**Practical-1**

**Aim:** Write a Program to implement lexical analyzer

**Program:**

#include <stdbool.h>

#include <stdio.h>

#include <string.h>

#include <stdlib.h>

bool delimiter (char ch)

{

if (ch == ' ' || ch == '+' || ch == '-' || ch == '\*' ||

ch == '/' || ch == ',' || ch == ';' || ch == '>' ||

ch == '<' || ch == '=' || ch == '(' || ch == ')' ||

ch == '[' || ch == ']' || ch == '{' || ch == '}')

return (true);

return (false);

}

bool

operators (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' || delimiter (str[0]) == true)

return (false);

return (true);

}

bool keyword (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 integer (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);

}

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 test (char \*str)

{

int left = 0, right = 0;

int len = strlen (str);

while (right <= len && left <= right)

{

if (delimiter (str[right]) == false)

right++;

if (delimiter (str[right]) == true && left == right)

{

if (operators (str[right]) == true)

printf ("'%c' is an operator \n", str[right]);

right++;

left = right;

}

else if (delimiter (str[right]) == true && left != right

|| (right == len && left != right))

{

char \*subStr = subString (str, left, right - 1);

if (keyword (subStr) == true)

printf ("'%s' is a keyword \n", subStr);

else if (valididentifier (subStr) == true

&& delimiter (str[right - 1]) == false)

printf ("'%s' is an identifier \n", subStr);

else if (integer (subStr) == true)

printf ("'%s' is an integer \n", subStr);

else if (valididentifier (subStr) == false

&& delimiter (str[right - 1]) == false)

printf ("'%s'is an invalid identifier \n", subStr);

left = right;

}

}

return;

}

int main ()

{

char str[100];

printf ("\nEnter a string :");

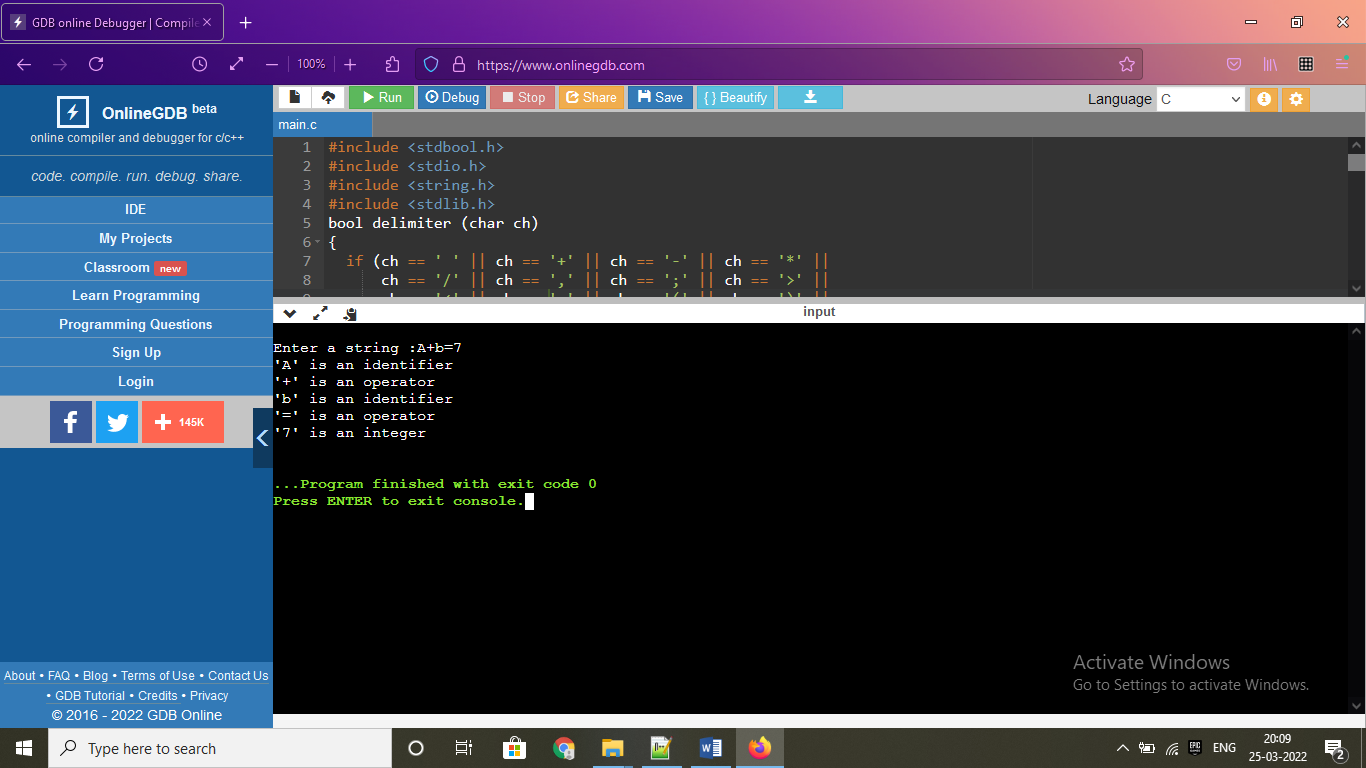
scanf ("%[^\n]%\*c", str);

test (str);

return (0);

}

**Output:**



**Practical-2**

**Aim:** Write a Program to find first of a production

**Program:**

#include<stdio.h>

#include<ctype.h>

void F(char );

int count,n=0;

char T[10][10], first[10];

void main()

{

int i,choice;

char c,ch;

printf("How many productions do yo want :");

scanf("%d",&count);

printf("Enter %d productions epsilon= e :\n\n",count);

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

scanf("%s%c",T[i],&ch);

do

{

n=0;

printf("Enter Element that you want the first of :");

scanf("%c",&c);

F(c);

printf("\n first(%c)= { ",c);

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

printf("%c ",first[i]);

printf("}\n");

printf("press 1 to continue or 2 to exit : ");

scanf("%d%c",&choice,&ch);

}

while(choice==1);

}

void F(char c)

{

int j;

if(!(isupper(c)))

first[n++]=c;

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

{

if(T[j][0]==c)

{

if(T[j][2]=='e')

first[n++]='e';

else if(islower(T[j][2]))

first[n++]=T[j][2];

else

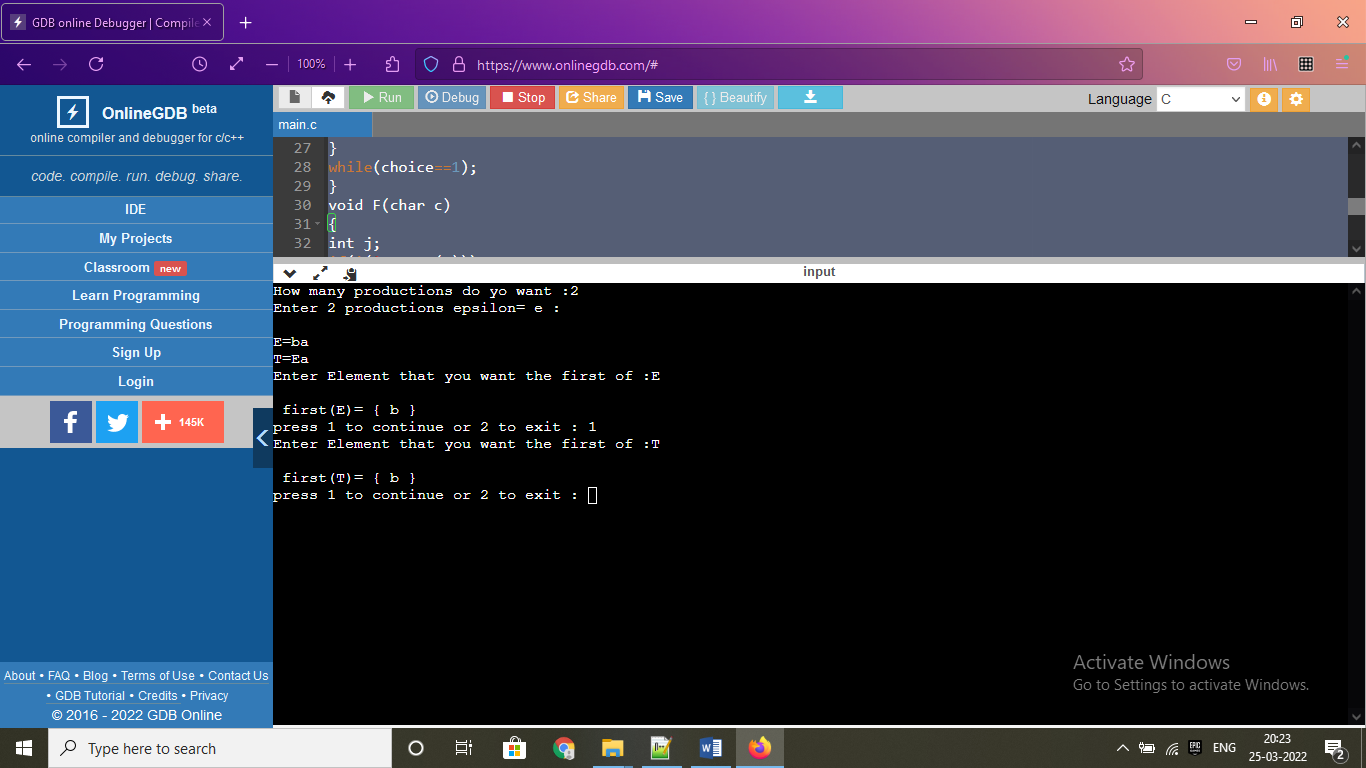
F(T[j][2]);

