lps

1.AIM : DFA
SOURCE CODE :

```
#include
#define max 100
void 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;
else if(str[i]=='c') f=1;
break;
case 2: if(str[i]=='b') f=3;
else if(str[i]=='a') f=2;
else if(str[i]=='c') f=1;
break;
case 3: if(str[i]=='c') f=4;
else if(str[i]=='a') f=2;
else if(str[i]=='b') f=1;
break;
case 4:if(str[i]=='c') f=1;
else if(str[i]=='a') f=2;
else if(str[i]=='b') f=3;
break;
}
}
if(f==4)
printf("String is accepted");
else printf("String is not accepted");
}
```

OUTPUT :

```
enter the string to be checked: aabbcaabc
String is accepted
```

2.AIM : LEXICAL ANALYZER

SOUCE CODE:

```
#include
#include
#include
#include
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,"printf")||
!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);
}

// Extracts the SUBSTRING.
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]);
else
printf("%c IS A DELIMITER\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 the string : ");
scanf("%[^\n]%*c", str);

parse(str);

return (0);
}
```

OUTPUT :

```
Enter the string : printf("Hello");
'printf' IS A KEYWORD
( IS A DELIMITER
'"Hello"' IS A VALID IDENTIFIER
) IS A DELIMITER
; IS A DELIMITER
```

---

3. AIM : Left Recursion

SOURCE CODE

```
#include
#include
#define SIZE 10
int main()
{
char nt;
char b,a;
int num;
int i;
char p[10][SIZE];
int index=3;
printf("Enter Number of productions:");
scanf("%d",&num);
printf("Enter the grammar as E->E-A:\n");
for(i=0;i scanf("%s",p[i]);
for(i=0;i {
printf("\nGRAMMAR:%s",p[i]);
nt=p[i][0];
if(nt==p[i][index])
{
a=p[i][index];
printf("is left recursive:\n");
while(p[i][index]!=0 && p[i][index]!='|')
{
index++;
```

```
}
if(p[i][index]!=0)
{
b=p[i][index+1];
printf("Grammar without left recursion;\n");
printf("%c->%c%c\'\n",nt,b,nt);
printf("\n%c\'->%c%c\'|E\n",nt,a,nt);
}
else
printf("can't be reduced\n");
}
else
printf("is not left recursive\n");
index=3;
}
}
```

OUTPUT:

```
Enter Number of productions:2
Enter the grammar as E->E-A:
E->E+E|T
E->A|B

GRAMMAR:E->E+E|Tis left recursive:
Grammar without left recursion;
E->TE'
E'->EE'|E

GRAMMAR:E->A|Bis not left recursive
```

---

4. AIM : LEFT FACTORING

SOURCE CODE

```
#include
#include
int main(){
char g[20],a[20],b[20],mg[20],ng[20],tg[20];
int i,j=0,k=0,l=0,pos;
printf("Enter Production : A->");
scanf("%s",g);
for(i=0;g[i]!='|';i++,j++)
a[j]=g[i];
a[j]='\0';
for(j=++i,i=0;g[j]!='\0';j++,i++)
b[i]=g[j];
b[i]='\0';
for(i=0;i if(a[i]==b[i]){
mg[k]=a[i];
k++;
pos=i+1;
}
}
for(i=pos,j=0;a[i]!='\0';i++,j++)
ng[j]=a[i];

ng[j++]='|';

for(i=pos;b[i]!='\0';i++,j++)
ng[j]=b[i];

mg[k]='X';
mg[++k]='\0';
ng[j]='\0';

printf("\ngmar Without Left Factoring : : \n");
printf(" A->%s",mg);
printf("\n X->%s\n",ng);
}
```

OUTPUT :

```
Enter Production : A->bE+Ab|bE+hg

grammar Without Left Factoring :
```

