DESCRIPTION OF PRACTICALS

1. **WAP in C language for converting infix expression into postfix expression.**

Code:

```
#include <stdio.h>
#include <ctype.h>
#define SIZE 50
char stack[SIZE];
int top=-1;
push(char elem)
{
   stack[++top]=elem;
}
char pop()
{
   return(stack[top--]);
}
int pr(char symbol)
{
            if(symbol == '^')
        {
                        return(3);
            }
            else if(symbol == '*' || symbol == '/')
            {
                        return(2);
            }
            else if(symbol == '+' || symbol == '-')
            {
                        return(1);
            }
            else
            {
                        return(0);
            }
}
void main()
{
   char infix[50],postfix[50],ch,elem;
   int i=0,k=0;
   printf("Enter Infix Expression : ");
   scanf("%s",infix);
     push('#');
   while( (ch=infix[i++]) != '\0')
   {
      if( ch == '(') push(ch);
      else
         if(isalnum(ch)) postfix[k++]=ch;
         else
            if( ch == ')')
            {
```

```
            while( stack[top] != '(')
                postfix[k++]=pop();
            elem=pop();
        }
        else
        {
            while( pr(stack[top]) >= pr(ch) )
                postfix[k++]=pop();
            push(ch);
        }
    }
    while( stack[top] != '#')
        postfix[k++]=pop();
    postfix[k]='\0';
    printf("\nPostfix Expression =  %s\n",postfix);
}
```

Output:

```
Enter Infix Expression : A*(B+C)-D

Postfix Expression =  ABC+*D-
```

2. **WAP in C language for a string that accepts alphabets a & b and ended with ab.**

Code:

```
#include<stdio.h>
#define max 100
main() {
  char str[max],f='1';
  int i;
  printf("enter the string to be checked: ");
  scanf("%s",str);
  for(i=0;str[i]!='\0';i++) {
    switch(f) {
      case '1': if(str[i]=='a') f='2';
        else if(str[i]=='b') f='1';
      break;
      case '2': if(str[i]=='a') f='2';
        else if(str[i]=='b') f='3';
      break;
      case '3': if(str[i]=='a') f='2';
        else if(str[i]=='b') f='3';
      break;
    }
  }
  if(f=='3')
    printf("String is accepted", f);
  else printf("String is not accepted", f);
    return 0;
}
```

Output:

```
enter the string to be checked: baaaaaba
String is not accepted
```

3. **WAP in C language to check the given no. is an integer, float or exponential.**

Code:

```
#include <stdio.h>

int main() {
    char number[100];
    int flag = 0;
    printf("Enter the number to check itself: ");
    scanf("%s", number);
    for (int i = 0; number[i] != 0; i++) {
        if (number[i] == '.') {
            flag = 1;
            break;
        }
    }
    if (flag)
    {
      printf("\n%s is a floating-point number.\n", number);
    }
    else{
      printf("\n%s is an integer number.\n", number);
      }
    return 0;
}
```

Output:

```
Enter the number to check itself: 10.2

10.2 is a floating-point number.
```

4. **WAP in C language to find out identifiers, keywords and operators in syntax.**