}

}

}

**Output:**



**Practical-3**

**Aim:** Write a Program to find follow of a production

**Program:**

#include<stdio.h>

#include<string.h>

#include<ctype.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);

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)))

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]))

addToResult(a[k][2]);

else first(a[k][2]);

}

}

}

void addToResult(char c)

{

int i;

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

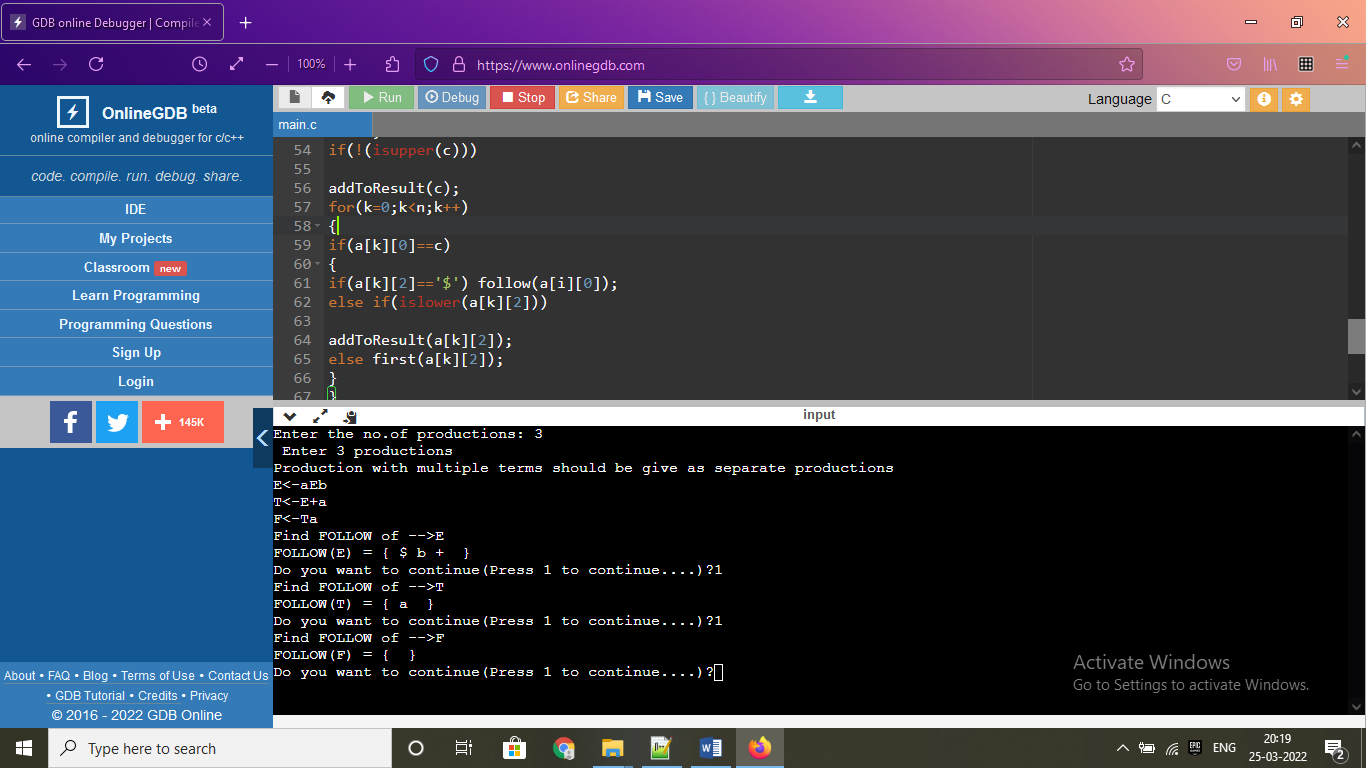
if(followResult[i]==c)

return;

followResult[m++]=c;

}

**Output:**



**Practical-4**

**Aim:** Write a Program to find leading and trailing of a production

**Program:**

#include<iostream>

#include<string.h>

#include<conio.h>

using namespace std;

int nt, t, top = 0;

char s[50], NT[10], T[10], st[50], l[10][10], tr[50][50];

int searchnt (char a)

{

int count = -1, i;

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

{

if (NT[i] == a)

return i;

}

return count;

}

int searchter (char a)

{

int count = -1, i;

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

{

if (T[i] == a)

return i;

}

return count;

}

void push (char a)

{

s[top] = a;

top++;

}

char pop ()

{

top--;

return s[top];

}

void installl (int a, int b)

{

if (l[a][b] == 'f')

{

l[a][b] = 't';

push (T[b]);

push (NT[a]);

}

}

void installt (int a, int b)

{

if (tr[a][b] == 'f')

{

tr[a][b] = 't';

push (T[b]);

push (NT[a]);

}

}

int main ()

{

int i, s, k, j, n;

char pr[30][30], b, c;

cout << "Enter the no of productions:";

cin >> n;

cout << "Enter the productions one by one\n";

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

cin >> pr[i];

nt = 0;

t = 0;

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

{

if ((searchnt (pr[i][0])) == -1)

NT[nt++] = pr[i][0];

}

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

{

for (j = 3; j < strlen (pr[i]); j++)

{

if (searchnt (pr[i][j]) == -1)

{

if (searchter (pr[i][j]) == -1)

T[t++] = pr[i][j];

}

}

}

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

{

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

l[i][j] = 'f';

}

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

{

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

tr[i][j] = 'f';

}

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

{

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

{

if (NT[(searchnt (pr[j][0]))] == NT[i])

{

if (searchter (pr[j][3]) != -1)

installl (searchnt (pr[j][0]), searchter (pr[j][3]));

else

{

for (k = 3; k < strlen (pr[j]); k++)

{

if (searchnt (pr[j][k]) == -1)

{

installl (searchnt (pr[j][0]), searchter (pr[j][k]));

break;

}

}

}

}

}

}

while (top != 0)

{

b = pop ();

c = pop ();

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

{

if (pr[s][3] == b)

installl (searchnt (pr[s][0]), searchter (c));

}

}

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

{

cout << "Leading of " << NT[i] << " : " << "{";

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

{

if (l[i][j] == 't')

cout << T[j] << ",";

}

cout << "}\n";

}

top = 0;

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

{

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

{

if (NT[searchnt (pr[j][0])] == NT[i])

{

if (searchter (pr[j][strlen (pr[j]) - 1]) != -1)

installt (searchnt (pr[j][0]),searchter (pr[j][strlen (pr[j]) - 1]));

else

{

for (k = (strlen (pr[j]) - 1); k >= 3; k--)

{

if (searchnt (pr[j][k]) == -1)

{

installt (searchnt (pr[j][0]), searchter (pr[j][k]));

break;

}

}

}

}

}

}

while (top != 0)

{

b = pop ();

c = pop ();

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

{

if (pr[s][3] == b)

installt (searchnt (pr[s][0]), searchter (c));

}

}

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

{

cout << "Trailing of " << NT[i] << " : " << "\t{";

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

{

if (tr[i][j] == 't')

cout << T[j] << ",";