A->bE+X
X->Ab|hg

---

5. Aim : First and Follow

SOURCE CODE :

```
#include
#include
#include
#include
#include
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:\n");
scanf("%d",&n);
printf("Enter the productions:\n");
for(i=0;i scanf("%s%c",a[i],&ch);
do
{
m=0;
printf("Enter the elements whose first and follow is to be found:");
scanf("%c",&c);
first(c);
printf("first(%c)={",c);
for(i=0;i printf("%c",f[i]);
printf("}\n");
strcpy(f," ");
m=0;
follow(c);
printf("follow(%c)={",c);
for(i=0;i printf("%c",f[i]);
printf("}\n");
printf("continue(0/1)?");
scanf("%d%c",&z,&ch);
}while(z==1);
return(0);
}
void first(char c)
{
int k;
if(!isupper(c))
f[m++]=c;
for(k=0;k {
if(a[k][0]==c)
{
if(a[k][2]=='$')
follow(a[k][0]);
else if(islower(a[k][2]))
f[m++]=a[k][2];
else first(a[k][2]);
}
}
}
void follow(char c)
{
if(a[0][0]==c)
f[m++]='$';
for(i=0;i {
for(j=2;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]);
}
}
}
}
```

OUTPUT:

```
Enter the no of productions:5
Enter the productions:
S=ABCD
A=Cf
A=a
C=gE
E=h
Enter the elements whose first and follow is to be found:S
first(S)={ga}
follow(S)={$}
continue(0/1)?1
Enter the elements whose first and follow is to be found:A
first(A)={ga}
follow(A)={}
continue(0/1)?0
```

6.Aim : Non-Recursive Predictive Parser

Source Code:

```
#include
#include
#include
char s[30],st[30];
void main()
{
char tab[5][6][5]=
{"ta","@","@","ta","@","@","@","+ta","@","@","!","!","fb","@","@","fb","@","@","@","!","*fb","@","!","!","i","@","@","
(e)","@","@"};
printf("Enter the string:\n");
scanf("%s",s);
strcat(s,"$");
st[0]='$';
st[1]='e';
int st_i,s_i,i,j,k,n,s1,s2;
st_i=1;
s_i=0;
char temp[20];
printf("\nStack Input\n");
while(st[st_i]!='$' || s[s_i]!='$')
{
switch(st[st_i])
{
case 'e':s1=0;break;
case 'a':s1=1;break;
case 't':s1=2;break;
case 'b':s1=3;break;
case 'f':s1=4;break;
default:s1=-1;
}
switch(s[s_i])
{
case 'i':s2=0;break;
case '+':s2=1;break;
case '*':s2=2;break;
case '(':s2=3;break;
case ')':s2=4;break;
case '$':s2=5;break;
default:s2=-1;
}
if(s1==-1 || s2==-1)
{
printf("Failure");
exit(0);
}
if(tab[s1][s2]=="@")
{
printf("Failure");
exit(0);
}
if(tab[s1][s2][0]=='!')
```

```
{
st[st_i]='\0';
st_i--;
}
else
{
j=strlen(tab[s1][s2]);
for(k=0;tab[s1][s2][k]!='\0';k++)
{
temp[j-k-1]=tab[s1][s2][k];
}
temp[j]='\0';
st[st_i]='\0';
strcat(st,temp);
st_i=strlen(st)-1;
}
printf("%s ",st);
for(n=s_i;s[n]!='\0';n++)
printf("%c",s[n]);
printf("\n");
if(st[st_i]==s[s_i] && s[s_i]!='$')
{
st[st_i]='\0';
s_i++;
st_i--;
}
}
printf("Success");
}
```

```
OUTPUT:
Enter the string:
i*i+i

Stack Input
$at i*i+i$
$abf i*i+i$
$abi i*i+i$
$abf* *i+i$
$abi i+i$
$a +i$
$at+ +i$
$abf i$
$abi i$
$a $
$ $
Success
```

