**Compiler Desig lab manual**

**EX.NO.1  Implementation of Lexical Analyzer**

**AIM:**

To Write a C program to implementation of Lexical Analyzer.

**ALGORITHM:**

Step 1 : Start the program

Step 2 : Include necessary header files.

Step 3: The ctype header file is to load the file with predicate is digit.

Step 4 : The define directive defines the buffer size, numerics, assignment operator, relational operator.

Step 5 : Initialize the necessary variables.

Step 6: To return index of new string S, token t using insert() function.

Step 7 : Initialize the length of every string.

Step 8: Check the necessary condition.

Step 9: Call the initialize() function. This function loads the keywords into the symbol table.

Step 10 : Check the conditions such as white spaces, digits, letters and alphanumerics.

Step 11 : To return index of entry for string S, or 0 if S is not found using lookup( ) function.

Step 12 : Check this until EOF is found.

Step 13 : Otherwise initialize the token value to be none.

Step 14 : In the main function if lookahead equals numeric then the value of attribute num is given by the global variable tokenval.

Step 15 : Check the necessary conditions such as arithmetic operators , parenthesis , identifiers, assignment operators and relational operators.

Step 16 : Stop the program

**PROGRAM:**

#include<stdio.h>

#include<ctype.h>

#include<string.h>

int main()

{

FILE \*input, \*output;

int l=1;

int t=0;

int j=0;

int i,flag;

char ch,str[20];

input = fopen("input.txt","r");

output = fopen("output.txt","w");

char keyword[30][30] = {"int","main","if","else","do","while"};

fprintf(output,"Line no. \t Token no. \t Token \t Lexeme\n\n");

while(!feof(input))

{

i=0;

flag=0;

ch=fgetc(input);

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

{

fprintf(output,"%7d\t\t %7d\t\t Operator\t %7c\n",l,t,ch);

t++;

}

else if( ch==';' || ch=='{' || ch=='}' || ch=='(' || ch==')' || ch=='?' ||

ch=='@' || ch=='!' ||

ch=='%')

{

fprintf(output,"%7d\t\t %7d\t\t Special symbol\t %7c\n",l,t,ch);

t++;

}

else if(isdigit(ch))

{

fprintf(output,"%7d\t\t %7d\t\t Digit\t\t %7c\n",l,t,ch);

t++;

}

else if(isalpha(ch))

{

str[i]=ch;

i++;

ch=fgetc(input);

while(isalnum(ch) && ch!=' ')

{

str[i]=ch;

i++;

ch=fgetc(input);

}

str[i]='\0';

for(j=0;j<=30;j++)

{

if(strcmp(str,keyword[j])==0)

{

flag=1;

break;

}

}

if(flag==1)

{

fprintf(output,"%7d\t\t %7d\t\t Keyword\t %7s\n",l,t,str);

t++;

}

else

{

fprintf(output,"%7d\t\t %7d\t\t Identifier\t %7s\n",l,t,str);

t++;

}

}

else if(ch=='\n')

{

l++;

}

}

fclose(input);

fclose(output);

return 0;

}

Input:

//input.txt

#include<stdio.h>

void main()

{

printf("Hello World");

}

Output:

//output.txt

Line no. Token no. Token Lexeme

1 0 Identifier include

1 1 Identifier stdio

1 2 Identifier h

2 3 Identifier void

2 4 Keyword main

2 5 Special symbol )

3 6 Special symbol {

4 7 Identifier printf

4 8 Identifier Hello

4 9 Identifier World

4 10 Special symbol )

4 11 Special symbol ;

5 12 Special symbol }

1. **CONVERSION OF REGULAR EXPRESSION TO NFA**

**AIM:**

To write a C program to convert the regular expression to NFA.

**ALGORITHM:**

1. Start the program.
2. Declare all necessary header files.
3. Define the main function.
4. Declare the variables and initialize variables r & c to ‘0’.
5. Use a for loop within another for loop to initialize the matrix for NFA states.
6. Get a regular expression from the user & store it in ‘m’.
7. Obtain the length of the expression using strlen() function and store it in ‘n’.
8. Use for loop upto the string length and follow steps 8 to 12.
9. Use switch case to check each character of the expression

10.If case is ‘\*’, set the links as ‘E’ or suitable inputs as per rules.

11.If case is ‘+’, set the links as ‘E’ or suitable inputs as per rules.