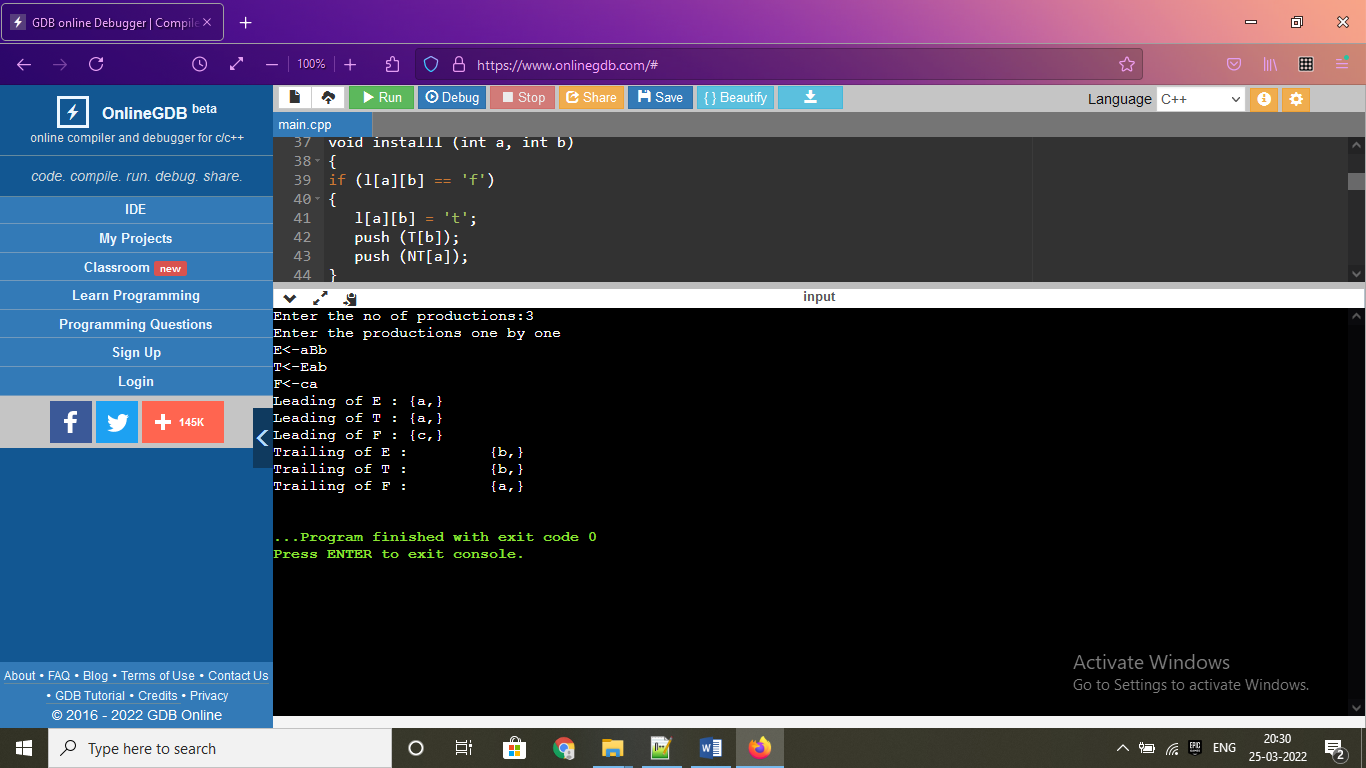
}

cout << "}\n";

}

}

**Output:**



**Practical-5**

**Aim:** Write a Program to check for and remove left recursion

**Program:**

#include<stdio.h>

#include<string.h>

#define Length 10

int main ()

{

char nt;

char b, a;

int n;

char prod[10][Length];

int index = 2;

printf ("Enter Number of Productions that you want : ");

scanf ("%d", &n);

printf ("Enter the productions :\n");

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

{

scanf ("%s", prod[i]);

}

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

{

printf ("\nThe input grammer : %s", prod[i]);

nt = prod[i][0];

if (nt == prod[i][index])

{

a = prod[i][index + 1];

printf (" has left recursion \n");

while (prod[i][index] != 0 && prod[i][index] != '|')

index++;

if (prod[i][index] != 0)

{

b = prod[i][index + 1];

printf ("After solving the left recursion :\n");

printf ("%c=%c%c\'", nt, b, nt);

printf ("\n%c\'=%c%c\'|ε \n", nt, a,nt);

}

else

printf (" give proper input \n");

}

else

printf (" doesn't have left recursion.\n");

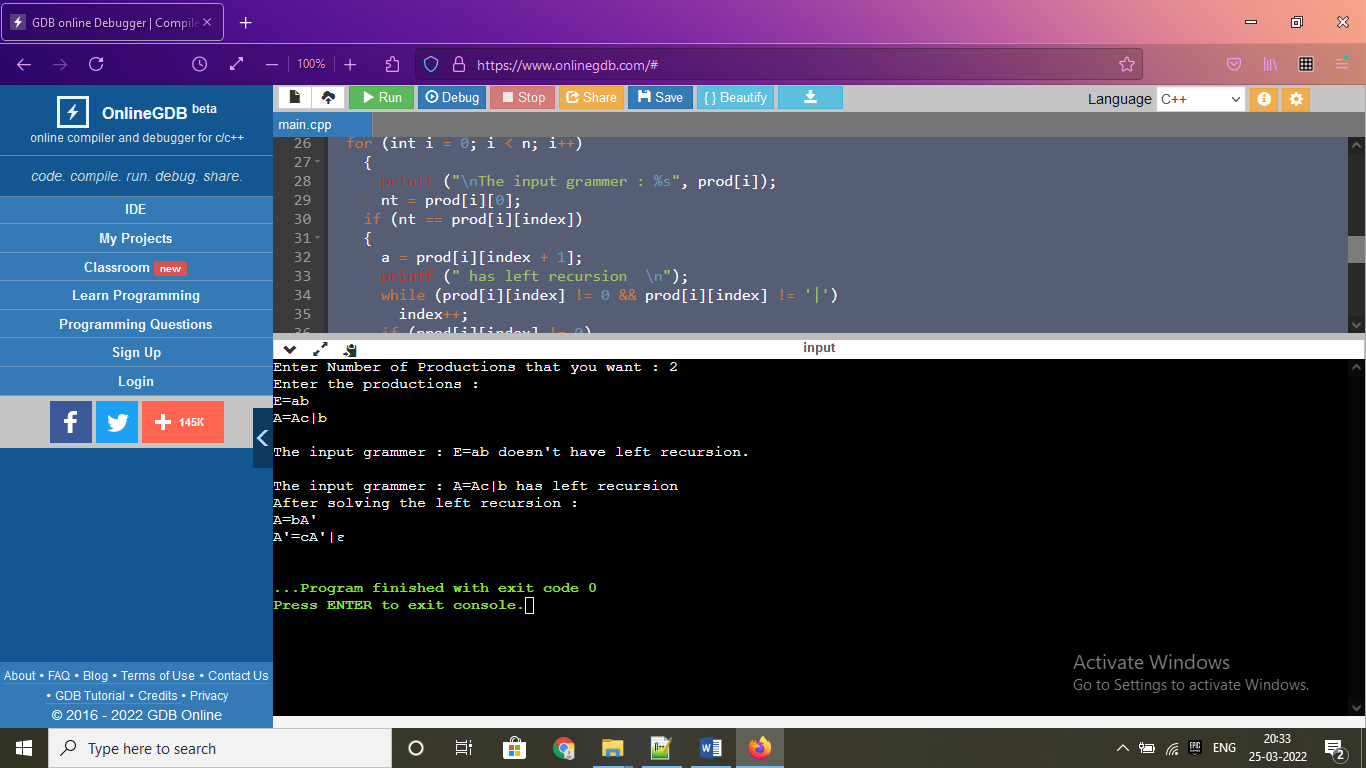
index = 2;

}

return 0;

}

**Output:**



**Practical-6**

**Aim: Program to implement Predictive Parsing**

#include<stdio.h>

#include<conio.h>

#include<string.h>

void main()

{

char fin[10][20],st[10][20],ft[20][20],fol[20][20];

int a=0,e,i,t,b,c,n,k,l=0,j,s,m,p;

printf("enter the no. of coordinates\n");

scanf("%d",&n);

printf("enter the productions in a grammar\n");

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

scanf("%s",st[i]);

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

fol[i][0]='\0';

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

{

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

{

j=3;

l=0;

a=0;

l1:if(!((st[i][j]>64)&&(st[i][j]<91)))

{

for(m=0;m<l;m++)

{

if(ft[i][m]==st[i][j])

goto s1;

}

ft[i][l]=st[i][j];

l=l+1;

s1:j=j+1;

}

else

{

if(s>0)

{

while(st[i][j]!=st[a][0])

{

a++;

}

b=0;

while(ft[a][b]!='\0')

{

for(m=0;m<l;m++)

{

if(ft[i][m]==ft[a][b])

goto s2;

}

ft[i][l]=ft[a][b];

l=l+1;

s2:b=b+1;

}

}

}

while(st[i][j]!='\0')

{

if(st[i][j]=='|')

{

j=j+1;

goto l1;

}

j=j+1;

}

ft[i][l]='\0';

}

}

printf("first pos\n");

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

printf("FIRS[%c]=%s\n",st[i][0],ft[i]);

fol[0][0]='$';

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

{

k=0;

j=3;

if(i==0)

l=1;

else

l=0;

k1:while((st[i][0]!=st[k][j])&&(k<n))

{

if(st[k][j]=='\0')

{

k++;

j=2;

}

j++;

}

j=j+1;

if(st[i][0]==st[k][j-1])

{

if((st[k][j]!='|')&&(st[k][j]!='\0'))

{

a=0;

if(!((st[k][j]>64)&&(st[k][j]<91)))

{

for(m=0;m<l;m++)

{

if(fol[i][m]==st[k][j])

goto q3;

}

fol[i][l]=st[k][j];

l++;

q3:

j++;

}

else

{

while(st[k][j]!=st[a][0])

{

a++;

}

p=0;

while(ft[a][p]!='\0')

{

if(ft[a][p]!='@')

{

for(m=0;m<l;m++)

{

if(fol[i][m]==ft[a][p])

goto q2;

}

fol[i][l]=ft[a][p];

l=l+1;

}

else

e=1;

q2:p++;

}

if(e==1)

{

e=0;

goto a1;

}

}

}

else

{

a1:c=0;

a=0;

while(st[k][0]!=st[a][0])

{

a++;

}

while((fol[a][c]!='\0')&&(st[a][0]!=st[i][0]))

{

for(m=0;m<l;m++)