7.Aim : Shift-Reduce Parser using stack
Source Code:

```
#include
#include
int k=0,z=0,i=0,j=0,c=0;
char a[16],ac[20],stk[15],act[10];
void check();
int main()
{
puts("GRAMMAR is E->E+E \n E->E*E \n E->(E) \n E->id");
puts("enter input string ");
scanf("%s",a);
c=strlen(a);
strcpy(act,"SHIFT->");
puts("stack \t input \t action");
for(k=0,i=0; j {
if(a[j]=='i' && a[j+1]=='d')
{
stk[i]=a[j];
stk[i+1]=a[j+1];
stk[i+2]='\0';
a[j]=' ';
a[j+1]=' ';
printf("\n$%s\t%s$\t%sid",stk,a,act);
check();
}
else
{
```

```
stk[i]=a[j];
stk[i+1]='\0';
a[j]=' ';
printf("\n$%s\t%s$\t%ssymbols",stk,a,act);
check();
}
}
}
void check() {
strcpy(ac,"REDUCE TO E");
for(z=0; z if(stk[z]=='i' && stk[z+1]=='d'){
stk[z]='E';
stk[z+1]='\0';
printf("\n$%s\t%s$\t%s",stk,a,ac);
j++;
}
for(z=0; z if(stk[z]=='E' && stk[z+1]=='+' && stk[z+2]=='E'){
stk[z]='E';
stk[z+1]='\0';
stk[z+2]='\0';
printf("\n$%s\t%s$\t%s",stk,a,ac);
i=i-2;
}
for(z=0; z if(stk[z]=='E' && stk[z+1]=='*' && stk[z+2]=='E'){
stk[z]='E';
stk[z+1]='\0';
stk[z+1]='\0';
printf("\n$%s\t%s$\t%s",stk,a,ac);
i=i-2;
}
for(z=0; z if(stk[z]=='(' && stk[z+1]=='E' && stk[z+2]==')'){
stk[z]='E';
stk[z+1]='\0';
stk[z+1]='\0';
printf("\n$%s\t%s$\t%s",stk,a,ac);
i=i-2;
}
}


OUTPUT:
GRAMMAR is E->E+E
E->E*E
E->(E)
E->id
enter input string
id+id*id+id
stack input action

$id +id*id+id$ SHIFT->id
$E +id*id+id$ REDUCE TO E
$E+ id*id+id$ SHIFT->symbols
$E+id *id+id$ SHIFT->id
$E+E *id+id$ REDUCE TO E
$E *id+id$ REDUCE TO E
$E* id+id$ SHIFT->symbols
$E*id +id$ SHIFT->id
$E*E +id$ REDUCE TO E
$E +id$ REDUCE TO E
$E+ id$ SHIFT->symbols
$E+id $ SHIFT->id
$E+E $ REDUCE TO E
$E $ REDUCE TO Ec

--------------------------------------------------------------------------------
8.Aim : Operator Precendence Parsing
Source Code:

#include
#include
#include
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 {
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 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

STACK INPUT ACTION
$i *(i+i)*i$ Shift
$E *(i+i)*i$ Reduced: E->i
$E* (i+i)*i$ Shift
$E*( i+i)*i$ Shift
$E*(i +i)*i$ Shift
$E*(E +i)*i$ Reduced: E->i
$E*(E+ i)*i$ Shift
$E*(E+i )*i$ Shift
$E*(E+E )*i$ Reduced: E->i
$E*(E )*i$ Reduced: E->E+E
$E*(E) *i$ Shift
$E*E *i$ Reduced: E->)E(
$E *i$ Reduced: E->E*E
$E* i$ Shift
$E*i $ Shift
$E*E $ Reduced: E->i
$E $ Reduced: E->E*E
$E$ Shift
$E$ Shift
Accepted;
--------------------------------------------------------------------------------
9.Aim : Stack Allocation Strategy
```