12.Check the default case, i.e.,for single alphabet or 2 consecutive alphabets and set the links to respective alphabet.

13.End the switch case.

14.Use for loop to print the states along the matrix.

15.Use a for loop within another for lop and print the value of respective links.

16.Print the states start state as ‘0’ and final state.

17.End the program.

**PROGRAM:**

#include<stdio.h>

#include<conio.h>

void main()

{

char m[20],t[10][10];

intn,i,j,r=0,c=0;

clrscr();

printf("\n\t\t\t\tSIMULATION OF NFA");

printf("\n\t\t\t\t\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*");

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

{

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

{

t[i][j]=' ';

}

}

printf("\n\nEnter a regular expression:");

scanf("%s",m);

n=strlen(m);

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

{

switch(m[i])

{

case '|' : {

t[r][r+1]='E';

t[r+1][r+2]=m[i-1];

t[r+2][r+5]='E';

t[r][r+3]='E';

t[r+4][r+5]='E';

t[r+3][r+4]=m[i+1];

r=r+5;

break;

}

case '\*':{

t[r-1][r]='E';

t[r][r+1]='E';

t[r][r+3]='E';

t[r+1][r+2]=m[i-1];

t[r+2][r+1]='E';

t[r+2][r+3]='E';

r=r+3;

break;

}

case '+': {

t[r][r+1]=m[i-1];

t[r+1][r]='E';

r=r+1;

break;

}

default:

{

if(c==0)

{

if((isalpha(m[i]))&&(isalpha(m[i+1])))

{

t[r][r+1]=m[i];

t[r+1][r+2]=m[i+1];

r=r+2;

c=1;

}

c=1;

}

else if(c==1)

{

if(isalpha(m[i+1]))

{

t[r][r+1]=m[i+1];

r=r+1;

c=2;

}

}

else

{

if(isalpha(m[i+1]))

{

t[r][r+1]=m[i+1];

r=r+1;

c=3;

}

}

}

break;

}

}

printf("\n");

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

printf(" %d",j);

printf("\n\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\n");

printf("\n");

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

{

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

{

printf(" %c",t[i][j]);

}

printf(" | %d",i);

printf("\n");

}

printf("\nStart state: 0\nFinal state: %d",i-1);

getch();

}

**OUTPUT:**

Enter a regular Expression: a|b

SIMULATION OF NFA

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

Enter a regular expression:a|b

0 1 2 3 4 5

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

E E | 0

a | 1

E | 2

b | 3

E | 4

| 5

Start state: 0

Final state: 5

**RESULT:**

Thus the C program to convert regular expression to NFA has been executed and the output has been verified successfully.

**EX.NO : 3**

Converting  NFA to DFA

**AIM:** To write a program to convert NFA to DFA

**ALGORITHM:**

1. Start the program
2. Assign an input string terminated by end of file, DFA with start
3. The final state is assigned to F
4. Assign the state to S
5. Assign the input string to variable C
6. While C!=e of do

S=move(s,c)

C=next char

1. If it is in ten return yes else no
2. Stop the program

**SOURCE CODE:\_**

#include<conio.h>

#include<string.h>

#include<process.h>

#include<math.h>

Int n[11],I,j,c,k,h,l,h1,f,h2,temp1[12],temp2[12],count=0,ptr=0;

Char a[20][20],s[5][8];

Inttr[5][2],ecl[5][8],str[5],flag;

Inttr[5][2],ecl[5][8],st[5],flag;

Void ecls(int b[10],int x)

{

I=0;

K=-1;flag=0;

While(l<x)

{

N[++k]=b[l];

I=b[l];

h=k+1;

a:

for(j=I;j<=11;j++)

{

If(a[i][j]==’e’)

{n[++k]=j;

}

If(j==11&&h<=k)

{

I=n[h];

H++;

Goto a;

}}

L++;

}for(i=0;i<k;i++)

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

if(n[i]>n[j])

{

C=n[i];

N[i]=n[j];

N[j]=c;

}for(i=0;i<ptr;i++)

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

{

If(ecl[i][j]!=n[j])

{

If(i<count)

{i++;

J=0;

}

Else

Goto b;

}

Else if((ecl[i][j]==n[j])&&(j==k))

{tr[ptr][f]=st[i];

Flag=1;

Break;

}}

B:

If(flag==0)

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

Ecl[count][i]=n[i];

st[count]=count+65;

tr[ptr][f]=st[count];

count++;

