**EXPERIMENT-1**

**AIM:** *Write a program to check whether the entered string is valid identifier or not*

**CODE**

#include <bits/stdc++.h>

using namespace std;

bool check(string str, int n)

{

if(!((str[0] >= 'a' && str[0] <='z') || (str[0] >= 'A' && str[0] <= 'Z') || str[0] == '\_'))

return false;

for(int i=1; i<str.length(); i++)

{

if(!((str[i] >= 'a' && str[i] <='z') || (str[i] >= 'A' && str[i] <= 'Z') || (str[i] >= '0' && str[i] <= '9') || str[i] == '\_' ))

return false;

}

if((str == "if") || (str == "else") || (str == "while") || (str == "do") ||

(str == "break") || (str == "volatile") || (str == "continue") || (str == "int") ||

(str == "double") || (str == "float") || (str == "return") || (str == "char") ||

(str == "case") || (str == "char") || (str == "sizeof") || (str == "long") ||

(str == "short") || (str == "typedef") || (str == "switch") || (str == "unsigned") ||

(str == "void") || (str == "static") || (str == "struct") || (str == "goto") ||

(str == "auto") || (str == "register") || (str == "enumf") || (str == "union") ||

(str == "const") || (str == "for") || (str == "short") || (str == "signed"))

return false;

return true;

}

int main()

{

string str;

cout<<"Enter a string: "<<endl;

cin>>str;

int n = str.length();

if(check(str,n))

cout<<"Valid";

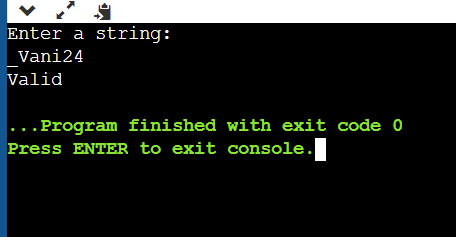
else

cout<<"Invalid";

return 0;

}

**OUTPUT**



**EXPERIMENT-2**

**AIM:** a. *Write a program to remove the comments from a given segment of code.*

*b. Write a program to remove the blank spaces from a given segment of code.*

**CODE**

1. *Removing Comments*

#include <iostream>

using namespace std;

string removeComments(string str)

{

int n = str.length();

string res;

bool sc = false;

bool mc = false;

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

{

if (sc == true && str[i] == '\n')

sc = false;

else if (mc == true && str[i] == '\*' && str[i+1] == '/')

mc = false, i++;

else if (sc || mc)

continue;

else if (str[i] == '/' && str[i+1] == '/')

sc = true, i++;

else if (str[i] == '/' && str[i+1] == '\*')

mc = true, i++;

else res += str[i];

}

return res;

}

int main()

{

string str;

cout<<"Enter string: "<<endl;

cin>>str;

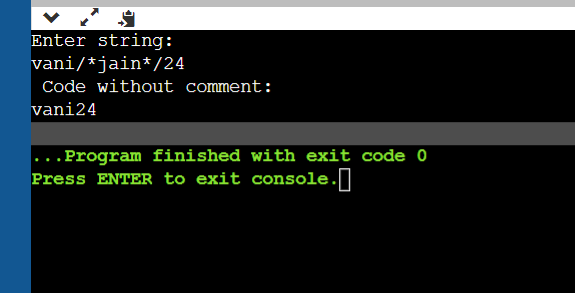
cout << " Code without comment: "<<endl;

cout << removeComments(str);

return 0;

}

**OUTPUT**

****

1. *Blankspaces*

#include <iostream>

using namespace std;

void removeSpaces(char \*str)

{

int count = 0;

for (int i = 0; str[i]; i++)

if (str[i] != ' ')

str[count++] = str[i];

str[count] = '\0';

}

int main()

{

char str[] = "H ey Go od Mo rni ng ";

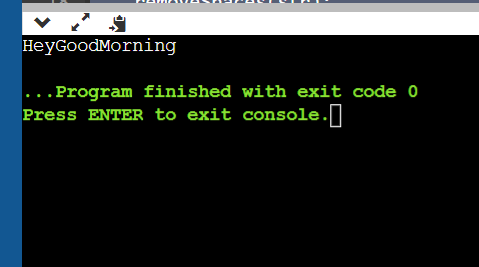
removeSpaces(str);

cout << str;

return 0;