Code:

```
#include <stdbool.h>
#include <stdio.h>
#include <string.h>
#include <stdlib.h>
bool isDelimiter(char ch)
{
        if (ch == ' ' || ch == '+' || ch == '-' || ch == '*' ||
```

```
		ch == '/' || ch == ',' || ch == ';' || ch == '>' ||
		ch == '<' || ch == '=' || ch == '(' || ch == ')' ||
		ch == '[' || ch == ']' || ch == '{' || ch == '}')
		return (true);
	return (false);
}
bool isOperator(char ch)
{
	if (ch == '+' || ch == '-' || ch == '*' ||
		ch == '/' || ch == '>' || ch == '<' ||
		ch == '=')
		return (true);
	return (false);
}
bool validIdentifier(char* str)
{
	if (str[0] == '0' || str[0] == '1' || str[0] == '2' ||
		str[0] == '3' || str[0] == '4' || str[0] == '5' ||
		str[0] == '6' || str[0] == '7' || str[0] == '8' ||
		str[0] == '9' || isDelimiter(str[0]) == true)
		return (false);
	return (true);
}
bool isKeyword(char* str)
{
	if (!strcmp(str, "if") || !strcmp(str, "else") ||
		!strcmp(str, "while") || !strcmp(str, "do") ||
		!strcmp(str, "break") ||
		!strcmp(str, "continue") || !strcmp(str, "int")
		|| !strcmp(str, "double") || !strcmp(str, "float")
		|| !strcmp(str, "return") || !strcmp(str, "char")
		|| !strcmp(str, "case") || !strcmp(str, "char")
		|| !strcmp(str, "sizeof") || !strcmp(str, "long")
		|| !strcmp(str, "short") || !strcmp(str, "typedef")
		|| !strcmp(str, "switch") || !strcmp(str, "unsigned")
		|| !strcmp(str, "void") || !strcmp(str, "static")
		|| !strcmp(str, "struct") || !strcmp(str, "goto"))
		return (true);
	return (false);
}
bool isInteger(char* str)
{
	int i, len = strlen(str);

	if (len == 0)
		return (false);
	for (i = 0; i < len; i++) {
		if (str[i] != '0' && str[i] != '1' && str[i] != '2'
			&& str[i] != '3' && str[i] != '4' && str[i] != '5'
			&& str[i] != '6' && str[i] != '7' && str[i] != '8'
			&& str[i] != '9' || (str[i] == '-' && i > 0))
			return (false);
	}
	return (true);
}
bool isRealNumber(char* str)
```

```
{
	int i, len = strlen(str);
	bool hasDecimal = false;

	if (len == 0)
		return (false);
	for (i = 0; i < len; i++) {
		if (str[i] != '0' && str[i] != '1' && str[i] != '2'
			&& str[i] != '3' && str[i] != '4' && str[i] != '5'
			&& str[i] != '6' && str[i] != '7' && str[i] != '8'
			&& str[i] != '9' && str[i] != '.' ||
			(str[i] == '-' && i > 0))
			return (false);
		if (str[i] == '.')
			hasDecimal = true;
	}
	return (hasDecimal);
}
char* subString(char* str, int left, int right)
{
	int i;
	char* subStr = (char*)malloc(
					sizeof(char) * (right - left + 2));

	for (i = left; i <= right; i++)
		subStr[i - left] = str[i];
	subStr[right - left + 1] = '\0';
	return (subStr);
}
void parse(char* str)
{
	int left = 0, right = 0;
	int len = strlen(str);

	while (right <= len && left <= right) {
		if (isDelimiter(str[right]) == false)
			right++;

		if (isDelimiter(str[right]) == true && left == right) {
			if (isOperator(str[right]) == true)
				printf("'%c' IS AN OPERATOR\n", str[right]);

			right++;
			left = right;
		} else if (isDelimiter(str[right]) == true && left != right
				|| (right == len && left != right)) {
			char* subStr = subString(str, left, right - 1);

			if (isKeyword(subStr) == true)
				printf("'%s' IS A KEYWORD\n", subStr);

			else if (isInteger(subStr) == true)
				printf("'%s' IS AN INTEGER\n", subStr);

			else if (isRealNumber(subStr) == true)
				printf("'%s' IS A REAL NUMBER\n", subStr);
```

```
                    else if (validIdentifier(subStr) == true
                                && isDelimiter(str[right - 1]) == false)
                        printf("'%s' IS A VALID IDENTIFIER\n", subStr);

                    else if (validIdentifier(subStr) == false
                                && isDelimiter(str[right - 1]) == false)
                        printf("'%s' IS NOT A VALID IDENTIFIER\n", subStr);
                    left = right;
                }
            }
            return;
}
int main()
{
            char str[100];
            printf("Enter  Expression : ");
    scanf("%s",str);
            parse(str);
            return (0);
}
```

Output:

```
Enter  Expression : a=b+1c
'a' IS A VALID IDENTIFIER
'=' IS AN OPERATOR
'b' IS A VALID IDENTIFIER
'+' IS AN OPERATOR
'1c' IS NOT A VALID IDENTIFIER
```