{

if(fol[i][m]==fol[a][c])

goto q1;

}

fol[i][l]=fol[a][c];

l++;

q1:c++;

}

}

goto k1;

}

fol[i][l]='\0';

}

printf("follow pos\n");

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

printf("FOLLOW[%c]=%s\n",st[i][0],fol[i]);

printf("\n");

s=0;

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

{

j=3;

while(st[i][j]!='\0')

{

if((st[i][j-1]=='|')||(j==3))

{

for(p=0;p<=2;p++)

{

fin[s][p]=st[i][p];

}

t=j;

for(p=3;((st[i][j]!='|')&&(st[i][j]!='\0'));p++)

{

fin[s][p]=st[i][j];

j++;

}

fin[s][p]='\0';

if(st[i][k]=='@')

{

b=0;

a=0;

while(st[a][0]!=st[i][0])

{

a++;

}

while(fol[a][b]!='\0')

{

printf("M[%c,%c]=%s\n",st[i][0],fol[a][b],fin[s]);

b++;

}

}

else if(!((st[i][t]>64)&&(st[i][t]<91)))

printf("M[%c,%c]=%s\n",st[i][0],st[i][t],fin[s]);

else

{

b=0;

a=0;

while(st[a][0]!=st[i][3])

{

a++;

}

while(ft[a][b]!='\0')

{

printf("M[%c,%c]=%s\n",st[i][0],ft[a][b],fin[s]);

b++;

}

}

s++;

}

if(st[i][j]=='|')

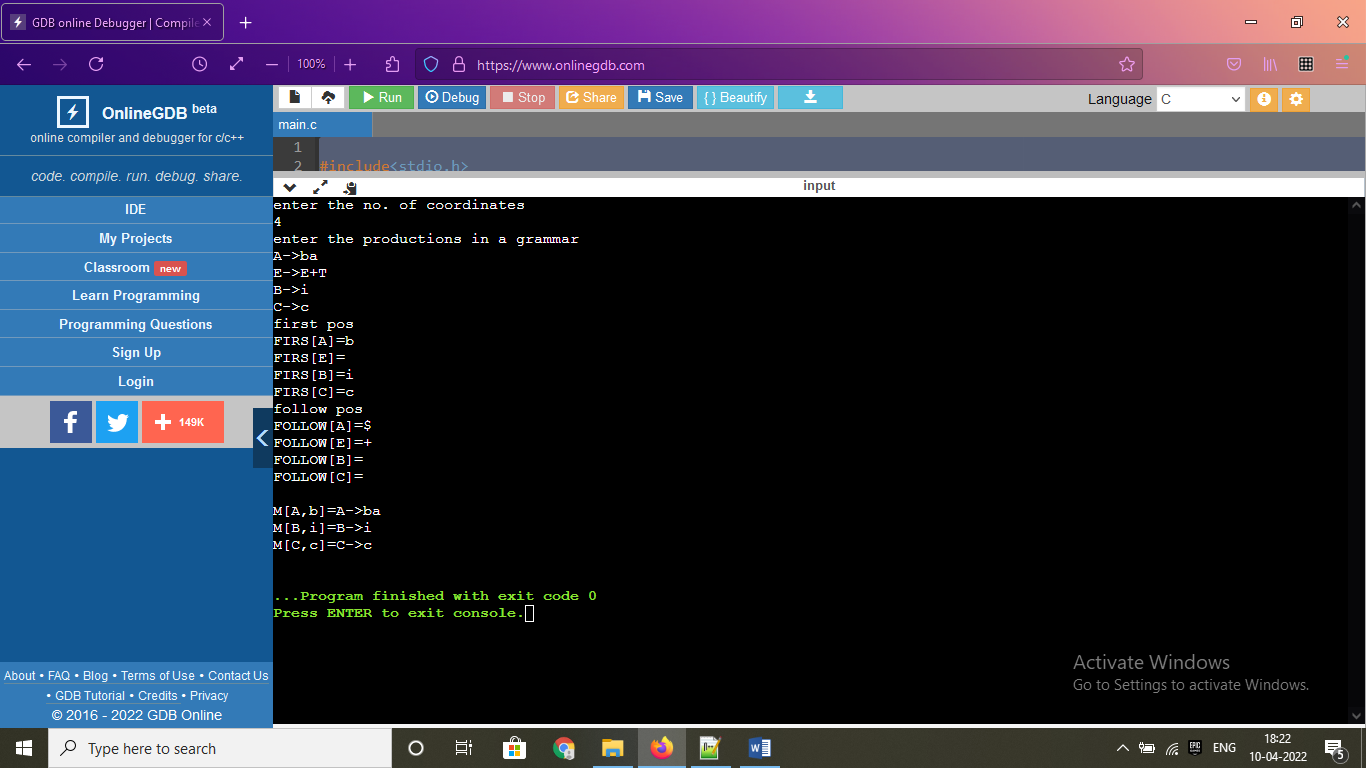
j++;

}

}

}

**Output:-**



**Practical-7-a**

**Aim: Program to convert Infix to Prefix**

#include <limits.h>

#include <stdio.h>

#include <stdlib.h>

#include <string.h>

#define MAX 100

int top = -1;

char stack[MAX];

// checking if stack is full

int isFull() {

return top == MAX - 1;

}

// checking is stack is empty

int isEmpty() {

return top == -1;

}

// Push function here, inserts value in stack and increments stack top by 1

void push(char item) {

if (isFull())

return;

top++;

stack[top] = item;

}

// Function to remove an item from stack. It decreases top by 1

int pop() {

if (isEmpty())

return INT\_MIN;

// decrements top and returns what has been popped

return stack[top--];

}

// Function to return the top from stack without removing it

int peek(){

if (isEmpty())

return INT\_MIN;

return stack[top];

}

// A utility function to check if the given character is operand

int checkIfOperand(char ch) {

return (ch >= 'a' && ch <= 'z') || (ch >= 'A' && ch <= 'Z');

}

// Fucntion to compare precedence

// If we return larger value means higher precedence

int precedence(char ch)

{

switch (ch)

{

case '+':

case '-':

return 1;

case '\*':

case '/':

return 2;

case '^':

return 3;

}

return -1;

}

// The driver function for infix to postfix conversion

int getPostfix(char\* expression)

{

int i, j;

for (i = 0, j = -1; expression[i]; ++i)

{

// Here we are checking is the character we scanned is operand or not

// and this adding to to output.

if (checkIfOperand(expression[i]))

expression[++j] = expression[i];

// Here, if we scan character ‘(‘, we need push it to the stack.

else if (expression[i] == '(')

push(expression[i]);

// Here, if we scan character is an ‘)’, we need to pop and print from the stack

// do this until an ‘(‘ is encountered in the stack.

else if (expression[i] == ')')

{

while (!isEmpty(stack) && peek(stack) != '(')

expression[++j] = pop(stack);

if (!isEmpty(stack) && peek(stack) != '(')

return -1; // invalid expression

else

pop(stack);

}

else // if an opertor

{

while (!isEmpty(stack) && precedence(expression[i]) <= precedence(peek(stack)))

expression[++j] = pop(stack);

push(expression[i]);

}

}

// Once all inital expression characters are traversed

// adding all left elements from stack to exp

while (!isEmpty(stack))

expression[++j] = pop(stack);

expression[++j] = '\0';

}

void reverse(char \*exp){

int size = strlen(exp);

int j = size, i=0;

char temp[size];

temp[j--]='\0';

while(exp[i]!='\0')

{

temp[j] = exp[i];

j--;

i++;

}

strcpy(exp,temp);

}

void brackets(char\* exp){

int i = 0;

while(exp[i]!='\0')

{

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

exp[i]=')';

else if(exp[i]==')')

exp[i]='(';

i++;

}

}

void InfixtoPrefix(char \*exp){

int size = strlen(exp);

// reverse string

reverse(exp);

//change brackets

brackets(exp);

getPostfix(exp);

reverse(exp);

}

int main()

{

char expression[50];

printf("Enter the Infix Expression : ");

scanf("%s",expression);

InfixtoPrefix(expression);

