PRACTICAL:-1

AIM:- Implementation of Finite Automata & String Validation.

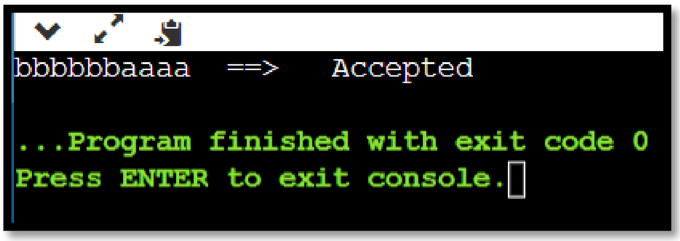
Code:

```
#include <stdio.h>
#include <string.h>
int dfa = 0;
void start(char c) {
  if (c == 'b') {
    dfa = 1;
  } else if (c == 'a') {
    dfa = 3;
  } else {
    dfa = -1;
  }
}
void state1(char c) {
  if (c == 'b') {
    dfa = 2;
  } else if (c == 'a') {
    dfa = 4;
  } else {
    dfa = -1;
  }
}
void state2(char c) {
  if (c == 'a') {
    dfa = 3;
  } else if (c == 'b') {
    dfa = 1;
  } else {
    dfa = -1;
  }
```

```
}
void state3(char c) {
  if (c == 'a') {
    dfa = 3;
  } else if (c == 'b') {
    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() {
  char str[] = "bbbbbbaaaa";
  if (isAccepted(str)) {
    printf("%s", str);
    printf("  ==>   Accepted");
  } else {
    printf("Not Accepted ");
  }
  return 0;
```

Output:



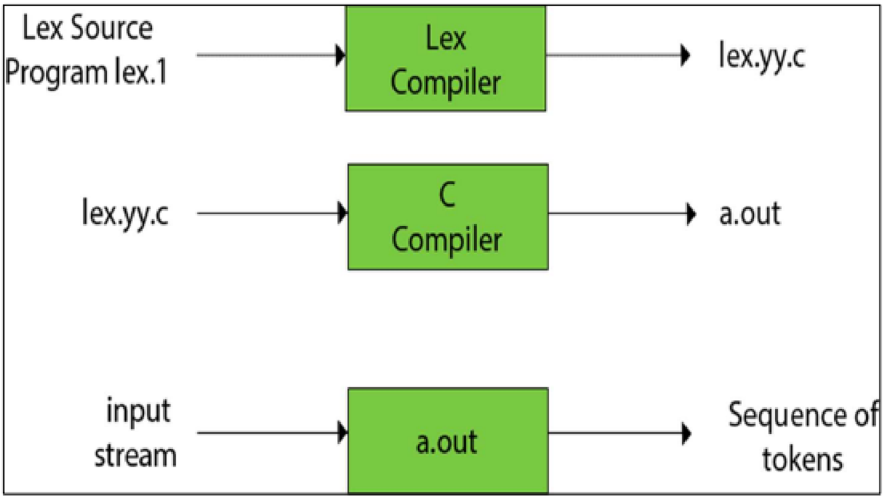
```
bbbbbbaaaa  ==>   Accepted

...Program finished with exit code 0
Press ENTER to exit console.
```

PRACTICAL:-2

AIM:- Introduction to Lex Tool.

What is Lex?

- 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.



Function of Lex:

- Firstly, lexical analyzer creates a program lex.l in the Lex language. Then Lex compiler runs the lex.l 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.

Lex File Format:

A Lex program is separated into three sections by %% delimiters. The formal of Lex source is as follows:

{definitions}

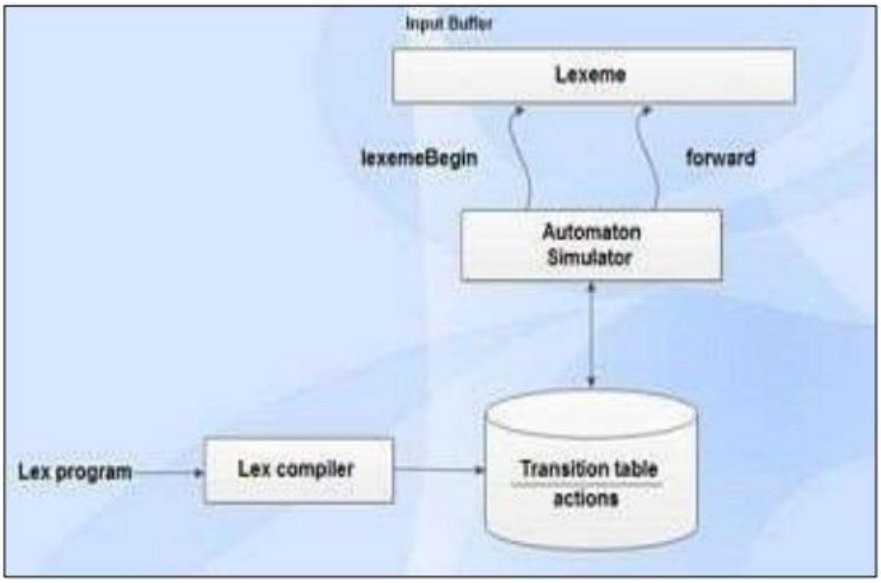
%% {rules} %%

{user subroutines}

Definitions include declarations of constant, variable and regular definitions.

Rules define the statement of form p1 {action1} p2 {action2} … pn {anctionn}. Where pi describes the regular expression and **action1** describes the actions what action the lexical analyzer should take when pattern pi matches a lexeme.

User subroutines are auxiliary procedures needed by the actions. The subroutine can be loaded with lexical analyzer and compiled separately.