Source Code:

```
#include
int stack[100],ch,n,top,x,i;
void push(void);
void pop(void);
void display(void);
int main(){
top=-1;
printf("Enter the size of stack[MAX=100]");
scanf("%d",&n);
printf("\n\t stack opertaions:");
printf("\n\t-------------------------:");
printf("\n\t 1.push\t2.pop\t 3.display\t4.EXIT\t");
do{
printf("\n Enter the choice");
scanf("%d",&ch);
switch(ch)
{
case 1:push();break;
case 2:pop();break;
case 3:display();break;
case 4:printf("\n\tExit");break;
default:printf("Please nter a valid choice:");
}
}while(ch!=4);
return 0;
}

void push(){
if(top>=n-1)
printf("\n\n stack overflow");
else{
printf("Enter a value to be pushed");
scanf("%d",&x);
top++;
stack[top]=x;
}
}

void pop(){
if(top==-1)
printf("\n\t stack underflow");
else{
printf("\n\t the popped element is %d",stack[top]);
top--;
}
}

void display()
{
if(top>=0){
printf("\n The elements in stack\n");
for(i=top;i>=0;i--)
printf("\n%d",stack[i]);
printf("\n Select next choice");
}
else
printf("\n the stack is empty");
}


OUTPUT:
Enter the size of stack[MAX=100]10

stack opertaions:
-------------------------:
1.push 2.pop 3.display 4.EXIT
Enter the choice1
Enter a value to be pushed10

Enter the choice1
Enter a value to be pushed20

Enter the choice1
Enter a value to be pushed30

Enter the choice3

The elements in stack
```

```
30
20
10
Select next choice
Enter the choice2

the popped element is 30
Enter the choice3

The elements in stack

20
10
Select next choice
Enter the choice4

Exit

--------------------------------------------------------------------------------
10.Aim : Intermediate Code generation
Source Code:

#include

#include
int i=1,j=0,no=0,tmpch=90;
char str[100],left[15],right[15];
void findopr();
void explore();
void fleft(int);
void fright(int);
struct exp
{
int pos;
char op;
}k[15];

void main()
{

printf("\t\tIntermediate code generation\n:");
printf("Enter the expression:");
scanf("%s",str);
printf("The Intermediate code:\t\t Expression\n");
findopr();
explore();

}
void findopr(){
for(i=0;str[i]!='\0';i++)
if(str[i]==':'){
k[j].pos=i;
k[j++].op=':';
}

for(i=0;str[i]!='\0';i++)
if(str[i]=='/'){
k[j].pos=i;
k[j++].op='/';
}
for(i=0;str[i]!='\0';i++)
if(str[i]=='*'){
k[j].pos=i;
k[j++].op='*';
}
for(i=0;str[i]!='\0';i++)
if(str[i]=='-'){
k[j].pos=i;
k[j++].op='-';
}
for(i=0;str[i]!='\0';i++)
if(str[i]=='+'){
k[j].pos=i;
k[j++].op='+';
}
}

void explore(){
i=1;
```

```
while(k[i].op!='\0'){
fleft(k[i].pos);
fright(k[i].pos);
str[k[i].pos]=tmpch--;
printf("\t %c:=%s%c%s\t\t",str[k[i].pos],left,k[i].op,right);
for(j=0;j if(str[j]!='$')
printf("%c",str[j]);
printf("\n");
i++;
}
fright(-1);
if(no==0){
fleft(strlen(str));
printf("\t %s:=%s",right,left);
}
printf("\t %s:=%c",right,str[k[--i].pos]);

}

void fleft(int x){
int w=0,flag=0;
x--;
while(x!=-1 && str[x]!='+' && str[x]!='*' && str[x]!='=' && str[x]!='\0' && str[x]!='-' && str[x]!='/' &&
str[x]!=':'){
if(str[x]!='$' && flag==0)
{
left[w++]=str[x];
left[w]='\0';
str[x]='$';
flag=1;
}
x--;
}
}
void fright(int x){
int w=0,flag=0;
x++;
while(x!=-1 && str[x]!='+' && str[x]!='*' && str[x]!='=' && str[x]!='\0' && str[x]!='-' && str[x]!='/' &&
str[x]!=':'){
if(str[x]!='$'&& flag==0)
{
right[w++]=str[x];
right[w]='\0';
str[x]='$';
flag=1;
}
x++;
}
}
```

OUTPUT:

```
Intermediate code generation
:Enter the expression:w:=a*b+c/d-e/f+g*h
The Intermediate code: Expression
Z:=c/d w:=a*b+Z-e/f+g*h
Y:=e/f w:=a*b+Z-Y+g*h
X:=a*b w:=X+Z-Y+g*h
W:=g*h w:=X+Z-Y+W
V:=Z-Y w:=X+V+W
U:=X-V w:=U+W
T:=U-W w:=T
w:=T w:=
```