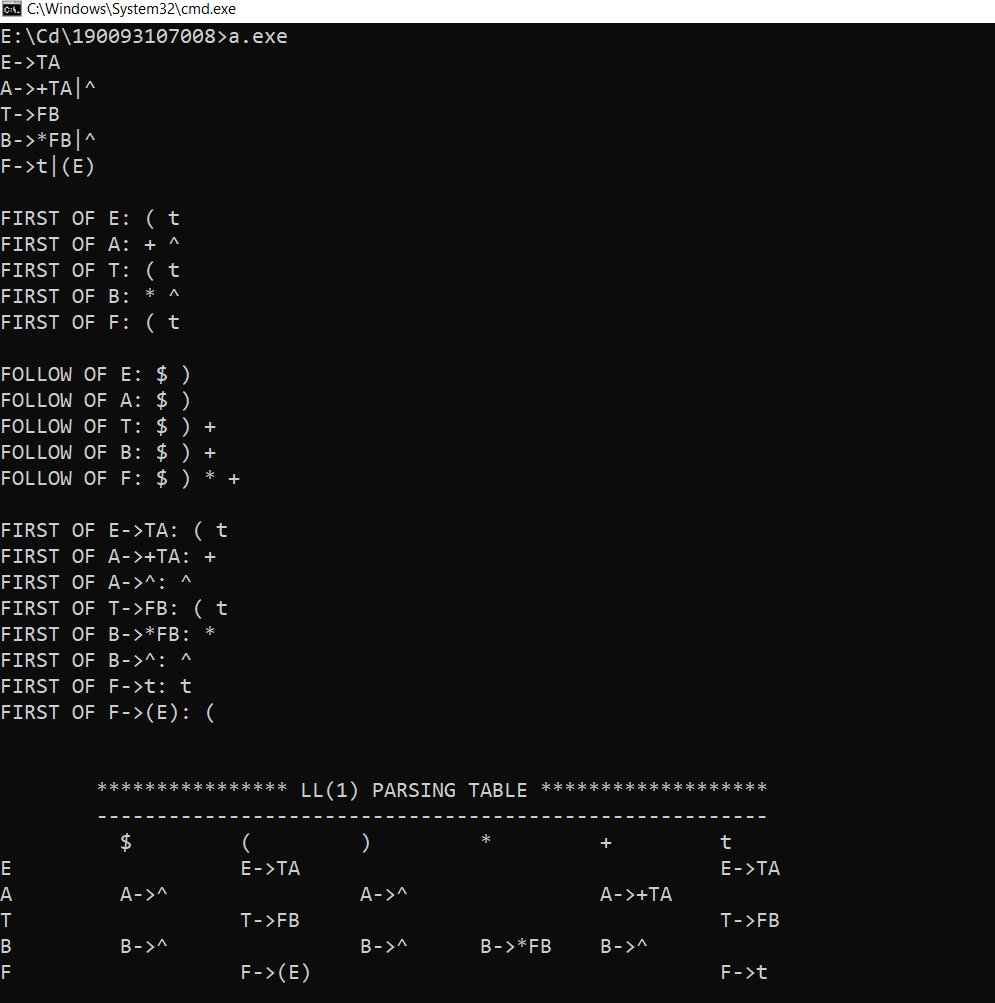
}

}

Inputfile:-LL1.txt E->TA

A->+TA|^ T->FB

B->\*FB|^ F->t|(E)

**Output:**

**Practical - 12**

**Aim:** Implement a C program for constructing LALR parsing.

**Code:**

#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, 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':

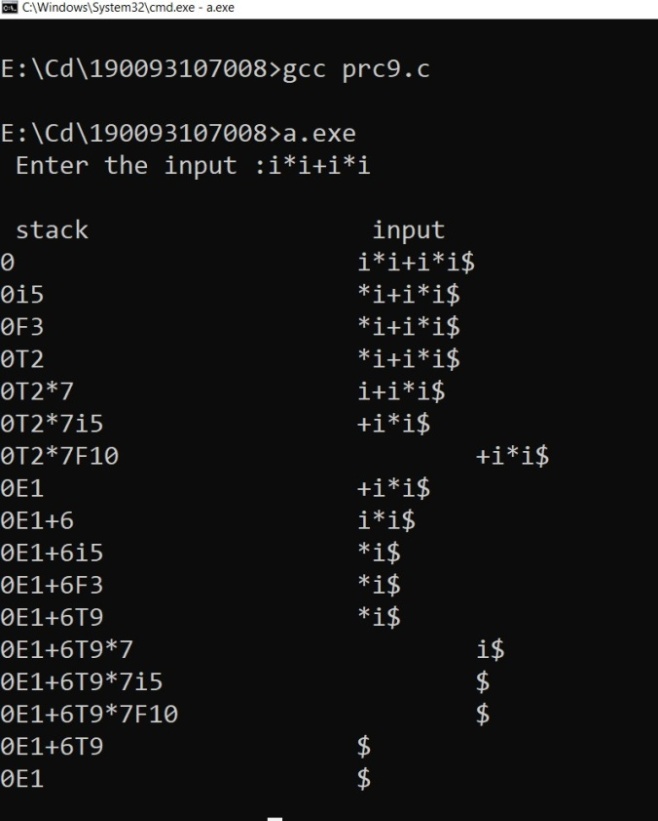
printf("11"); break;

default: printf("%c", t[r]);

break;