}}

Void mova(int g)

{h1=0;

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

{if(ecl[g][i]==3)

Temp1[h1++]=4;

if(ecl[g][i]==8)

Temp1[h1++]=9;

}

Printf(“\n move(%c,a):”,st[g]);

For(i=0;i<h1;i++)

Printf(“%d”,temp1[i]);

F=0;

Ecls(temp1,h1);

}

Void movb(int g)

{

H2=0;

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

{

If(ecl[g][i]==5)

Temp2[h2++]=6;

If(ecl[g][i]==9)

Temp2[h2++]=10;

If(ecl[g][i]==10)

Temp2[h2++]=11;}

Printf(“move(%c,b):”st[g]);

For(i=0;i<h2;i++)

Printf(“%d”,temp2[i]);

F=1;

Ecls(temp2,h2);

}

Void main()

{

Clrscr();

Printf(“\n the no. of states in NFA (a/b)\*abb are:11”);

For(i=0;i<=11;i++)

For(j=0;j<=11;j++)

A[i][j]=’\0’;

A[1][2]=’e’;

A[1][8]=’e’;

A[2][3]=’e’;

A[2][5]=’e’;

A[3][4]=’a’;

A[5][6]=’b’;

A[4][7]=’e’;

A[6][7]=’e’;

A[7][8]=’e’;

A[7][2]=’e’;

A[8][9]=’a’;

A[9][10]=’b’;

A[10][11]=’b’;

Printf(“\n the transmission table is as Follows”);

Printf(“\n states 1 2 3 4 5 6 7 8 9 10 11”);

Getch();

For(i=1;i<=11;i++)

{

Printf(“\n %d \t”,i);

For(j=1;j<=11;j++)

Printf(“%c”,a[i][j]);

}

Getch();

Printf(“\n \n press any key to continue”);

Clrscr();

I=1;k=1;h=1;

N[0]=I;

Printf(“\n”);

A:

For(j=1;j<=11;j++)

{if(a[i][j]==’e’)

{

N[k++]=j;

}

If(j==11&&h<k)

{

I=n[h];

H++;

Goto a;

}}

For(i=1;j<k;i++)

For(j=i+1;j<k;j++)

If(n[i]>n[j])

{c=n[i];

N[i]=n[j];

N[j]=c;

}

Count++;

St[0]=65;

For(i=0;i<k;i++)

Ecl[0][i]=n[i];

Printf(“the moves are of the Following manner”);

Mova(ptr);

Movb(ptr);

Ptr++;}

Printf(“\n the new states of DFA are as Follows”);

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

{printf(“\n %c”,st[i]);

For(j=0;j<7;j++)

Printf”(“%d”,ecl[i][j]);

}

Printf(“ the transition table are as Follows”);

Printf(“\n a \n b \n”);

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

{

Printf(“%c”,st[i]);

For(j=0;j<2;j++)

Printf(“%c \t”,tr[i][j]);

}

Getch();

}

**OUTPUT:**

The no. of states in NFA(a/b)\*abb are:11

The transition table is as Follows

1 2 3 4 5 6 7 8 9 10 11

1 e e

2 e e

3 a

4 e

5 b e

6 e

7 e e

8

9 a

10 b

11 b

**RESULT:**

Thus the above conversion program is successfully executed

**EX.NO : 4 Elimination of Ambiguity, Left Recursion and Left Factoring**

**AIM:**

To Write a program Elimination of Ambiguity, Left Recursion and Left Factoring

**PROGRAM:**

#include<stdio.h>

#include<string.h>

void main() {

char input[100],l[50],r[50],temp[10],tempprod[20],productions[25][50];

int i=0,j=0,flag=0,consumed=0;

printf("Enter the productions: ");

printf("Enter the productions: ");

printf("Enter the productions: ");

printf("Enter the productions: ");

scanf("%1s->%s",l,r);

printf("%s",r);

while(sscanf(r+consumed,"%[^|]s",temp) == 1 && consumed <= strlen(r)) {

if(temp[0] == l[0]) {

flag = 1;

sprintf(productions[i++],"%s->%s%s'\0",l,temp+1,l);

} else

sprintf(productions[i++],"%s'->%s%s'\0",l,temp,l);

consumed += strlen(temp)+1;

}

if(flag == 1) {

sprintf(productions[i++],"%s->ε\0",l);

printf("The productions after eliminating Left Recursion are:\n");