5. WAP in C language for removing Left Recursion from given grammar.

Code:

```
#include<stdio.h>
#include<string.h>
#define SIZE 10
int main () {
char non_terminal;
char  beta,alpha;
int num;
char production[10][SIZE];
int index=3; /* starting of the string following "->" */
printf("Enter Number of Production : ");
scanf("%d",&num);
printf("Enter the grammar as E->E-A :\n");
for(int i=0;i<num;i++){
scanf("%s",production[i]);
}
for(int i=0;i<num;i++){
printf("\nGRAMMAR : : : %s",production[i]);
non_terminal=production[i][0];
if(non_terminal==production[i][index]) {
alpha=production[i][index+1];
```

```
printf(" is left recursive.\n");
while(production[i][index]!=0 && production[i][index]!='|')
index++;
if(production[i][index]!=0) {
beta=production[i][index+1];
printf("Grammar without left recursion:\n");
printf("%c->%c%c\'",non_terminal,beta,non_terminal);
printf("\n%c\'->%c%c\'|E\n",non_terminal,alpha,non_terminal);
}
else
printf(" can't be reduced\n");
}
else
printf(" is not left recursive.\n");
index=3;
}
}
```

Output:

```
Enter Number of Production : 2
Enter the grammar as E->E-A :
E->EA|A
A->AT|a

GRAMMAR : : : E->EA|A is left recursive.
Grammar without left recursion:
E->AE'
E'->AE'|E

GRAMMAR : : : A->AT|a is left recursive.
Grammar without left recursion:
A->aA'
A'->TA'|E
```

6. WAP in C language for removing Left Factoring from given grammar.

Code:

```
#include<stdio.h>
#include<string.h>
int main()
 {
     char gram[20],part1[20],part2[20],modifiedGram[20],newGram[20],tempGram[20];
     int i,j=0,k=0,l=0,pos;
     printf("Enter Production : A->");
    gets(gram);
    for(i=0;gram[i]!='|';i++,j++)
        part1[j]=gram[i];
     part1[j]='\0';
     for(j=++i,i=0;gram[j]!='\0';j++,i++)
            part2[i]=gram[j];
     part2[i]='\0';
```

```
    for(i=0;i<strlen(part1)||i<strlen(part2);i++)
    {
        if(part1[i]==part2[i])
        {
            modifiedGram[k]=part1[i];
            k++;
            pos=i+1;
        }
    }
    for(i=pos,j=0;part1[i]!='\0';i++,j++){
        newGram[j]=part1[i];
    }
    newGram[j++]='|';
    for(i=pos;part2[i]!='\0';i++,j++){
        newGram[j]=part2[i];
    }
    modifiedGram[k]='X';
      modifiedGram[++k]='\0';
    newGram[j]='\0';
    printf("\n A->%s",modifiedGram);
    printf("\n X->%s\n",newGram);
}
```

Output:

```
Enter Production : A->aE+bcD|aE+eIT

 A->aE+X
 X->bcD|eIT
```

7. WAP in C language to compute First () for given grammar.

Code:

```
#include<stdio.h>
#include<ctype.h>
void FIRST(char[],char );
void addToResultSet(char[],char);
int num Productions;
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++)
   {
      printf("Enter productions Number %d : ",i+1);
      scanf(" %s",productionSet[i]);
   }
   do
   {
```

```
        printf("\n Find the FIRST of  :");
        scanf(" %c",&c);
        FIRST(result,c);
        printf("\n FIRST(%c)= { ",c);
        for(i=0;result[i]!='\0';i++)
        printf(" %c ",result[i]);
        printf("}\n");
         printf("press 'y' to continue : ");
        scanf(" %c",&choice);
    }
    while(choice=='y'||choice =='Y');
    }
void FIRST(char* Result,char c)
{
    int i,j,k;
    char subResult[20];
    int foundEpsilon;
    subResult[0]='\0';
    Result[0]='\0';
    if(!(isupper(c)))
    {
        addToResultSet(Result,c);
              return ;
    }
    for(i=0;i<numOfProductions;i++)
    {
        if(productionSet[i][0]==c)
        {
 if(productionSet[i][2]=='$') addToResultSet(Result,'$');
      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;
                   }
               if(!foundEpsilon)
                   break;
               j++;
              }
           }
    }
}
    return ;
}

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:

```
How many number of productions ? :8
Enter productions Number 1 : E=TD
Enter productions Number 2 : D=+TD
Enter productions Number 3 : D=$
Enter productions Number 4 : T=FS
Enter productions Number 5 : S=*FS
Enter productions Number 6 : S=$
Enter productions Number 7 : F=(E)
Enter productions Number 8 : F=a