Input Buffer
Lexeme
lexemeBegin
forward
Automaton Simulator
Lex program
Lex compiler
Transition table actions

PRACTICAL:-3

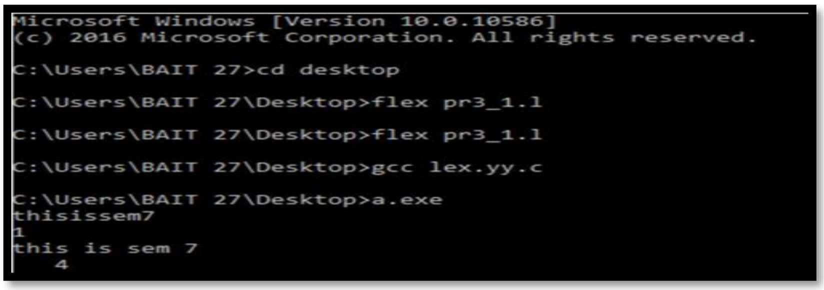
AIM:- : Implementation following programs using Lex.

A. **Generate Histogram of words**
B. **Caesar Cipher**
C. **Extract single and multiline comments from C program**
D. **Convert Roman to Decimal**

Code(A):

```
% {
  #include<stdio.h>
  #include<string.h>
  int i = 0; %
}
%%
([a - zA - Z0 - 9]) * {
  i++;
}
"\n" {
  printf("%d\n", i);
  i = 0;
}
%%
int yywrap(void) {}
int main() {
  yylex();
  return 0;
}
```

Output:

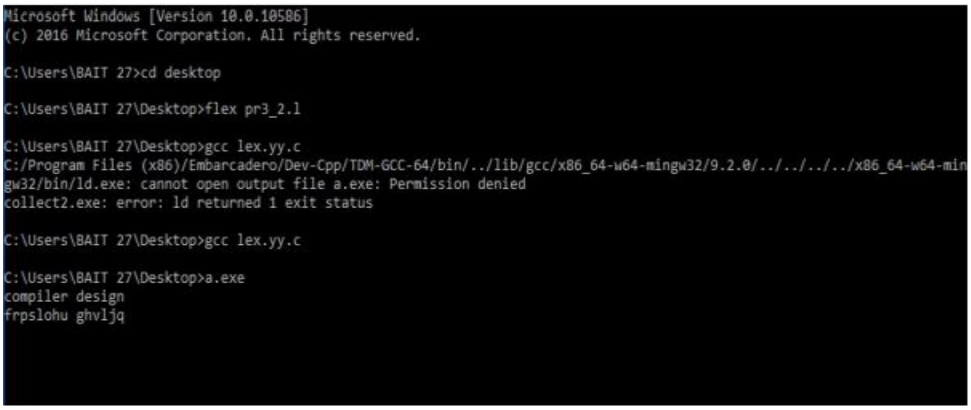


```
Microsoft Windows [Version 10.0.10586]
(c) 2016 Microsoft Corporation. All rights reserved.

C:\Users\BAIT 27>cd desktop

C:\Users\BAIT 27\Desktop>flex pr3_1.l

C:\Users\BAIT 27\Desktop>flex pr3_1.l

C:\Users\BAIT 27\Desktop>gcc lex.yy.c

C:\Users\BAIT 27\Desktop>a.exe
thisissem7
1
this is sem 7
   4
```

Code(B):

```
% {
  #include<stdio.h>
  #include<string.h>
  int cipher_char = 0; %
} %%
[a - z] {
  char cipher_char = yytext[0];
  cipher_char += 3;
  if (cipher_char > 'z') cipher_char -= ('z' + 1 - 'a');
  printf("%c", cipher_char);
}
[A - Z] {
  char cipher_char = yytext[0];
  cipher_char += 3;
  if (cipher_char > 'Z') cipher_char -= ('Z' + 1 - 'A');
  printf("%c", cipher_char);
}
%%
int yywrap(void) {}
int main() {
  yylex();
  return 0;
}
```

Output:

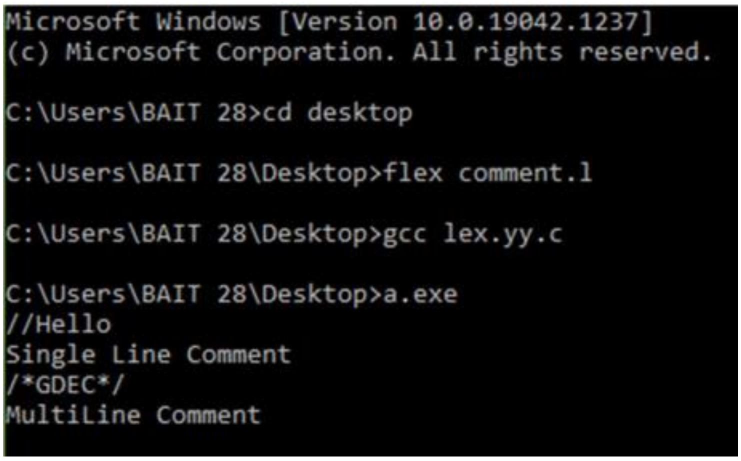


```
Microsoft Windows [Version 10.0.10586]
(c) 2016 Microsoft Corporation. All rights reserved.

C:\Users\BAIT 27>cd desktop

C:\Users\BAIT 27\Desktop>flex pr3_2.l

C:\Users\BAIT 27\Desktop>gcc lex.yy.c
C:/Program Files (x86)/Embarcadero/Dev-Cpp/TDM-GCC-64/bin/../lib/gcc/x86_64-w64-mingw32/9.2.0/../../../../x86_64-w64-min
gw32/bin/ld.exe: cannot open output file a.exe: Permission denied
collect2.exe: error: ld returned 1 exit status

C:\Users\BAIT 27\Desktop>gcc lex.yy.c

C:\Users\BAIT 27\Desktop>a.exe
compiler design
frpslohu ghvljq
```

Code(C):

```
% {
  #include<stdio.h>
  %
} %
% [/]{1}[/]{1}[a-zA-Z0-9]* printf("Single Line Comment");  [/]{1}[*]{1}[a-zA-Z0-9]*[*]{1}[/]{1} printf("MultiLine Comment");  %
    %
    yywrap() {}
    int main() {
      yylex();
      return 0;
    }
```