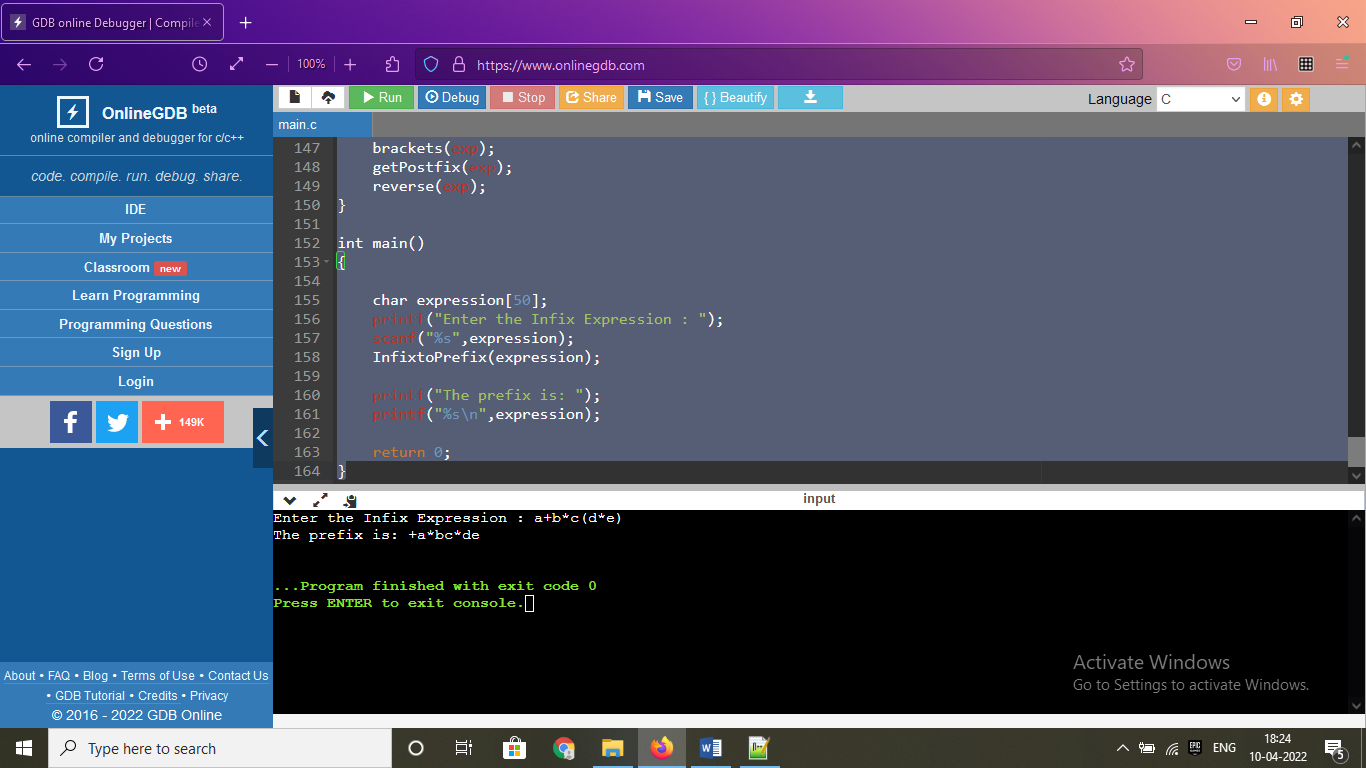
printf("The prefix is: ");

printf("%s\n",expression);

return 0;

}

**Output:**



**Practical-7-b**

**Aim: Program to convert Infix to Postfix**

#include <limits.h>

#include <stdio.h>

#include <stdlib.h>

#include <string.h>

#define MAX 100

int top = -1;

char stack[MAX];

int isFull()

{

return top == MAX - 1;

}

int isEmpty()

{

return top == -1;

}

void push(char item) {

if (isFull())

return;

top++;

stack[top] = item;

}

int pop()

{

if (isEmpty())

return INT\_MIN;

return stack[top--];

}

int peak()

{

if (isEmpty())

return INT\_MIN;

return stack[top];

}

int character(char ch)

{

return (ch >= 'a' && ch <= 'z') || (ch >= 'A' && ch <= 'Z');

}

int precedence(char ch)

{

switch (ch)

{

case '+':

case '-':

return 1;

case '\*':

case '/':

return 2;

case '^':

return 3;

}

return -1;

}

int convert(char\* expression)

{

int i, j;

for (i = 0, j = -1; expression[i]; ++i)

{

if (character(expression[i]))

expression[++j] = expression[i];

else if (expression[i] == '(')

push(expression[i]);

else if (expression[i] == ')')

{

while (!isEmpty(stack) && peak(stack) != '(')

expression[++j] = pop(stack);

if (!isEmpty(stack) && peak(stack) != '(')

return -1;

else

pop(stack);

}

else

{

while (!isEmpty(stack) && precedence(expression[i]) <= precedence(peak(stack)))

expression[++j] = pop(stack);

push(expression[i]);

}

}

while (!isEmpty(stack))

expression[++j] = pop(stack);

expression[++j] = '\0';

}

void parameters(char\* exp)

{

int i = 0;

while(exp[i]!='\0')

{

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

exp[i]=')';

else if(exp[i]==')')

exp[i]='(';

i++;

}

}

void post(char \*exp)

{

int size = strlen(exp);

parameters(exp);

convert(exp);

}

int main()

{

char expression[50];

printf("Enter the Infix Expression : ");

scanf("%s",expression);

post(expression);

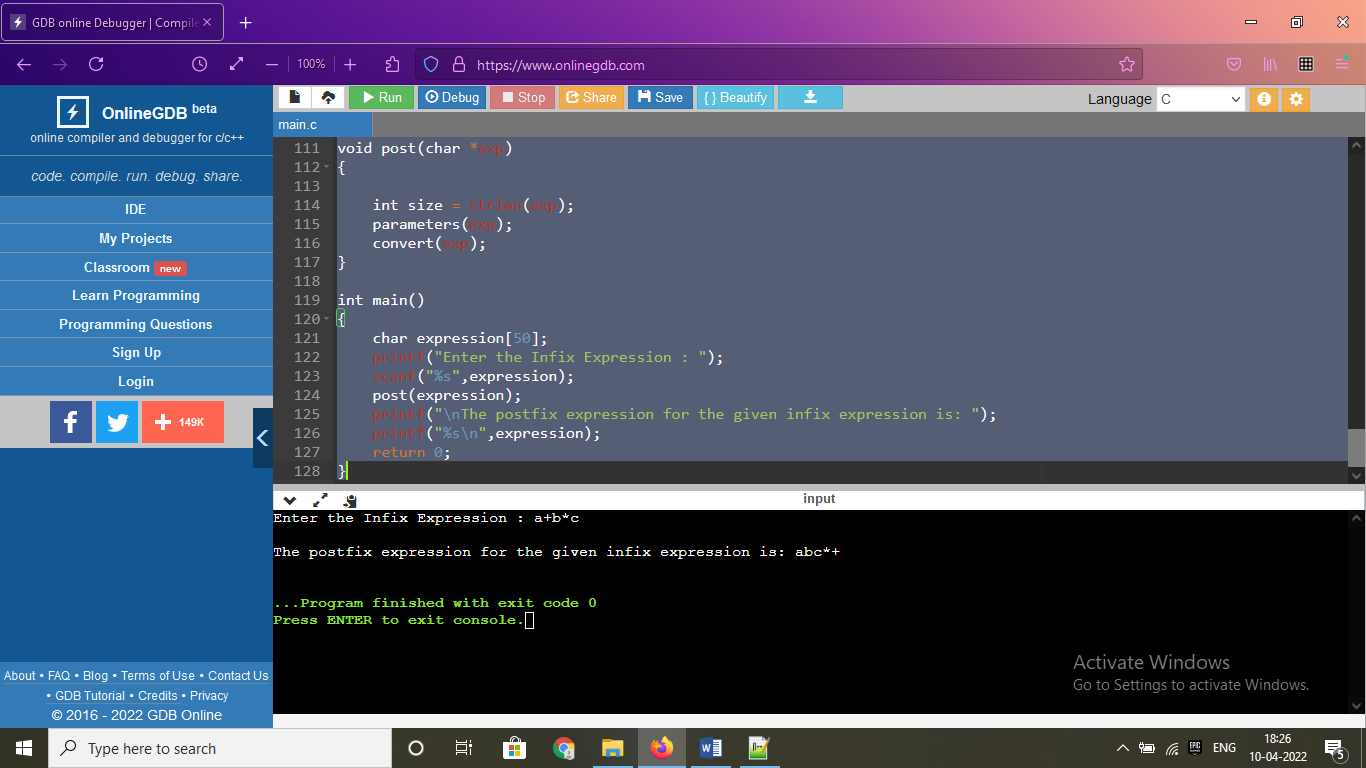
printf("\nThe postfix expression for the given infix expression is: ");

printf("%s\n",expression);

return 0;

}

**Output:**



**Practical 8**

**Aim:** Program to design LALR bottom up parser

#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;

clrscr();

printf(" Enter the input :");

scanf("%s",inp);

len=strlen(inp);

inp[len]='$';

inp[len+1]='\0';

push(stack,&top,bl);

printf("\n stack \t\t\t input");

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");