}

**OUTPUT**



**EXPERIMENT-3**

**AIM:** Write a program to calculate the first of a given grammar.

**CODE**

#include<stdio.h>

#include<ctype.h>

void FIRST(char );

int count,n=0;

char prodn[10][10], first[10];

int 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];

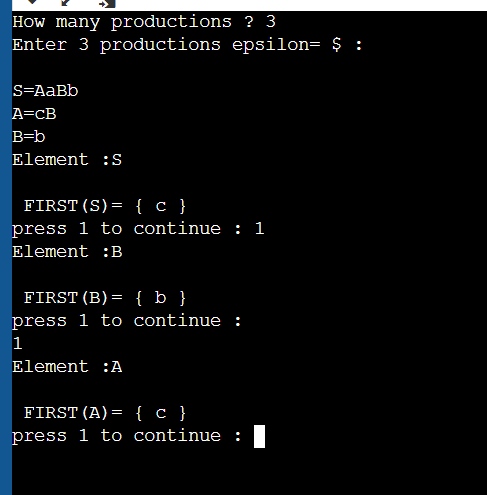
else FIRST(prodn[j][2]);

}

}

}

**OUTPUT**

****

**EXPERIMENT-4**

**AIM:** *Write a program to* calculate the follow of a given grammar.

**CODE**

#include<stdio.h>

#include<ctype.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];

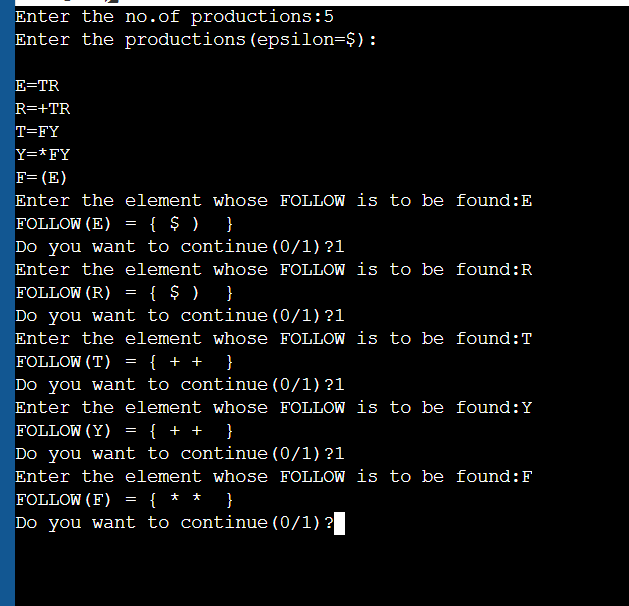
else first(a[k][2]);

}

}

}

**OUTPUT**

****

**EXPERIMENT-5**

**AIM:** *Write a program to check if there exist left recursion if exist then remove the left recursion using the studied rules.*

**CODE**

#include<iostream>

using namespace std;

int main () {

char terminal;

char b,a;

int n;

bool flag=false;

char production[10][10];

int index=3;

cout<<"Enter Number of Production : "<<endl;

cin>>n;

cout<<"Enter "<<n<<" productions"<<endl;

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

cin>>production[i];

}

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

terminal=production[i][0];

if(terminal==production[i][index]) {

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

cout<<"Grammar is left recursive"<<endl;

flag=true;

while(production[i][index]!='\0' && production[i][index]!='|'){

index++;

}

if(production[i][index]!='\0'){

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

cout<<"Grammar without left recursion"<<endl;

cout<<terminal<<"->"<<b<<terminal<<"'"<<endl;

cout<<terminal<<"'->"<<a<<terminal<<"'|e"<<endl;

}

}

index=3;

}

if(flag==false){

cout<<"Grammar is not left recursive"<<endl;

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

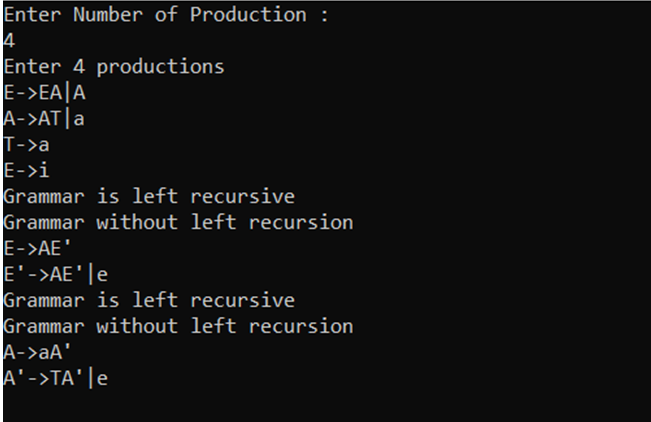
cout<<production[i]<<endl;