Output:



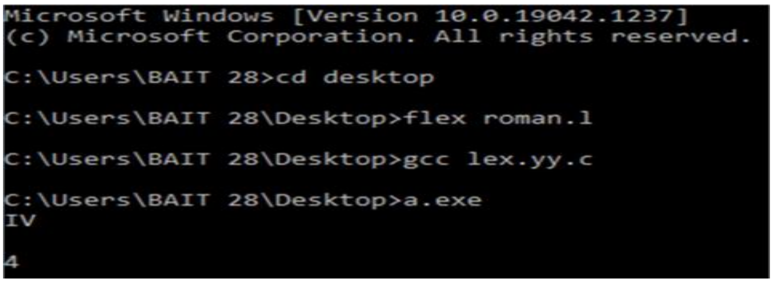
```
Microsoft Windows [Version 10.0.19042.1237]
(c) Microsoft Corporation. All rights reserved.

C:\Users\BAIT 28>cd desktop

C:\Users\BAIT 28\Desktop>flex comment.l

C:\Users\BAIT 28\Desktop>gcc lex.yy.c

C:\Users\BAIT 28\Desktop>a.exe
//Hello
Single Line Comment
/*GDEC*/
MultiLine Comment
```

Code(D):

```
WS[\t] +
%%
  int total = 0;
I total += 1;
IV total += 4;
V total += 5;
IX total += 9;
X total += 10;
XL total += 40;
```

```
L total += 50;
XC total += 90;
C total += 100;
CD total += 400;
D total += 500;
CM total += 900;
M total += 1000; {
  WS
} | \n
return total;
%%
yywrap() {}
int main(void) {
  int first, second;
  first = yylex();
  second = yylex();
  printf("%d\n", first);
  return 0;
}
```

Output:



```
Microsoft Windows [Version 10.0.19042.1237]
(c) Microsoft Corporation. All rights reserved.

C:\Users\BAIT 28>cd desktop

C:\Users\BAIT 28\Desktop>flex roman.l

C:\Users\BAIT 28\Desktop>gcc lex.yy.c

C:\Users\BAIT 28\Desktop>a.exe
IV

4
```

PRACTICAL:-4

AIM:- Implementation of Recursive Descent Parser without backtracking.

Input: The string to be parsed.

Output: Whether string parsed successfully or not.

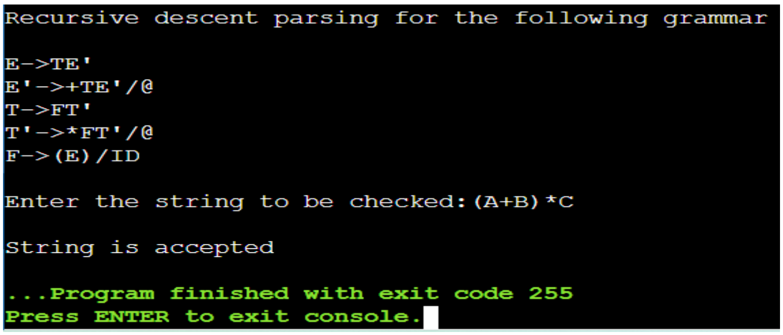
Explanation: Students have to implement the recursive procedure for RDP for a typical grammar. The production no. is displayed as they are used to derive the string.

Code:

```
#include<stdio.h>
#include<conio.h>
#include<string.h>
char input[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("\nEnter the string to be checked:");
  gets(input);
  if (E()) {
    if (input[i + 1] == '\0')
      printf("\nString is accepted");
    else
      printf("\nString is not accepted");
  } else
    printf("\nString not accepted");
  getch();
}
E() {
  if (T()) {
    if (EP())
      return (1);
    else
      return (0);
  } else
    return (0);
}
EP() {
  if (input[i] == '+') {
    i++;
    if (T()) {
      if (EP())
        return (1);
      else
        return (0);
    } else
      return (0);
  } else
    return (1);
}
T() {
  if (F()) {
    if (TP())
      return (1);
    else
      return (0);
  } else
    return (0);
}
TP() {
  if (input[i] == '*') {
    i++;
    if (F()) {
      if (TP())
```

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

Output:



```
Recursive descent parsing for the following grammar

E->TE'
E'->+TE'/@
T->FT'
T'->*FT'/@
F->(E)/ID

Enter the string to be checked:(A+B)*C

String is accepted

...Program finished with exit code 255
Press ENTER to exit console.
```