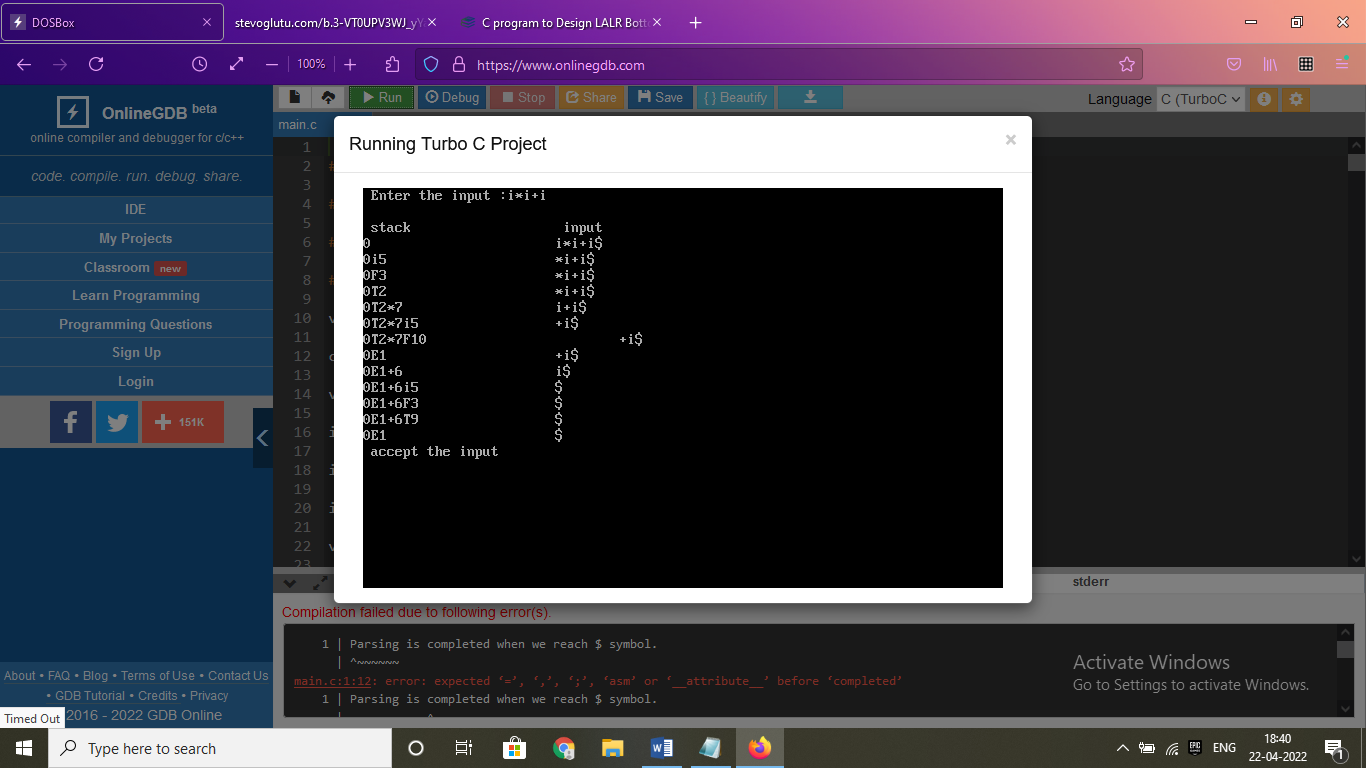
break;

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

break;

}

}



**Practical-9**

**Aim:** Program to implement Operator precedence parsing

#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 :\n");

scanf("%d", & n);

printf("\nEnter the terminals :\n");

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("\n\*\*\*\* OPERATOR 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\n");

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]);

}

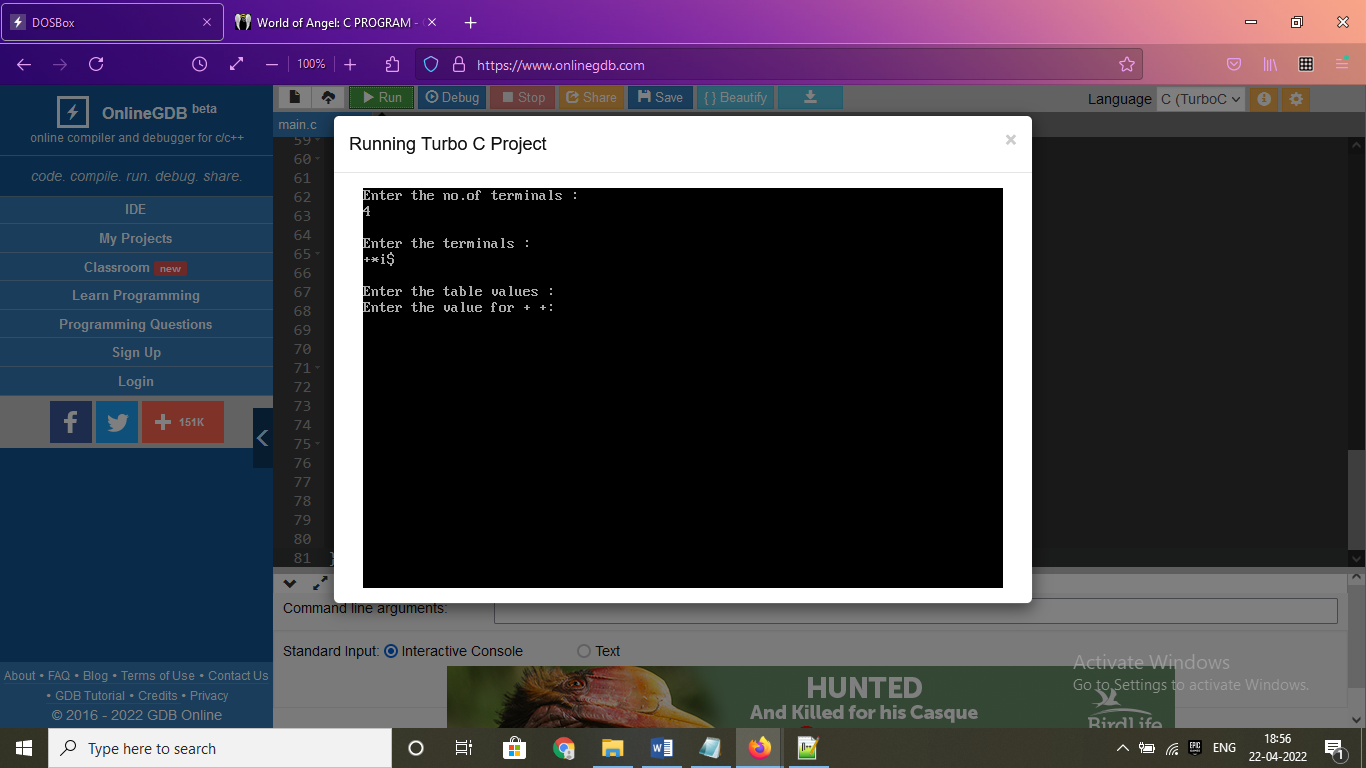
printf("\t\t\t");