}

}

}

**OUTPUT**

****

**EXPERIMENT-6**

**AIM:** *Write a program to check if there exist left factoring if exist then remove the left factoring using the studied rules.*

**CODE**

#include<stdio.h>

#include<string.h>

int main()

{

char production[20],a1[20],a2[20],m[20],n[20],t[20];

int i,j=0,k=0,l=0,pos;

int flag=0;

printf("Enter Production : A->");

gets(production);

for(i=0;production[i]!='|';i++,j++)

a1[j]=production[i];

a1[j]='\0';

for(j=++i,i=0;production[j]!='\0';j++,i++)

a2[i]=production[j];

a2[i]='\0';

for(i=0;i<strlen(a1)||i<strlen(a2);i++)

{

if(a1[i]==a2[i])

{

flag=1;

m[k]=a1[i];

k++;

pos=i+1;

}

}

for(i=pos,j=0;a1[i]!='\0';i++,j++){

n[j]=a1[i];

}

n[j++]='|';

for(i=pos;a2[i]!='\0';i++,j++){

n[j]=a2[i];

}

m[k]='X';

m[++k]='\0';

n[j]='\0';

if(flag==0){

printf("Grammar is not left factoring\n");

}

else{

printf("Grammar is left factoring\n");

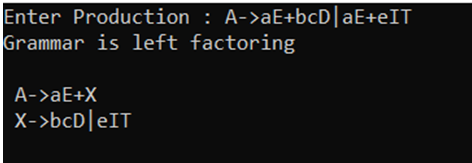
printf("\n A->%s",m);

printf("\n X->%s\n",n);

}

}

**OUTPUT**



**EXPERIMENT-7**

**AIM:** *Write a program to implement recursive descent parsing.*

**CODE**

#include<stdio.h>

#include<conio.h>

#include<string.h>

char input[100];

int i,l;

int E();

int F();

int EP();

int T();

int TP();

int 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();

}

int E(){

if(T()){

if(EP())

return(1);

else

return(0);

}

else

return(0);

}

int EP(){

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

i++;

if(T()){

if(EP())

return(1);

else

return(0);

}

else

return(0);

}

else

return(1);

}

int T(){

if(F()){

if(TP())

return(1);

else

return(0);

}

else

return(0);

}

int TP(){

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

i++;

if(F()){

if(TP())

return(1);

else

return(0);

}

else

return(0);

}

else

return(1);

}

int 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);

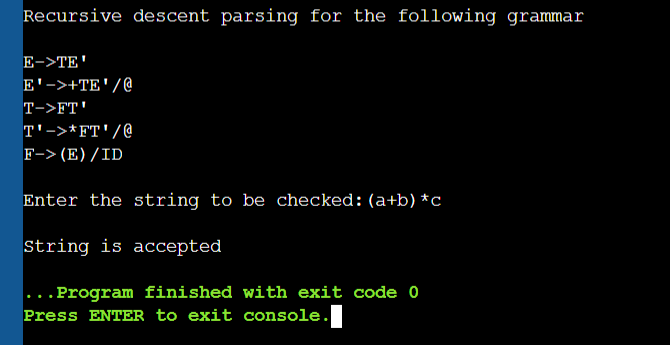
}

else

return(0);

}

**OUTPUT**



**EXPERIMENT-8**

**AIM:** *Write a* program to perform implement predictive parsing using LL(1) parser.

**CODE**

#include<stdio.h>

#include<string.h>

char prol[7][10]={"S","A","A","B","B","C","C"};

char pror[7][10]={"A","Bb","Cd","aB","@","Cc","@"};

char prod[7][10]={"S->A","A->Bb","A->Cd","B->aB","B->@","C->Cc","C->@"};

char first[7][10]={"abcd","ab","cd","a@","@","c@","@"};

char follow[7][10]={"$","$","$","a$","b$","c$","d$"};

char table[5][6][10];

numr(char c){

switch(c)

{

case 'S': return 0;

case 'A': return 1;

case 'B': return 2;

case 'C': return 3;

case 'a': return 0;

case 'b': return 1;

case 'c': return 2;

case 'd': return 3;

case '$': return 4;

} return(2);