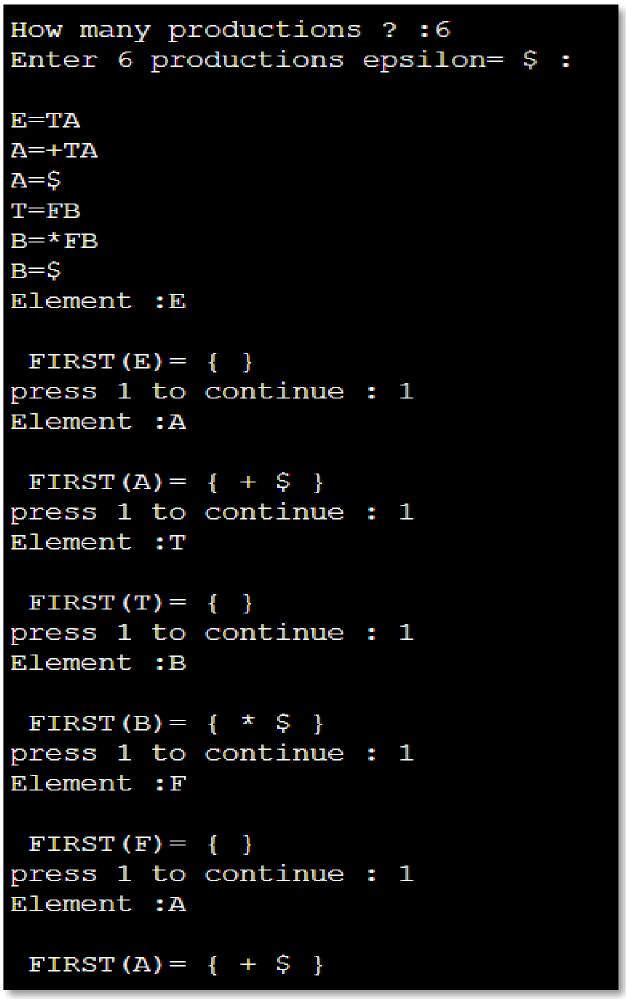
PRACTICAL:-5

AIM:- Finding "First" set Input: The string consists of grammar symbols. Output: The First set for a given string. Explanation: The students have 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<ctype.h>
void FIRST(char);
int count, n = 0;
char prodn[10][10], first[10];
main() {
  int i, choice;
  char c, ch;
  printf("How many productions ? :");
  scanf("%d", & count);
  printf("Enter %d productions epsilon= $ :\n\n", count);
  for (i = 0; i < count; i++)
    scanf("%s%c", prodn[i], & ch);
  do {
    n = 0;
    printf("Element :");
    scanf("%c", & c);
    FIRST(c);
    printf("\n FIRST(%c)= { ", c);
    for (i = 0; i < n; i++)
      printf("%c ", first[i]);
    printf("}\n");
    printf("press 1 to continue : ");
    scanf("%d%c", & choice, & ch);
  }
  while (choice == 1);
}
void FIRST(char c) {
  int j;
  if (!(isupper(c))) first[n++] = c;
  for (j = 0; j < count; j++) {
    if (prodn[j][0] == c) {
      if (prodn[j][2] == '$') first[n++] = '$';
      else if (islower(prodn[j][2])) first[n++] = prodn[j][2];
      else FIRST(prodn[j][2]);
    }
  }
}
```

Output:



```
How many productions ? :6
Enter 6 productions epsilon= $ :

E=TA
A=+TA
A=$
T=FB
B=*FB
B=$
Element :E

 FIRST(E)= {  }
press 1 to continue : 1
Element :A

 FIRST(A)= { + $ }
press 1 to continue : 1
Element :T

 FIRST(T)= {  }
press 1 to continue : 1
Element :B

 FIRST(B)= { * $ }
press 1 to continue : 1
Element :F

 FIRST(F)= {  }
press 1 to continue : 1
Element :A

 FIRST(A)= { + $ }
```

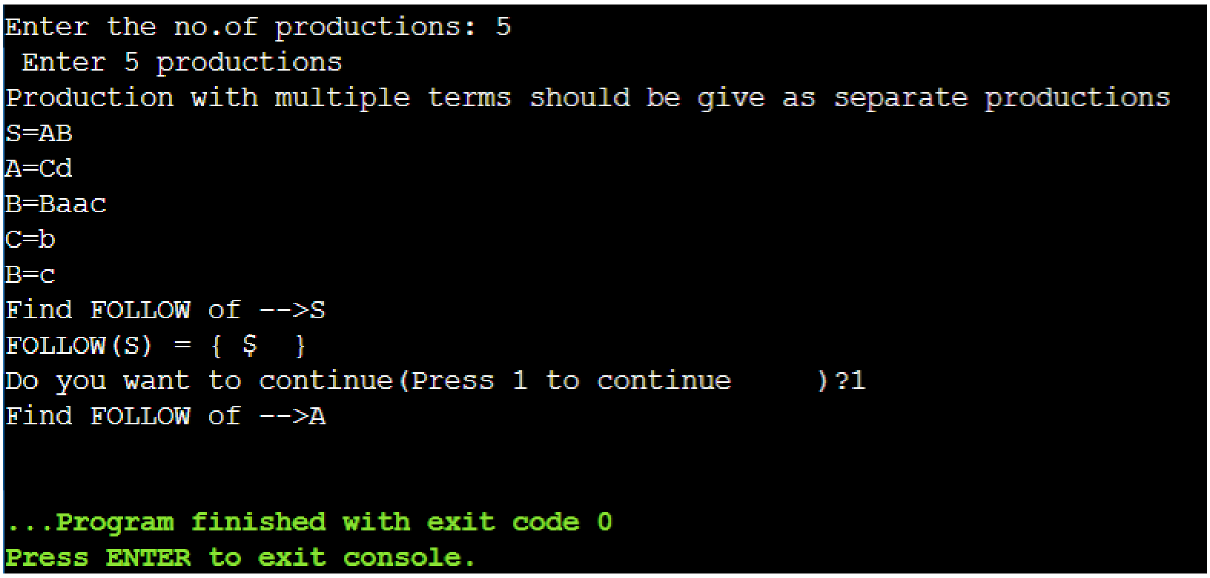
PRACTICAL:-6

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

Code:

```
#include<stdio.h>
#include<string.h>
int n, m = 0, p, i = 0, j = 0;
char a[10][10], followResult[10];
void follow(char c);
void first(char c);
void addToResult(char);
int main() {
  int i;
  int choice;
  char c, ch;
  printf("Enter the no.of productions: ");
  scanf("%d", & n);
  printf(" Enter %d productions\nProduction with multiple terms should be give as separate 
productions \n", n);
  for (i = 0; i < n; i++)
    scanf("%s%c", a[i], & ch);
  // gets(a[i]);
  do {
    m = 0;
    printf("Find FOLLOW of -->");
    scanf(" %c", & c);
    follow(c);
    printf("FOLLOW(%c) = { ", c);
    for (i = 0; i < m; i++)
      printf("%c ", followResult[i]);
    printf(" }\n");
    printf("Do you want to continue(Press 1 to continue         )?");
    scanf("%d%c", & choice, & ch);
  }
  while (choice == 1);
}
void follow(char c) {
  if (a[0][0] == c) addToResult('$');
  for (i = 0; i < n; i++) {
    for (j = 2; j < strlen(a[i]); j++) {
      if (a[i][j] == c) {
        if (a[i][j + 1] != '\0') first(a[i][j + 1]);
        if (a[i][j + 1] == '\0' && c != a[i][0])
          follow(a[i][0]);
      }
    }
  }
}
void first(char c) {
  int k;
  if (!(isupper(c)))
    //f[m++]=c;
    addToResult(c);
  for (k = 0; k < n; k++) {
    if (a[k][0] == c) {
      if (a[k][2] == '$')
        follow(a[i][0]);
      else if (islower(a[k][2]))
        //f[m++]=a[k][2];
        addToResult(a[k][2]);
      else first(a[k][2]);
    }
  }
}
void addToResult(char c) {
  int i;
  for (i = 0; i <= m; i++)
    if (followResult[i] == c)
      return;
  followResult[m++] = c;
}
```

Output:



```
Enter the no.of productions: 5
 Enter 5 productions
Production with multiple terms should be give as separate productions
S=AB
A=Cd
B=Baac
C=b
B=c
Find FOLLOW of -->S
FOLLOW(S) = { $  }
Do you want to continue(Press 1 to continue     )?1
Find FOLLOW of -->A


...Program finished with exit code 0
Press ENTER to exit console.
```

PRACTICAL:-7

AIM:- Introduction to YACC.

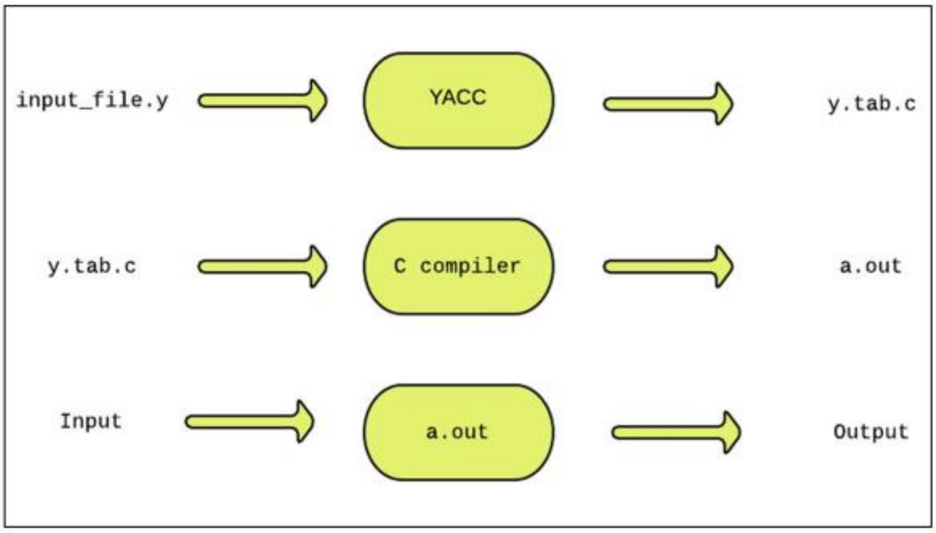
- YACC stands for Yet Another Compiler-Compiler.
- YACC provides a tool to produce a parser for a given grammar.
- YACC is a program designed to compile a LALR (1) grammar.
- It is used to produce the source code of the syntactic analyzer of the language produced by LALR (1) grammar.
- The input of YACC is the rule or grammar and the output is a C program.

Input: A CFG-file.y

Output: A parser y.tab.c (vacc)

- The output file “file.output” contains the parsing tables.
- The file “file.tab.h” contains declarations.
- The parser called the yyparse().
- Parser expects to use a function called yylex () to get tokens.

The operational sequence is as follow:



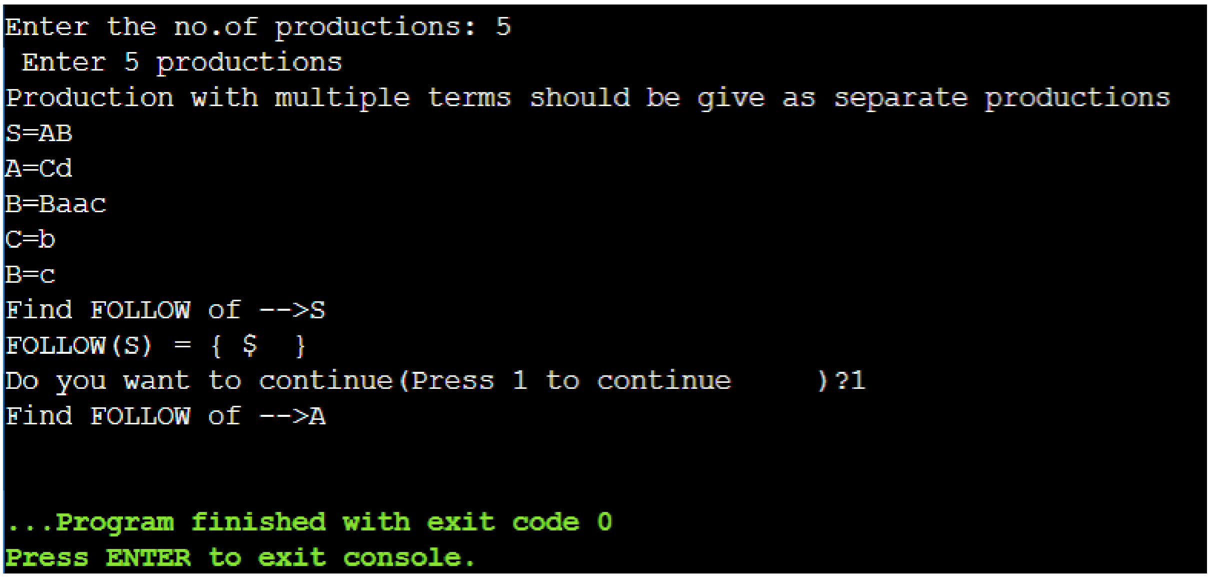
PRACTICAL:-8

AIM:- Finding “Follow” set Input: The string consists of grammar symbols. Output: The Follow set for a given string. Explanation: The students have to assume a typical grammar. The program when run will ask for the string to be entered. The program will find the Follow set of the given string.

Code:

```
#include<stdio.h>
#include<string.h>
int n, m = 0, p, i = 0, j = 0;
char a[10][10], followResult[10];
void follow(char c);
void first(char c);
void addToResult(char);
int main() {
  int i;
  int choice;
  char c, ch;
  printf("Enter the no.of productions: ");
  scanf("%d", & n);
  printf(" Enter %d productions\nProduction with multiple terms should be give as separate
productions \n", n);
  for (i = 0; i < n; i++)
    scanf("%s%c", a[i], & ch);
  // gets(a[i]);
  do {
    m = 0;
    printf("Find FOLLOW of -->");
    scanf(" %c", & c);
    follow(c);
    printf("FOLLOW(%c) = { ", c);
    for (i = 0; i < m; i++)
      printf("%c ", followResult[i]);
    printf(" }\n");
    printf("Do you want to continue(Press 1 to continue         )?");
    scanf("%d%c", & choice, & ch);
  }
  while (choice == 1);
}
void follow(char c) {
  if (a[0][0] == c) addToResult('$');
  for (i = 0; i < n; i++) {
    for (j = 2; j < strlen(a[i]); j++) {
      if (a[i][j] == c) {
        if (a[i][j + 1] != '\0') first(a[i][j + 1]);
        if (a[i][j + 1] == '\0' && c != a[i][0])
          follow(a[i][0]);
      }
    }
  }
}
void first(char c) {
  int k;
  if (!(isupper(c)))
    //f[m++]=c;
    addToResult(c);
  for (k = 0; k < n; k++) {
    if (a[k][0] == c) {
      if (a[k][2] == '$')
        follow(a[i][0]);
      else if (islower(a[k][2]))
        //f[m++]=a[k][2];
        addToResult(a[k][2]);
      else first(a[k][2]);
    }
  }
}
void addToResult(char c) {
  int i;
  for (i = 0; i <= m; i++)
    if (followResult[i] == c)
      return;
  followResult[m++] = c;
}
```

Output:



```
Enter the no.of productions: 5
 Enter 5 productions
Production with multiple terms should be give as separate productions
S=AB
A=Cd
B=Baac
C=b
B=c
Find FOLLOW of -->S
FOLLOW(S) = { $  }
Do you want to continue(Press 1 to continue     )?1
Find FOLLOW of -->A


...Program finished with exit code 0
Press ENTER to exit console.
```

PRACTICAL:-9

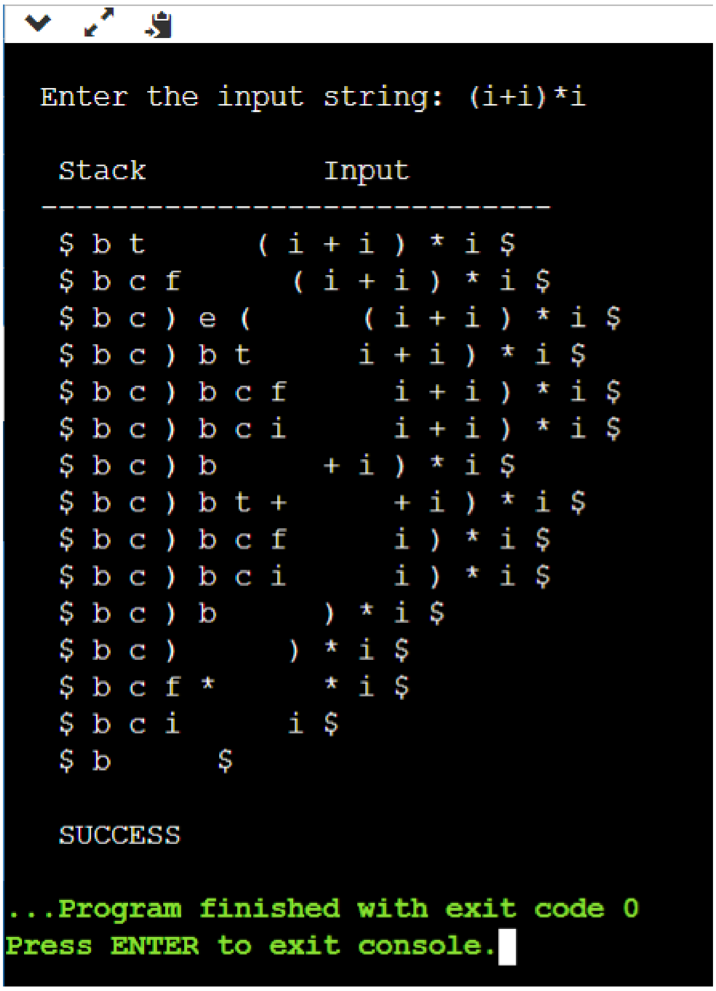
AIM:- Implement a C program for constructing LL (1) parsing.

Code:

```
#include<stdio.h>
#include<conio.h>
#include<string.h>
#include<stdlib.h>
char s[20], stack[20];
void main() {
  char m[5][6][3]={"tb"," "," ","tb"," "," "," ","+tb"," "," ","n","n","fc"," "," ","fc"," "," ","
  ","n","*fc"," a ","n","n","i"," "," ","(e)"," "," "};
int size[5][6]={2,0,0,2,0,0,0,3,0,0,1,1,2,0,0,2,0,0,0,1,3,0,1,1,1,0,0,3,0,0};
  int i, j, k, n, str1, str2;
  printf("\n  Enter the input string: ");
  scanf("%s", s);
  strcat(s, "$");
  n = strlen(s);
  stack[0] = '$';
  stack[1] = 'e';
  i = 1;
  j = 0;
  printf("\n   Stack         Input\n  ");
  printf("-----------------------------\n  ");
  while ((stack[i] != '$') && (s[j] != '$')) {
    if (stack[i] == s[j]) {
      i--;
      j++;
    }
    switch (stack[i]) {
    case 'e': str1 = 0;
      break;
    case 'b': str1 = 1;
      break;
    case 't': str1 = 2;
      break;
    case 'c': str1 = 3;
      break;
    case 'f': str1 = 4;
      break;
    }
    switch (s[j]) {
    case 'i': str2 = 0;
      break;
    case '+': str2 = 1;
      break;
    case '*': str2 = 2;
      break;
    case '(': str2 = 3;
      break;
    case ')': str2 = 4;
      break;
    case '$': str2 = 5;
      break;
    }
    if (m[str1][str2][0] == '\0') {
      printf("\nERROR");
      exit(0);
    } else if (m[str1][str2][0] == 'n')
      i--;
    else if (m[str1][str2][0] == 'i')
      stack[i] = 'i';
    else {
      for (k = size[str1][str2] - 1; k >= 0; k--) {
        stack[i] = m[str1][str2][k];
```