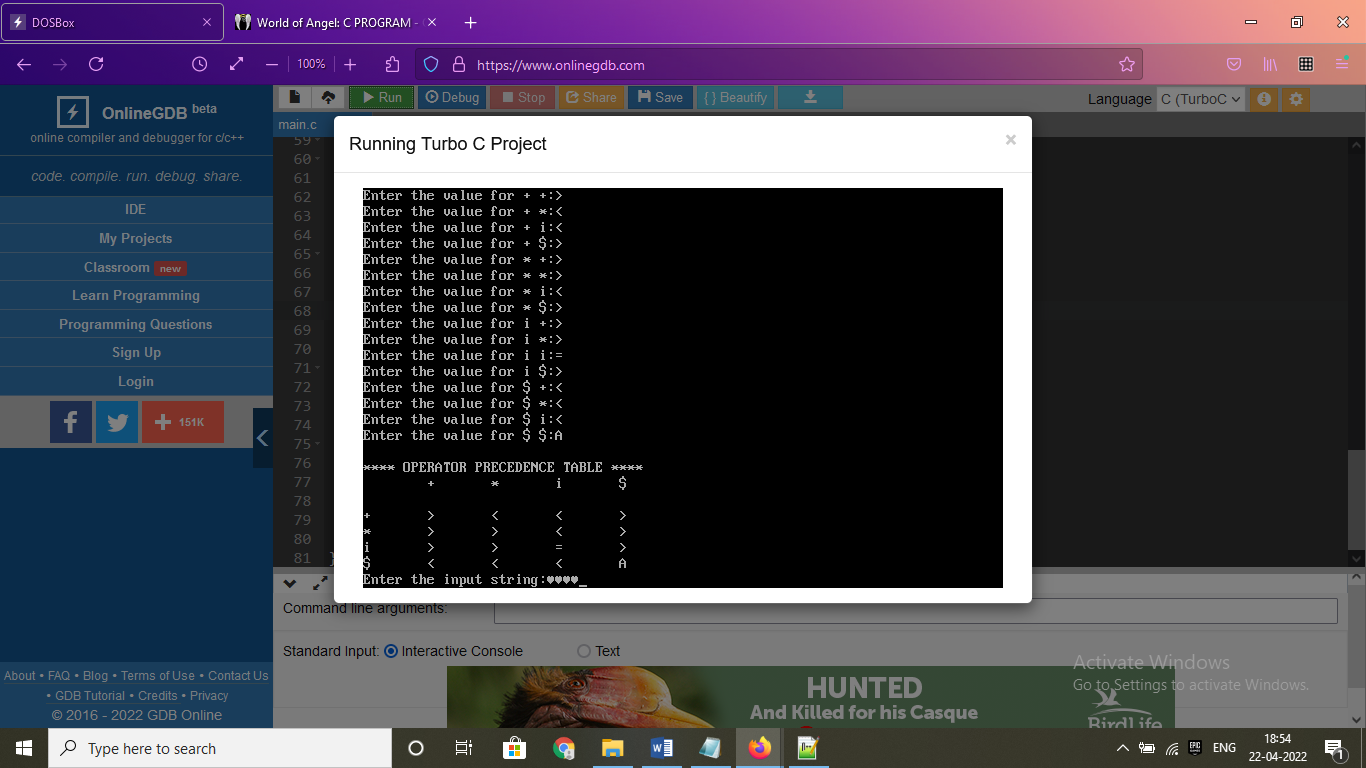
}

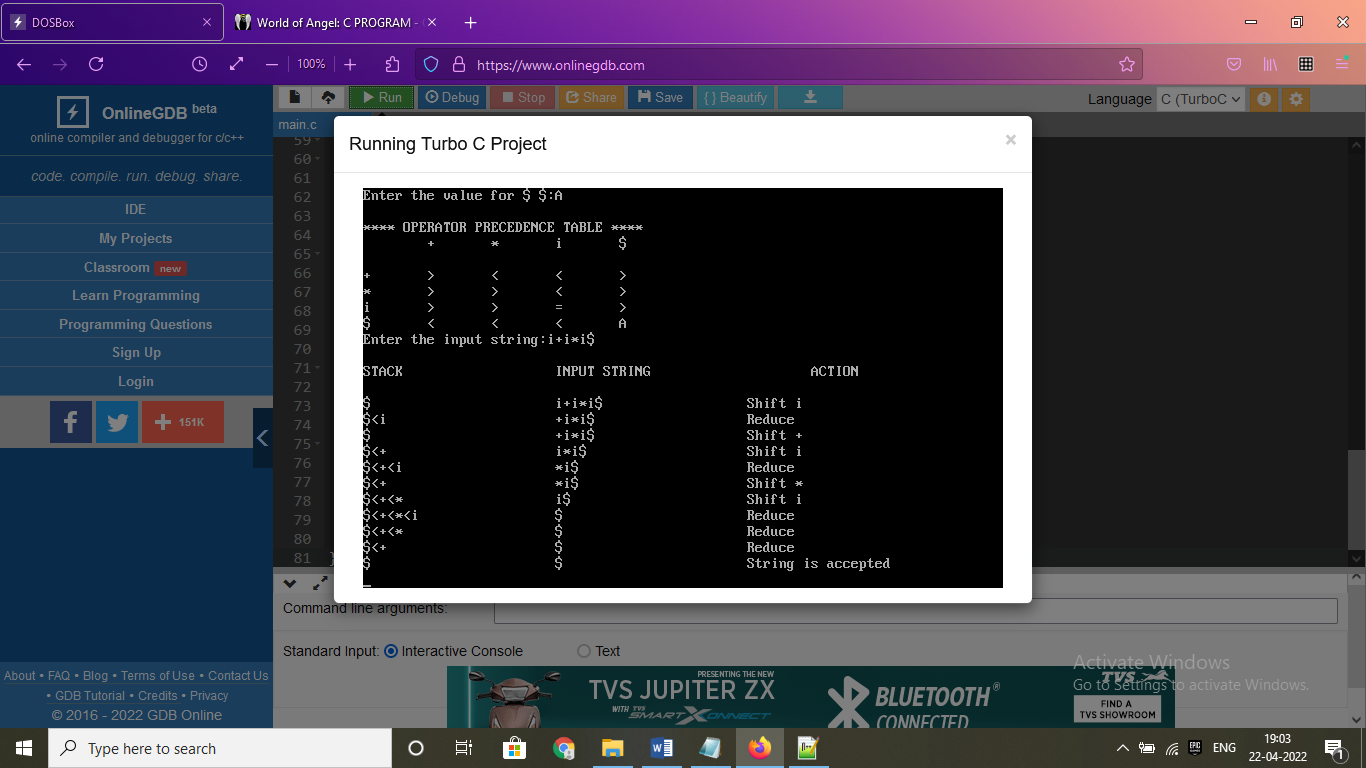
getch();

}

**Output:**







**Practical-10**

**Aim:** Program to implement three address 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()

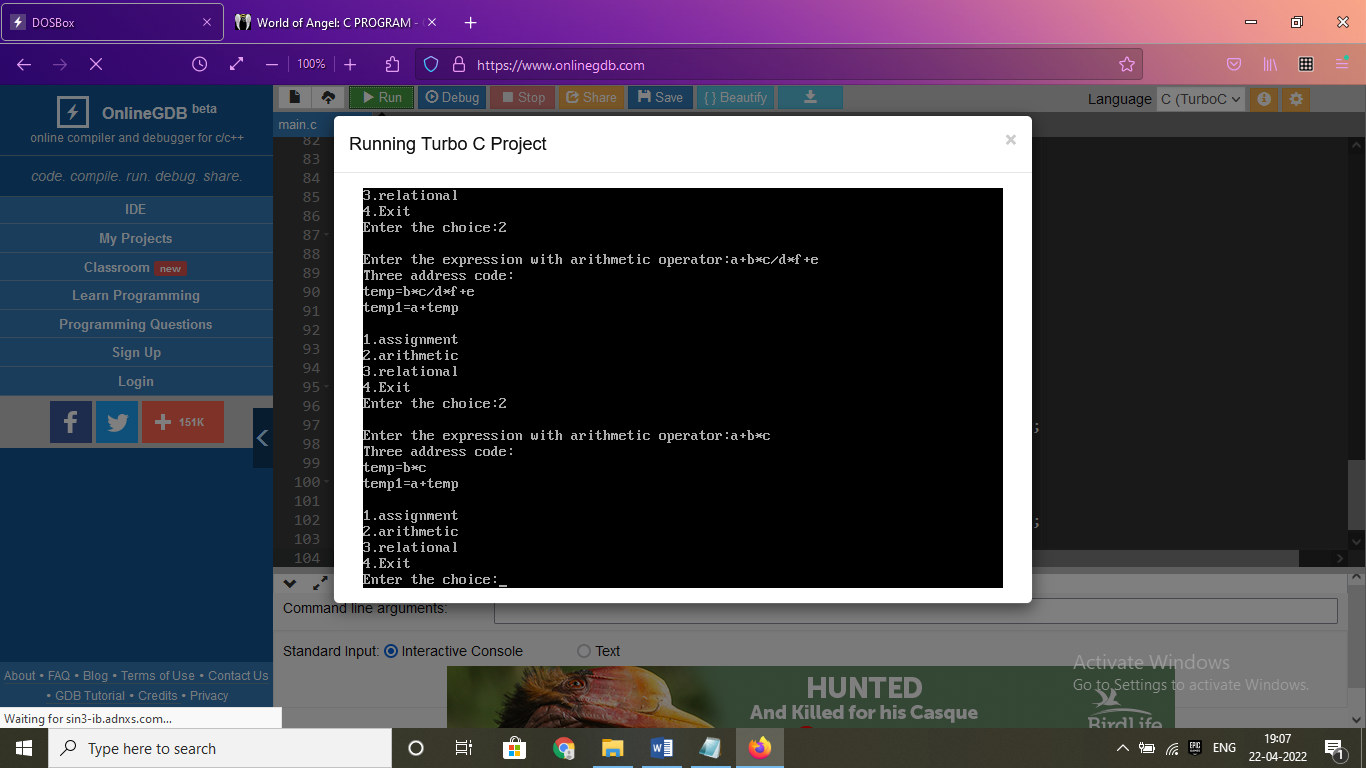
{

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-10**

**Aim:** Program to implement code optimization

#include<stdio.h>

#include<conio.h>

#include<string.h>

struct op

{ char l;

char r[20];

}

op[10],pr[10];

void main()

{ int a,i,k,j,n,z=0,m,q;

char \*p,\*l;

char temp,t;

char \*tem;

clrscr();

printf("Enter the Number of Values:");

scanf("%d",&n);

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

{

printf("left: ");

op[i].l=getche();

printf("\tright: ");

scanf("%s",op[i].r);

}

printf("Intermediate Code\n") ;

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

{

printf("%c=",op[i].l);

printf("%s\n",op[i].r);

}

for(i=0;i<n-1;i++)

{ temp=op[i].l;

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

{

p=strchr(op[j].r,temp);

if(p)

{

pr[z].l=op[i].l;

strcpy(pr[z].r,op[i].r);

z++; }}}

pr[z].l=op[n-1].l;

strcpy(pr[z].r,op[n-1].r);

z++;

printf("nAfter Dead Code Eliminationn");

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

printf("%ct=",pr[k].l);

printf("%sn",pr[k].r);

}

for(m=0;m<z;m++) {

tem=pr[m].r;

for(j=m+1;j<z;j++) {

p=strstr(tem,pr[j].r);

if(p) {

t=pr[j].l;

pr[j].l=pr[m].l;

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

l=strchr(pr[i].r,t) ;

if(l) {

a=l-pr[i].r;

printf("pos: %d",a);

pr[i].r[a]=pr[m].l; }}}}}

printf("\n Eliminate Common Expression\n");

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

{

printf("%c\t=",pr[i].l);

printf("%s\n",pr[i].r);

}

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

{

for(j=i+1;j<z;j++)

{

q=strcmp(pr[i].r,pr[j].r);

if((pr[i].l==pr[j].l)&&!q)