 Find the FIRST of  :E

 FIRST(E)= {  (  a }
press 'y' to continue : y

 Find the FIRST of  :D

 FIRST(D)= {  +  $ }
press 'y' to continue : y

 Find the FIRST of  :T

 FIRST(T)= {  (  a }
press 'y' to continue : y

 Find the FIRST of  :S

 FIRST(S)= {  *  $ }
press 'y' to continue :
```

8. WAP in C language to compute Follow () for given grammar.

Code:

```
#include<stdio.h>
#include<string.h>
int n,m=0,p,i=0,j=0;
char a[10][10],f[10];
void follow(char c);
void first(char  c);
int main()
{
 int i,z;
 char c,ch;
 printf("Enter the no.of productions:");
 scanf("%d",&n);
 printf("Enter the productions(epsilon=$):\n");
 for(i=0;i<n;i++)
  scanf("%s%c",a[i],&ch);

 do
 {
  m=0;
  printf("Enter the element whose FOLLOW is to be found:");

  scanf("%c",&c);
  follow(c);
  printf("FOLLOW(%c) = { ",c);
  for(i=0;i<m;i++)
   printf("%c ",f[i]);
  printf(" }\n");
  printf("Do you want to continue(0/1)?");
  scanf("%d%c",&z,&ch);
```

```
 }
 while(z==1);
}
void follow(char c)
{

 if(a[0][0]==c)f[m++]='$';
 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;
                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];
                else first(a[k][2]);
                }
                }

}
```

Output:

```
Enter the no.of productions:8
Enter the productions(epsilon=$):
E=TD
D=+TD
D=$
T=FS
S=*FS
S=$
F=(E)
F=a
Enter the element whose FOLLOW is to be found:E
FOLLOW(E) = { $ )  }
Do you want to continue(0/1)?1
Enter the element whose FOLLOW is to be found:T
FOLLOW(T) = { + $ )  }
Do you want to continue(0/1)?1
Enter the element whose FOLLOW is to be found:S
FOLLOW(S) = { + $ )  }
Do you want to continue(0/1)?
```

9. WAP in C language to recognize strings under 'a', 'a*b+ ', 'abb'.

Code:

```
#include<stdio.h>
#include<conio.h>
#include<string.h>
#include<stdlib.h>
void main()
{
char s[20],c;
int state=0,i=0;

printf("\n Enter a string:");
gets(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("\n %s is not recognised.",s);
 exit(0);
 }
}
if(state==1)
 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);
 getch();
}
```

Output:

```
Enter a string:aaaabbbbb

aaaabbbbb is accepted under rule 'a*b+'
```

10. WAP in C language to identify whether a given line is a comment or not.

Code:

Create Binary Tree & Tree Traversal:

```
#include<stdlib.h>
#include<stdio.h>
struct node{
        int data;
        struct  node  *left;
        struct node *right;
};
typedef struct node;
void print_preorder(struct node * root){
   if (root){
      printf("%d\n",root->data);
      print_preorder(root->left);
      print_preorder(root->right);}
}

void print_inorder(struct node * root){
   if (root){
      print_inorder(root->left);
      printf("%d\n",root->data);
      print_inorder(root->right);}
}
void print_postorder(struct node * root){
   if (root){
      print_postorder(root->left);
      print_postorder(root->right);
      printf("%d\n",root->data);}
}
void main(){
   int i,n,val,ch;
   struct node *p , *q , *root;
   root = NULL;
   printf(" Create Binary Tree & Tree Traversal: ");
   printf(" \n 1. create \n 2. print_preorder \n 3. print_inorder \n 4. print_postorder \n 5. exit ");
   while (1){
      printf("\n enter your choice : ");
      scanf("%d",&ch);
   switch(ch){
 case 1:{
        printf("\n Enter the number of nodes : ");
        scanf("%d",&n);
        for(i=0;i<n;i++){
                p = (struct node*)malloc(sizeof(struct node));
                scanf("%d",&p->data);
                p->left = NULL;
                p->right = NULL;
                if(i == 0){
                        root = p;}
                else{
                        q = root;
                        while(1)
                                if(p->data > q->data){
                                        if(q->right == NULL){
```