```
        i++;
      }
      i--;
    }
    for (k = 0; k <= i; k++)
      printf(" %c", stack[k]);
    printf("     ");
    for (k = j; k <= n; k++)
      printf(" %c", s[k]);
    printf("\n  ");
  }
  printf("\n   SUCCESS");
  getch();
}
```

Output:



```
Enter the input string: (i+i)*i

 Stack          Input
-----------------------------
 $ b t       ( i + i ) * i $
 $ b c f       ( i + i ) * i $
 $ b c ) e (       ( i + i ) * i $
 $ b c ) b t       i + i ) * i $
 $ b c ) b c f       i + i ) * i $
 $ b c ) b c i       i + i ) * i $
 $ b c ) b       + i ) * i $
 $ b c ) b t +       + i ) * i $
 $ b c ) b c f       i ) * i $
 $ b c ) b c i       i ) * i $
 $ b c ) b       ) * i $
 $ b c )       ) * i $
 $ b c f *       * i $
 $ b c i       i $
 $ b       $

 SUCCESS

...Program finished with exit code 0
Press ENTER to exit console.
```

PRACTICAL:-10

AIM:- Implement a C program to implement 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 :");
```

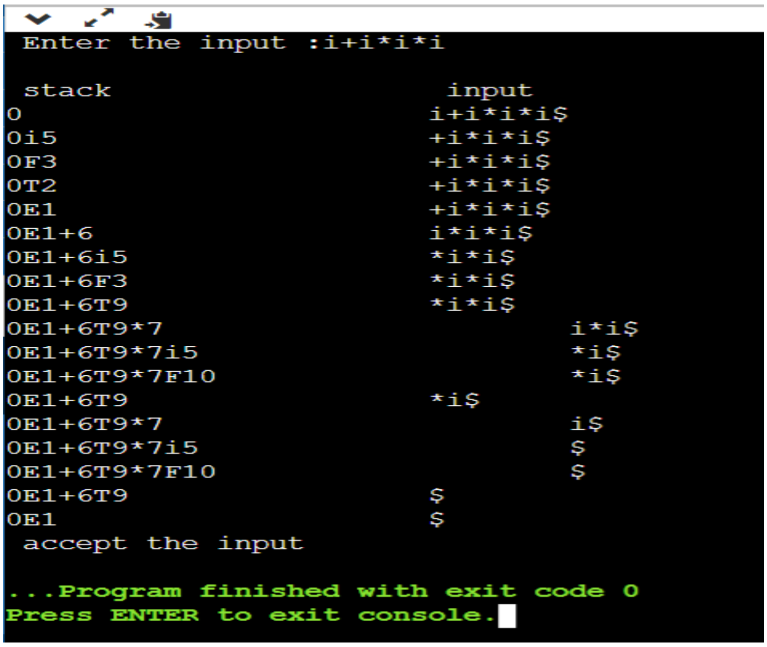
```
scanf("%s", inp);
len = strlen(inp);
inp[len] = '$';
inp[len + 1] = '\0';
push(stack, & top, bl);
printf("\n stack \t\t\t input");
printt(stack, & top, inp, i);
do {
  x = inp[i];
  p = stacktop(stack);
  isproduct(x, p);
  if (strcmp(temp, "emp") == 0)
    error();
  if (strcmp(temp, "acc") == 0)
    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:



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

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

...Program finished with exit code 0
Press ENTER to exit console.
```

PRACTICAL:-11

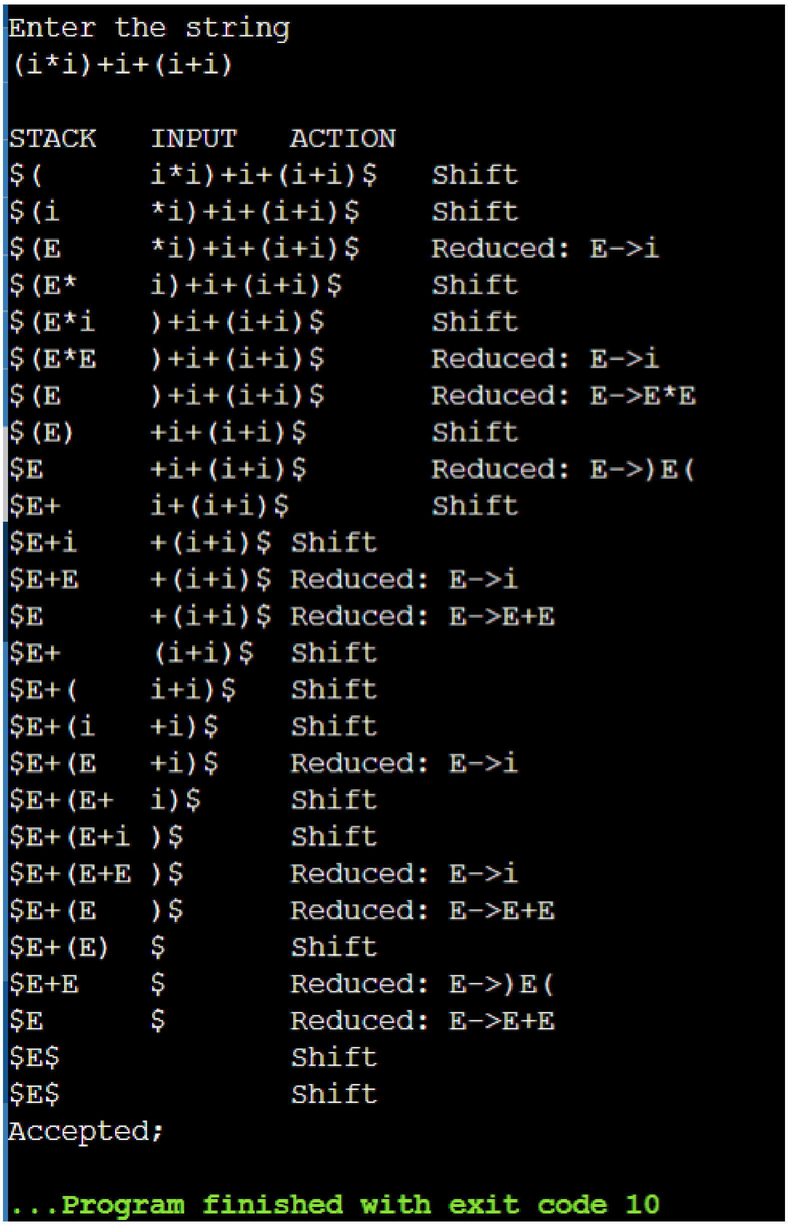
AIM:- : Implement a C program to implement operator precedence parsing.