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

printf("%s\n",productions[j]);

} else

printf("The Given Grammar has no Left Recursion");

}

**Output:**

Enter the productions: E->E+E|T

The productions after eliminating Left Recursion are:

E->+EE'

E'->TE'

E->ε

**EX.NO : 5**

Computation of FIRST and FOLLOW sets

**AIM:** To calculate the first and Follow of the given expression

**ALGORITHM:**

1. Start the program
2. In the production the first terminal on R.H.S becomes the first of it
3. If the first character is non-terminal then its first is taken else Follow of left is taken
4. To find Follow find where all the non terminals appear. the first of its Follows is its Follow
5. If the Follow is t then the Follow of left is taken
6. Finally print first and its Follow
7. Stop the program

**SOURCE CODE:\_**

#include<stdio.h>

#include<conio.h>

#include<process.h>

Void main()

{

Intnop,x=0,y=0,l=0,k=0,c=0,s=0,z=0;

Char p[10][10],o,fi[10][10],fo[10][10];

Clrscr();

For(x=0;x<10;x++)

{

For(y=0;y<10;y++)

{

P[x][y]=’0’;

Fi[x][y]=’0’;

Fo[x][y]=’0’;

}}

Printf(“enter the no of productions:”);

Scanf(“%d”,&nop);

Printf(“\n enter the number of production”);

For(x=0;x<nop;x++)

{

Scanf(“%s”,&p[x]);

If(p[x][0]==p[x][2])

{

Printf(“\a production is not free from left recursion”);

Printf(“\a program has to be terminated”);

Printf(”\a press any key”);

Getch();

Exit(0);

}}

Printf(“\n\n”)

Printf(“first \n”);

For(y=0;y<nop;y++)

{

Printf(“first(%c)=”,p[y][0]);

If(p[y][2]>=’A’ &&p[y][2]<=’z’)

{

O=p[y][2];

For(x=y+1;x<nop;x++)

{

If(p[x][0]==o)

{

If(p[x][2]>=’A’&&p[x][2]<=’z’)

O=p[x][2];

Else if(p[x][2]<’A’||p[x][2]>’z’)

{

Printf(“%c”,p[x][2]);

Fi[y][k++]=p[x][2]);

For(l=0;l<strlen(p[x]);l++)

{

If(p[x][l]==’/’)

{

Printf(“%c”,p[x][l+1]);

Fi[y][k++]=p[x][l+1];

Break;

}}}}}}

Else if(p[y][2]<’A’ || p[y][2]>’Z’)

{

Printf(“%c”,p[y][2]);

Fi[y][k++]=p[y][2];

}

L=strlen(p[y]);

For(c=0;c<l;c++)

{

If(p[y][c]==’/’)

{

Printf(“%c”,p[y][c+1]);

Fi[y][k++]=p[y][c+1];

}}

Printf(“\n”);

K=0;

Printf(“Follow \n”);

For(x=0;x<nop;x++)

{

For(y=0;y<nop;y++)

For(l=2;l<strlen(p[x]);l++)

If(p[y][l]==p[x][0])

{

If(p[y][l+1]<’A’||p[y][l+1]>’Z’||p[y][l+1]!=’/’||p[y][l+1]!=’0’)

{

Fo[x][k++]=p[y][l+1];

If(x==0)

{

Fo[x][k++]=’$’;

}}

C=k;

If(p[y][l+1]==’0’||p[y][l+1]==’/’)

{

For(s=0;s<c+10;s++)

{

Fo[x][k++]=fo[x-1][s];

}}

C=k;

If(p[y][l+1]>=’A’&&p[y][l+1]<=’z’)

{

For(s=0;s<=c;s++)

{

Fo[x][k++]=fo[x-1][s];

Fo[x][k++]=fo[x-2][s];

Fo[x][k++]=fi[x-1][s];

}}

C=k;

}

Printf(“Follow(%c)=”,p[x][0]);

For(z=0;z<=k+10;z++)

If(fo[x][z]==’\*’||fo[x][z]==’+’||fo[x][z]==’$’||fo[x][z]==’)’||fo[x][z]==’(‘||fo[x][z]==’-‘||fo[x][z]==’%’)

Printf(“%c”,fo[x][z]);

K=0;

Printf(“\n”);

}

Getch();

}

**OUTPUT:**

Enter the no. of production:5

**E->TE’**

**E’->+TE’/e**

**T->FT’**

**T’->\*FT’/e**

**F->(E) /id**

First

First(E)={(,id}

First(E’)={+,e}

First(T)={(,id}

First(T’)={\*,e}

First(F)={(,id}

Follow

Follow(E)={),$}

Follow(E’)={),+,$}

Follow(T)={),+,$}

Follow(T’)={\*,+,),$}

Follow(F)={\*,+,),$}

**RESULT:**

Thus the above computation of FIRST &FOLLOW program is successfully executed.