```
                    q->right = p;
                    break;}
                else
                    q = q->right;}
            else{
                if(q->left == NULL){
                    q->left = p;
                    break;}
                else
                    q = q->left;}}}}
        break;}
    case 2:{
        printf("\n Pre Order Display : \n");
        print_preorder(root);
        break;}
    case 3:{
        printf("\n In Order Display : \n");
        print_inorder(root);
        break;}
    case 4:{
        printf("\n Post Order Display : \n");
        print_postorder(root);
        break;}
    case 5:{
        exit(0);}}}
    }
```

Output:

```
Enter comment://hello

It is a comment
```

11. WAP in C language to convert a regular expression into NFA.

Code:

```
#include<stdio.h>
#include<conio.h>
#include<string.h>
#include<ctype.h>
int r[100];
static int pos = 0;
static int sc = 0;
void nfa(int st, int p, char*s)
{
 int i,sp,fs[15],fsc=0;
sp=st;pos=p;sc=st;
while(*s!=NULL)
{
if(isalpha(*s))
{
 r[pos++]=sp;
r[pos++]=*s;
r[pos++]=++sc;}
if(*s=='.')
```

```
{
 sp=sc;
 r[pos++]=sc;
 r[pos++]=238;
 r[pos++]=++sc;
 sp=sc;
}
 if(*s=='|')
{
 sp=st;
 fs[fsc++]=sc;
}
 if(*s=='*')
{
 r[pos++]=sc;
 r[pos++]=238;
 r[pos++]=sp;
 r[pos++]=sp;
 r[pos++]=238;
 r[pos++]=sc;
}
 if(*s=='(')
{
  char ps[50];
  int i=0,flag=1;
  s++;
while(flag!=0)
{
  ps[i++]=*s;
 if(*s=='(')
 flag++;
 if(*s==')')
 flag--;
  s++;
}
 ps[--i]='\0';
 nfa(sc,pos,ps);
 s--;
}
 s++;
}
 sc++;
for(i=0;i<fsc;i++)
{
 r[pos++]=fs[i];
 r[pos++]=238;
 r[pos++]=sc;
}
 r[pos++]=sc-1;
 r[pos++]=238;
 r[pos++]=sc;
}
void main()
{
 int i;
 char *inp;
```

```
 printf("enter the regular expression :");
 gets(inp);
 nfa(1,0,inp);
 printf("\nstate intput state\n");
for(i=0;i<pos;i=i+3)
 printf("%d  --%c--> %d\n", r[i], r[i+1], r[i+2]);
 printf("\n");
getch();
}
```

Output:

```
enter the regular expression :a+b

state intput state
1  --a--> 2
1  --b--> 3
3  --ε--> 4
```

12. WAP in C language to count no. of white-spaces and new lines in a program segment.

Code:

```
#include <stdio.h>
void main()
{
    char str [81];
    int nletter, ndigit, nspace, nother;
    int i;
    printf("Enter a line of text:\n");
    gets(str);
    nletter = ndigit = nspace = nother = 0;
    i = 0;
    while (str[i] != '\0')
      {
        char ch= str[i];
        if (ch>= 'A' && ch<= 'Z' || ch>= 'a' && ch<= 'z')
        nletter++;
        else if (ch>= '0' && ch<= '9')
              ndigit++;
        else if (ch == ' ' || ch =='\n' || ch == '\t')
              nspace++;
        else nother++;
        i++;
      }
        printf("Letters: %d \tWhite spaces : %d", nletter, nspace);
        printf(" Digits : %d \tOther chars : %d\n", ndigit, nother);
        getch();
}
```

Output:

```
Enter a line of text:
1- Today is beautiful day but I am not see.(By Blind Person)
Letters: 44     White spaces : 11 Digits : 1    Other chars : 4
```

13. WAP in C language to implement Symbol Table for a given program segment.

Code:

```
#include<stdio.h>
#include<math.h>
#include<string.h>
#include<ctype.h>
#include<stdlib.h>
void main()
{
 int x=0, n, i=0,j=0;
 void *mypointer,*T4Tutorials_address[5];
 char ch,T4Tutorials_Search,T4Tutorials_Array2[15],T4Tutorials_Array3[15],c;
 printf("Input the expression ending with $ sign:");
 while((c=getchar())!='$')
 {
  T4Tutorials_Array2[i]=c;
  i++;
 }
 n=i-1;
 printf("Given Expression:");
 i=0;
 while(i<=n)
 {
  printf("%c",T4Tutorials_Array2[i]);
  i++;
 }
 printf("\n Symbol Table display\n");
 printf("Symbol \t addr \t type");
 while(j<=n)
 {
  c=T4Tutorials_Array2[j];
  if(isalpha(toascii(c)))
  {
   mypointer=malloc(c);
   T4Tutorials_address[x]=mypointer;
   T4Tutorials_Array3[x]=c;
   printf("\n%c \t %d \t identifier\n",c,mypointer);
   x++;
   j++;
  }
  else
  {
   ch=c;
   if(ch=='+'||ch=='-'||ch=='*'||ch=='=')
   {
    mypointer=malloc(ch);
    T4Tutorials_address[x]=mypointer;
    T4Tutorials_Array3[x]=ch;
    printf("\n %c \t %d \t operator\n",ch,mypointer);
    x++;
    j++;
   }}}}
```

Output:

```
Input the expression ending with $ sign:x=a+b
$
Given Expression:x=a+b

 Symbol Table display
Symbol    addr     type
x         6909008          identifier

 =        6909136          operator

a         6909216          identifier

 +        6909328          operator

b         6909392          identifier
```

14. WAP in C language to test whether a given identifier is valid or not.

```
#include<stdio.h>
#include<conio.h>
#include<ctype.h>
void main()
{
 char a[10]
 int flag, i=1
 clrscr()
 printf("\n Enter an identifier:")
 gets(a)
 if(isalpha(a[0])) flag=1
 else printf("\n Not a valid identifier")
 while(a[i]!='\0') { if(!isdigit(a[i])&&!isalpha(a[i])) { flag=0
 break
 } i++
 } if(flag==1) printf("\n Valid identifier")
 getch()
}
```

OUTPUT

```
Enter an identifier: first

Valid identifier

Enter an identifier: 1aqw

Not a valid identifier
```

15.WAP in C language to find out whether the given grammar is LL(1) or not ,Grammar: E->TE', E'->+TE'| ε, T->FT', T'->*FT'| ε, F->(E)|id.

```
#include<stdio.h>
#include<conio.h>
#include<ctype.h>
char ch;
#define id 0
#define CONST 1
#define mulop 2
#define addop 3
#define op 4
#define cp 5
#define err 6
#define col 7
#define size 50
int token;
char lexbuff[size];
int lookahead=0;
int main() {
clrscr();
 printf(" Enter the string :");
 gets(lexbuff);
 parser();
return 0; }
parser() {
if(E())
printf("valid string");
else
printf("invalid string");
getch();
return 0; }
E() {
if(T()) {
 if(EPRIME())
return 1;
 else
 return 0;

}
 else
return 0; }
T() {
 if(F())

{
 if(TPRIME())
 return 1;
 else
 return 0;

}
 else
 return 0; }
```

```
EPRIME() {
 token=lexer();
if(token==addop)

{
 lookahead++;
 if(T())

{
 if(EPRIME())
 return 1;
 else
 return 0;

}
 else
 return 0;

}
els
e
 return 1; }
TPRIME() {
 token=lexer();
 if(token==mulop)

{
 lookahead++;
 if(F())

{
 if(TPRIME())
 return 1;
 else
 return 0;

}
 else
return 0;

}
 else
return 1;
}
F() {
 token=lexer();
 if(token==id)
 return 1;
 else

{
 if(token==4)

{
 if(E())
```

```
{
if(token==5)
return  1;
else
return 0;

}
 else
 return 0;

}
 else
 return 0; }}
lexer() {
 if(lexbuff[lookahead]!='
\n')

{
 while(lexbuff[lookahead]=='
\t')
 lookahead++;
 if(isalpha(lexbuff[lookahead]))

{
 while(isalnum(lexbuff[lookahead]))
 lookahead++;
 return(id);

}
 else

{
 if(isdigit(lexbuff[lookahead]))

{
while(isdigit(lexbuff[lookahead]))
lookahead++;
return CONST;

}
 else

{
 if(lexbuff[lookahead]=='+')

{
return(addop);

}
 else

{
if(lexbuff[lookahead]=='*') {
 return(mulop); }
else {
 if(lexbuff[lookahead]=='(')
```

```
{
 lookahead++;
 return(op);

}
 else
{
 if(lexbuff[lookahead]==')')
 {
 return(op);
 }
 else
 {
 return(err);
 }
 }
 }
 }
}
 }
 }
 else
 return (col);
 }
```

OUTPUT