Code:

```
#include<stdio.h>
#include<string.h>
char * input;
int i = 0;
char lasthandle[6], stack[50], handles[][5] = {")E(","E*E","E+E","i","E^E"};
//(E) becomes )E( when pushed to stack
int top = 0, l;
char prec[9][9] = {
               /*input*/
               /*stack + - * / ^ i ( ) $ */
            /* + */ '>', '>','<','<','<','<','<','>','>',
            /* - */ '>', '>','<','<','<','<','<','>','>',
            /* * */ '>', '>','>','>','<','<','<','>','>',
            /* / */ '>', '>','>','>','<','<','<','>','>',
            /* ^ */ '>', '>','>','>','<','<','<','>','>',
            /* i */ '>', '>','>','>','>','e','e','>','>',
            /* ( */ '<', '<','<','<','<','<','<','>','e',
            /* ) */ '>', '>','>','>','>','e','e','>','>',
            /* $ */ '<', '<','<','<','<','<','<','<','>',
            };
int getindex(char c) {
  switch (c) {
  case '+': return 0;
  case '-': return 1;
  case '*': return 2;
  case '/': return 3;
  case '^': return 4;
  case 'i': return 5;
  case '(': return 6;
  case ')': return 7;
  case '$': return 8;
  }
}
int shift() {
  stack[++top] = * (input + i++);
  stack[top + 1] = '\0';
}
int reduce() {
  int i, len, found, t;
  for (i = 0; i < 5; i++) //selecting handles
  {
    len = strlen(handles[i]);
    if (stack[top] == handles[i][0] && top + 1 >= len) {
      found = 1;
      for (t = 0; t < len; t++) {
        if (stack[top - t] != handles[i][t]) {
          found = 0;
          break;
        }
      }
      if (found == 1) {
        stack[top - t + 1] = 'E';
        top = top - t + 1;
        strcpy(lasthandle, handles[i]);
        stack[top + 1] = '\0';
        return 1; //successful reduction
      }
    }
  }
  return 0;
```

```
}
void dispstack() {
  int j;
  for (j = 0; j <= top; j++)
    printf("%c", stack[j]);
}
void dispinput() {
  int j;
  for (j = i; j < l; j++)
    printf("%c", *(input + j));
}
void main() {
  int j;
  input = (char * ) malloc(50 * sizeof(char));
  printf("\nEnter the string\n");
  scanf("%s", input);
  input = strcat(input, "$");
  l = strlen(input);
  strcpy(stack, "$");
  printf("\nSTACK\tINPUT\tACTION");
  while (i <= l) {
    shift();
    printf("\n");
    dispstack();
    printf("\t");
    dispinput();
    printf("\tShift");
    if (prec[getindex(stack[top])][getindex(input[i])] == '>') {
      while (reduce()) {
        printf("\n");
        dispstack();
        printf("\t");
        dispinput();
        printf("\tReduced: E->%s", lasthandle);
      }
    }
  }
  if (strcmp(stack, "$E$") == 0)
    printf("\nAccepted;");
  else
    printf("\nNot Accepted;");
}
```

Output:



```
Enter the string
(i*i)+i+(i+i)

STACK   INPUT   ACTION
$(      i*i)+i+(i+i)$   Shift
$(i     *i)+i+(i+i)$    Shift
$(E     *i)+i+(i+i)$    Reduced: E->i
$(E*    i)+i+(i+i)$     Shift
$(E*i   )+i+(i+i)$      Shift
$(E*E   )+i+(i+i)$      Reduced: E->i
$(E     )+i+(i+i)$      Reduced: E->E*E
$(E)    +i+(i+i)$       Shift
$E      +i+(i+i)$       Reduced: E->)E(
$E+     i+(i+i)$        Shift
$E+i    +(i+i)$ Shift
$E+E    +(i+i)$ Reduced: E->i
$E      +(i+i)$ Reduced: E->E+E
$E+     (i+i)$  Shift
$E+(    i+i)$   Shift
$E+(i   +i)$    Shift
$E+(E   +i)$    Reduced: E->i
$E+(E+  i)$     Shift
$E+(E+i )$      Shift
$E+(E+E )$      Reduced: E->i
$E+(E   )$      Reduced: E->E+E
$E+(E)  $       Shift
$E+E    $       Reduced: E->)E(
$E      $       Reduced: E->E+E
$E$             Shift
$E$             Shift
Accepted;

...Program finished with exit code 10
```