} int main(){

int i,j,k;

for(i=0;i<5;i++)

for(j=0;j<6;j++)

strcpy(table[i][j]," ");

printf("\nThe following is the predictive parsing table for the following grammar:\n");

for(i=0;i<7;i++)

printf("%s\n",prod[i]);

printf("\nPredictive parsing table is\n");

fflush(stdin);

for(i=0;i<7;i++){

k=strlen(first[i]);

for(j=0;j<10;j++)

if(first[i][j]!='@')

strcpy(table[numr(prol[i][0])+1][numr(first[i][j])+1],prod[i]);

}

for(i=0;i<7;i++){

if(strlen(pror[i])==1){

if(pror[i][0]=='@')

{

k=strlen(follow[i]);

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

strcpy(table[numr(prol[i][0])+1][numr(follow[i][j])+1],prod[i]);

}}}

strcpy(table[0][0]," ");

strcpy(table[0][1],"a");

strcpy(table[0][2],"b");

strcpy(table[0][3],"c");

strcpy(table[0][4],"d");

strcpy(table[0][5],"$");

strcpy(table[1][0],"S");

strcpy(table[2][0],"A");

strcpy(table[3][0],"B");

strcpy(table[4][0],"C");

printf("\n--------------------------------------------------------\n");

for(i=0;i<5;i++)

for(j=0;j<6;j++){

printf("%-10s",table[i][j]);

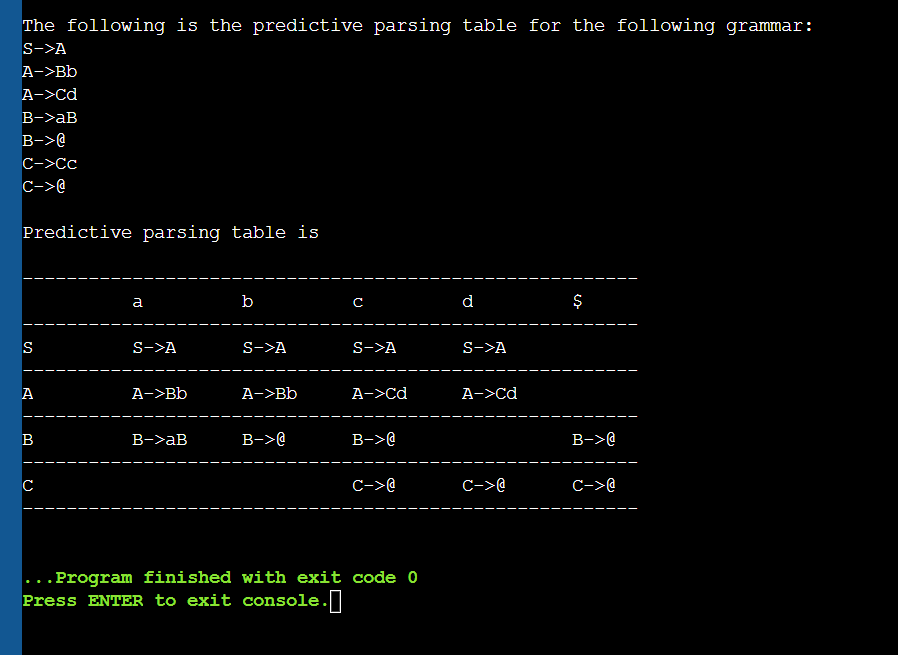
if(j==5)

printf("\n--------------------------------------------------------\n");

} return 0;

}

**OUTPUT**

****

**EXPERIMENT-9**

**AIM:** *Write a program to implement operator precedence parsing.*

**CODE**

#include <stdio.h>

int main()

{

char stack[20],ip[20],opt[10][10][1],ter[10];

int i,j,k,n,top=0,col,row;

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 number 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");

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]);