}

}

**Output:**

**Practical - 13**

**Aim:** Implement a C program to implement operator precedence parsing.

**Code:**

#include<stdio.h>

#include<conio.h>

void main()

{

char stack[20],ip[20],opt[10][10][1],ter[10]; int i,j,k,n,top=0,col,row;

clrscr();

for(i=0;i<10;i++){stack[i]=NULL;

ip[i]=NULL; for(j=0;j<10;j++){opt[i][j][1]=NULL;}}

printf("Enter the no.of terminals:");

scanf("%d",&n);

printf("\nEnter the terminals:");

scanf("%s",ter);

printf("\nEnter the table values:\n");

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

{

for(j=0;j<n;j++)

{

printf("Enter the value for %c %c:",ter[i],ter[j]);

scanf("%s",opt[i][j]);

}

}

printf("\nOPERATOR PRECEDENCE TABLE:\n");

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

{

printf("\t%c",ter[i]);

}

printf("\n");

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

{

printf("\n%c",ter[i]);

for(j=0;j<n;j++)

{

printf("\t%c",opt[i][j][0]);

}

}

stack[top]='$';

printf("\nEnter the input string:");

scanf("%s",ip);

i=0;

printf("\nSTACK\t\t\tINPUT STRING\t\t\tACTION\n");

printf("\n%s\t\t\t%s\t\t\t",stack,ip);

while(i<=strlen(ip))

{

for(k=0;k<n;k++)

{

if(stack[top]==ter[k]) col=k;

if(ip[i]==ter[k])

row=k;

}

if((stack[top]=='$')&&(ip[i]=='$')){ printf("String is accepted"); break;}

else if((opt[col][row][0]=='<') ||(opt[col][row][0]=='='))

{ stack[++top]=opt[col][row][0]; stack[++top]=ip[i];

printf("Shift %c",ip[i]); i++;

}

else{ if(opt[col][row][0]=='>')

{

while(stack[top]!='<'){--top;} top=top-1; printf("Reduce");

}

else

{

printf("\nString is not accepted"); break;

}

}

printf("\n"); for(k=0;k<=top;k++)

{

printf("%c",stack[k]);

}

printf("\t\t\t"); for(k=i;k<strlen(ip);k++){ printf("%c",ip[k]);

}

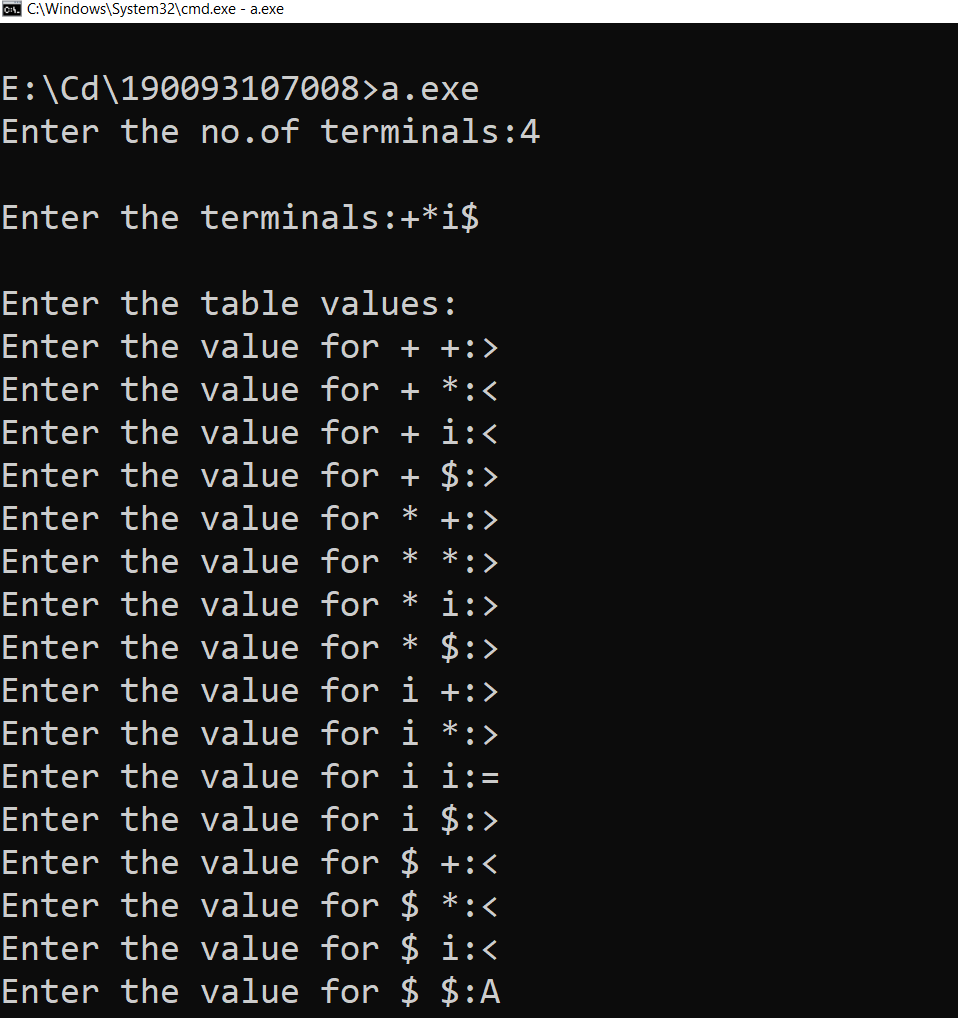
printf("\t\t\t");

}

getch();

}

**Output:**

****