{

pr[i].l='\0';

strcpy(pr[i].r,'\0');

}}}

printf("Optimized Code\n");

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

{ if(pr[i].l!='\0')

{

printf("%c=",pr[i].l);

printf("%s\n",pr[i].r);

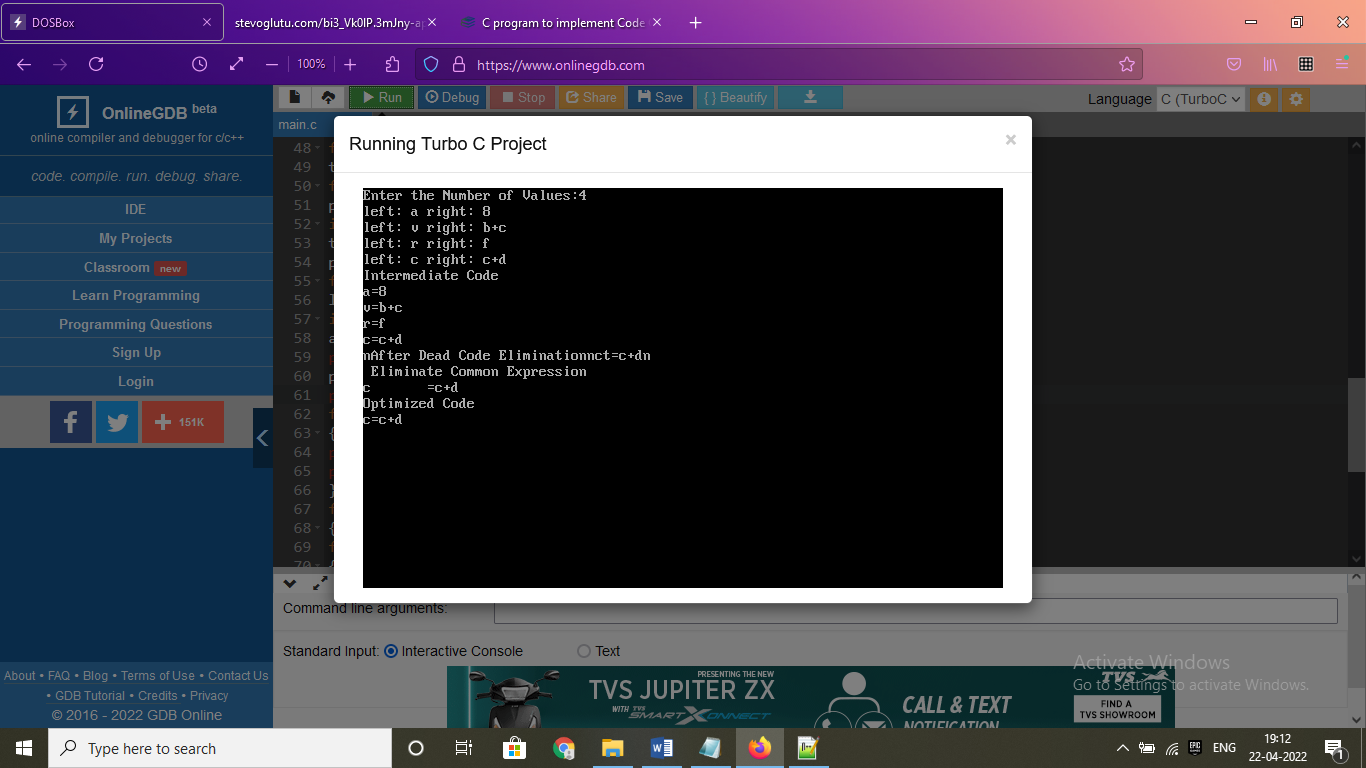
}

}

getch();

}

**Output:**



**Practical-10**

**Aim:** Program to implement heap storage stratergy

#include<stdio.h>

#include<conio.h>

#include<stdlib.h>

#define TRUE 1

#define FALSE 0

typedef struct Heap

{

int data;

struct Heap \*next;

} node;

node \*create ();

void

main ()

{

int choice, val;

char ans;

node \*head;

void display (node \*);

node \*search (node \*, int);

node \*insert (node \*);

void dele (node \*\*);

head = NULL;

do

{

clrscr ();

printf ("\n1.Create");

printf ("\n2.Display");

printf ("\n3.Insert an element in a list");

printf ("\n4.Delete an element from list");

printf ("\n5.Quit");

printf ("\n Enter Your Choice(1-5) :");

scanf ("%d", &choice);

switch (choice)

{

case 1:

head = create ();

break;

case 2:

display (head);

break;

case 3:

head = insert (head);

break;

case 4:

dele (&head);

break;

case 5:

exit (0);

default:

clrscr ();

printf ("\nInvalid Choice,Try again");

getch ();

}

}

while (choice != 5);

}

node \*

create ()

{

node \*temp, \*new1, \*head;

int val, flag;

char ans = 'y';

node \*get\_node ();

temp = NULL;

flag = TRUE;

do

{

printf ("\n Enter the Element");

scanf ("%d", &val);

new1 = get\_node ();

if (new1 == NULL)

printf ("\n Memory is not allocated");

new1->data = val;

if (flag == TRUE)

{

head = new1;

temp = head;

flag = FALSE;

}

else

{

temp->next = new1;

temp = new1;

}

printf ("\nDo you want to enter more elements?(y/n) : ");

ans = getch ();

}

while (ans == 'y');

printf ("\nThe list is created");

getch ();

clrscr ();

return head;

}

node \*

get\_node ()

{

node \*temp;

temp = (node \*) malloc (sizeof (node));

temp->next = NULL;

return temp;

}

void

display (node \* head)

{

node \*temp;

temp = head;

if (temp == NULL)

{

printf ("\n The list is empty\n");

getch ();

clrscr ();

return;

}

while (temp != NULL)

{

printf ("%d->", temp->data);

temp = temp->next;

}

printf ("NULL");

getch ();

clrscr ();

}

node \*

search (node \* head, int key)

{

node \*temp;

int found;

temp = head;

if (temp == NULL)

{

printf ("The linked list is empty\n");

getch ();

clrscr ();

return NULL;

}

found = FALSE;

while ((temp != NULL) && (found == FALSE))

{

if (temp->data != key)

temp = temp->next;

else

found = TRUE;

}

if (found == TRUE)

{

printf ("\n The Elements is present in the list\n");

getch ();

return temp;

}

else

printf ("\n The Element is not present in the list\n");

getch ();

return NULL;

}

node \*

insert (node \* head)

{

int choice;

node \*insert\_head (node \*);

void insert\_after (node \*);

void insert\_last (node \*);

printf ("\nInsert a node as a head node");

printf ("\nInsert a node as a last node");

printf ("\nInsert a node as at the intermediate position in the list ");

printf ("\nEnter your choice for insertion of node ");

scanf ("%d", &choice);

switch (choice)

{

case 1:

head = insert\_head (head);

break;

case 2:

insert\_last (head);

break;

case 3:

insert\_after (head);

break;

}

return head;

}

node \*

insert\_head (node \* head)

{

node \*New, \*temp;

New = get\_node ();

printf ("\n Enter the element which you want to insert ");

scanf ("%d", &New->data);

if (head == NULL)

head = New;

else

{

temp = head;

New->next = temp;

head = New;

}

return head;

}

void

insert\_last (node \* head)

{

node \*New, \*temp;

New = get\_node ();

printf ("\n Enter the element which you want to insert ");

scanf ("%d", &New->data);

if (head == NULL)

{

head = New;

}

else

{

temp = head;

while (temp->next != NULL)

temp = temp->next;

temp->next = New;

New->next = NULL;

}

}

void

insert\_after (node \* head)

{

int key;

node \*New, \*temp;

New = get\_node ();

printf ("Enter the element after which you want to insert ");

scanf ("%d", &key);

temp = head;

do

{

if (temp->data == key)

{

printf ("Enter element which you want to insert ");

scanf ("%d", &New->data);

New->next = temp->next;

temp->next = New;

return;

}

else

temp = temp->next;

}

while (temp != NULL);

}

node \*

get\_prev (node \* head, int val)

{

node \*temp, \*prev;

int flag;

temp = head;

if (temp == NULL)

return NULL;

flag = FALSE;

prev = NULL;

while (temp != NULL && !flag)

{

if (temp->data != val)

{

prev = temp;

temp = temp->next;

}

else

flag = TRUE;

}

if (flag)

return prev;

else

return NULL;

}

void

dele (node \*\* head)

{

int key;

node \*New, \*temp, \*prev;

temp = \*head;

if (temp == NULL)

{

printf ("\n The list is empty\n ");

getch ();

clrscr ();

return;

}

clrscr ();

printf ("\nENTER the Element you want to delete:");

scanf ("%d", &key);

temp = search (\*head, key);

if (temp != NULL)

{

prev = get\_prev (\*head, key);

if (prev != NULL)

{

prev->next = temp->next;

free (temp);

}

else

{

\*head = temp->next;

free (temp);

}

printf ("\nThe Element is deleted\n");

getch ();

clrscr ();

}

}