}

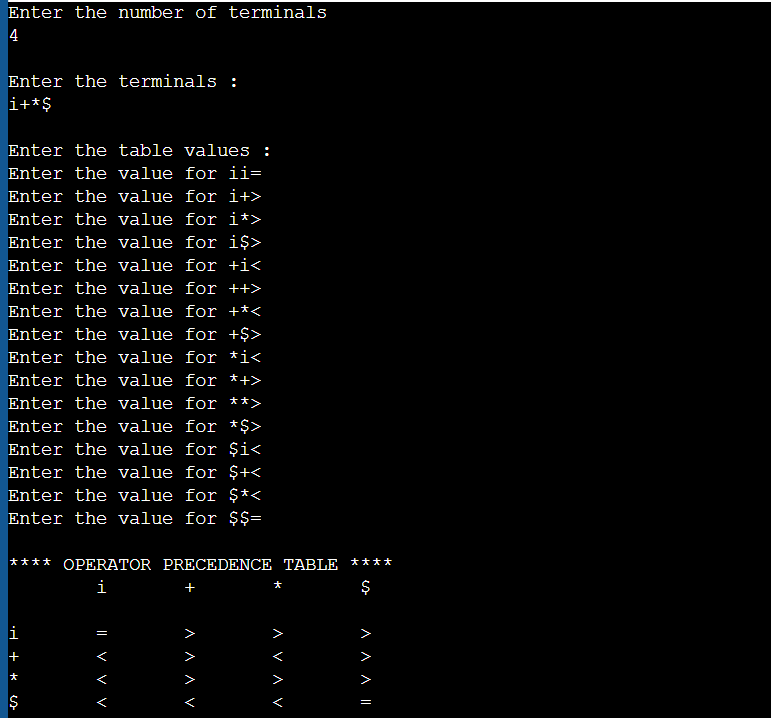
printf("\t\t\t");

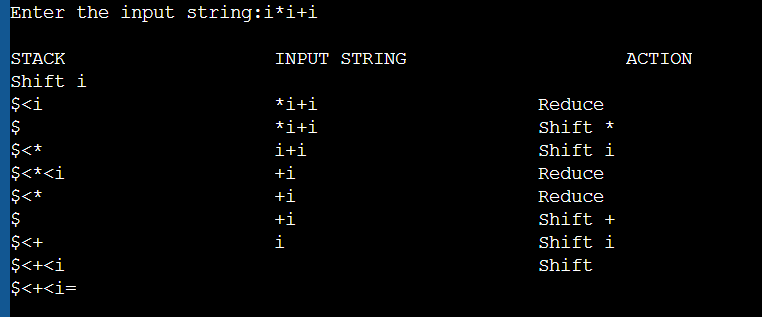
}

return 0;

}

**OUTPUT**





**EXPERIMENT-10**

**AIM:** *Write a program to generate Three address code.*

**CODE**

#include <stdio.h>

#include <string.h>

#include <stdlib.h>

char \*strrev(char \*str)

{

if (!str||!\*str)

return str;

int i=strlen(str)-1,j=0;

char ch;

while(i>j)

{

ch=str[i];

str[i]=str[j];

str[j]=ch;

i--;

j++;

}

return str;

}

void pm();

void plus();

void div1();

int i,ch,j,l,addr=100;

char ex[10],expr[10],expr1[10],expr2[10],id1[5],id2[5],op[5];

int main()

{

while(1){

printf("\nSelect the type of expression \n 1.Assignment\n2.Arithmetic\n3.Relational\n4.Exit\n\nEnter the choice");

scanf("%d",&ch);

switch(ch){

case 1:

printf("\nEnter the expression with assignment operator");

scanf("%s",expr);

l=strlen(expr);

expr2[0]='\0';

i=0;

while(expr[i]!='='){

i++;

}

strncat(expr2,expr,i);

strrev(expr);

expr1[0]='\0';

strncat(expr1,expr,l-(i+1));

strrev(expr1);

printf("Three address code:\ntemp=%s\n%s=temp\n",expr1,expr2);

case 2:

printf("\nEnter the expression with arithmetic operator:");

scanf("%s",ex);

strcpy(expr,ex);

l=strlen(expr);

expr1[0]='\0';

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

if(expr[i]=='+'||expr[i]=='-'){

if(expr[i+2]=='/'||expr[i+2]=='\*'){

pm();

break;

}

else{

plus();

break;

}

}

else if(expr[i]=='/'||expr[i]=='\*'){

div1();

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);

}

}

return 0;

}

void pm()

{

strrev(expr);

j=l-i-1;

strncat(expr1,expr,j);

strrev(expr1);

printf("Three address code:\ntemp=%s\ntemp1=%c%ctemp\n",expr1,expr[j+1],expr[j]);

}

void div1()

{

strncat(expr1,expr,i+2);

printf("Three address code:\ntemp=%s\ntemp1=temp%c%c\n",expr1,expr[i+2],expr[i+3]);

}

void plus()

{

strncat(expr1,expr,i+2);

printf("Three address code:\ntemp=%s\ntemp1=temp%c%c\n",expr1,expr[i+2],expr[i+3]);

}

**OUTPUT**