**EX.NO :6**

Construction of Predictive Parsing Table

**AIM:** To write a C program for the implementation of predictive parsing table

**ALGORITHM:**

1. start the program
2. Assign an input string and parsing table in for then G.
3. Set ip to point to the first symbol of to $
4. Repeat if X is a terminal of $ then if n=a,then pop X from the stack
5. Push Y into the stack with Y,on top
6. Output the production x->x,y
7. End else error until x=$
8. Terminate the program

**SOURCE CODE:\_**

#include<stdio.h>

#include<conio.h>

#include<string.h>

char str[10],out,in,output[10],input[10],temp;

char tl[10]={‘x’,’+’,’\*’,’(‘,’)’,’$’,’@’};

char ntl[10]={‘e’,’e’,’t’,’t’,’f’};

int err=0,flag=0,i,j,k,l,m;

char c[10][10][7]={{{“te”},{“error!”},{“error!”},{“te”},{“error!”},{“error!”},},{“error!”,”te”,”error!”,”error”,”@”,”@”},{“ft”,”error!”,”error”,”ft””error!”,”error”},{“error”,”@”,”\*ft”,”error!”,”@”,”@”},{“x”,”error!”,”error!”,”(e)”,”error!”,”error!”}};

struct stack

{

char sic[10];

int top;

};

void push(struct stack \*s,char p)

{

s->sic[++s->top]=p;

s->sic[s->top+1]=’\0’;

}

char pop(struct stack \*s)

{

char a;

a=s->sic[s->top];

s->sic[s->top--]=’\0’;

return(a);

}

char stop(struct stack \*s)

{

return(s->sic[s->top]);

}

voidpobo(struct stack \*s)

{

m=o;

while(str[m]!=’\0’)

m++;

m--;

while(m!=-1)

{

if(str[m]!=’@’)

push(s,str[m]);

m--;

}}

void search(int l)

{

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

if(in==tl[k])

break;

if(l==0)

strcpy(str,c[l][k]);

else if(l==1)

strcpy(str,c[l][k]);

else if(l==2)

strcpy(str,c[l][k]);

else if(l==3)

strcpy(str,c[l][k]);

elsestrcpy(str,c[l][k]);}

void main()

{

struct stack s1;

struct stack \*s;

s=&s1;

s->top=-1;

clrscr();

printf(“\t\t parsing table \t\t”);

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

{

printf(“%c\t”,ntl[i];

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

if(strcmp(c[i][j],”error!”)==0)

printf(“error!\t”);

else

printf(“%c->%s” \t”,ntl[i],c[i][j]);

}

push(s,’$’);

push(s,’e’);

printf(“enter the input string”);

scanf(“%s”,input);

printf(“\n\n the behaviour of the parser for given input string is: \n\ “);

printf(“\n stack\n input\n output”);

i=0;

in=input[i];

printf(“%s\t”,s->sic);

for(k=i;k<strlen(input);k++)

printf(“%c”,input[k]);

if(strcmp(str,’ ‘)!=0)

printf(“\t%c->%s”ntl[j],str);

while((s->sic[s->top]!=’$’)&&err!=1&&strcmp(str,”error!”)!=0)

{

strcpy(str,” “);

flag=0;

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

if(in==tl[j])

{

flag=1;

break;

}

if(flag==0)

in=’x’;

flag=0;

out=stop(s);

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

if(out==tl[j])

{

flag=1;

break;

}

if(flag==1)

{

if(out==in)

{

temp=pop(s);

in=input[++i];

if(str==’@’)

temp=pop(s);

}

else

{

strcpy(sstr,”error!”);

err=1;

}}

else

{

flag=0;

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

if(out==ntl[j])

{

flag=1;

break;

}

if(flag==1)

{

search(j);

temp=pop(s);

pobo(s);

}

else

{

strcpy(str,”error!”);

err=1;

}}

if(strcmp(str,”error!”)!=0)

{

printf(“%s\t”,s->sic);

for(k=i;k<strlen(input);k++)

printf(“%c”,input[k]);

if((strcmp(str,” “)!=0)&&(strcmp(str,”error!”)!=0))

printf(“\t %c->%s”,ntl[j],str);

}}

if(strcmp(str,”error!”)==0)

printf(“\n the string is not accepted!!”);

else

printf(“\t \t accept \n\n\n the string is accepted”);

getch();

}

**OUTPUT:**

Parsing table

X + \* ( ) $

E.E->Te ERROR! ERROR! E->teERROR! ERROR!

E ERROR! E->+teERROR! ERROR! E->@ e->@

T T->Ft ERROR! ERROR! T->ftERROR! ERROR!

T ERROR! T->@ t->\*ft ERROR t->@ t->@

F.F->x ERROR! ERROR! F-> (E) ERROR! ERROR!

Enter the input string: x+X$

The behaviour of the parser for given input string is

Stack input output

SE X+X$

SeT X+X$ E->Te

Set F X+X$ T->Ft

Set X X+X$ F->X

Set +X$

Se +X$ t->@

SeT+ +X$ e->+Te

seT X$

SetF X$ T->Ft

**RESULT:**

Thus the Predictive Parser program is executed successfully.

**EX.NO : 7**

**Implementation of Shift Reduce Parsing**

**AIM:**To write a C program for shift reduce parsing

**ALGORITHM:**

1. start the program
2. read the expression and declare the variables
3. set $ symbol to indicate the start of stack
4. Repeat for i=0 to n where n is the no. of productions

A. read prod(i)

B.set problem[i]=strlen(prod[i])

5. check for the non-terminla which corresponds to the terminal

6. If it equals then replace the terminal with non-terminal

7. Repeat this until terminals are replaced by non-terminals

**SOURCE CODE:\_**

#include<iostream.h>

#include<conio.h>

#include<string.h>

#include<stdlib.h>

void push(char);

char pop();

voidprintstack();

struct grammar

{

charlpr,rpr[10];

};char stack[20];

int top=-1;

void main()

{

grammar gr[10];

char buffer[10];

char ch,ch1,temp[10],start;

inti,j,k,s,t,len;

clrscr();

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

cin>>n;

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

{

cout<<"\n enter the left side of productions"<<i+1<<":";

cin>>gr[i].lpr;

cout<<"\n enter the right side of productions"<<":";

cin>>gr[i].rpr;

}

cout<<"\n enter the input string:";

cin>>buffer;

cout<<"\n the grammaris:\n";

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

cout<<gr[i].lpr<<"-->"<<gr[i].rpr<<endl;

cout<<"\n input string is :"<<buffer;

push('$');

start=gr[0].lpr;

len=strlen(buffer);

buffer[len]='$';

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

cout<<"\n \n stack \t\t buffer\t\t\t action\n";

cout<<stack<<"\t\t"<<buffer<<endl;

getch();

while(1)

{

ch=buffer[i];

lab:t=0;

for(k=top;k>0;k--)

{

temp[t++]=stack[t]='\0';

strrev(temp);

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

{

if(strcmp(temp,gr[j].rpr)==0)

{

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

ch1=pop();

push(gr[j].lpr);

printstack();

cout<<"\t\t\t"<<&buffer[i]<<"\t\tReduce"<<endl;

getch();

goto lab;

}}

strrev(temp);

}

ch1=pop();

if(ch!='$')

{push(ch1);

push(ch);

printstack();

cout<<"\t\t\t"<<&buffer[i+1]<<"\t\tshift"<<endl;

getch();

i++;

}

else if(ch=='$' && ch1==start && top==0)

{

cout<<"\n string is accepted";

getch();

exit(0);

}

else

{

cout<<"\n string is not accepted";

getch();

exit(0);

}

}

}

void push(char a)

{

stack[++top]=a;

}

char pop(){

return stack[top--];}

voidprintstack()

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

cout<<stack[i];

}

**OUTPUT:**

Enter the no. of production: 3

Enter the production: S->aABC

A->Abc/b

B->d

The production are

S>>aABC

A>>Abc/b

B>>d

ILR

S>>aABC

u>>bc/bu

B>>d

u>x

the first symbols are X

**RESULT:**

Thus the shift reduce parser program is successfully executed

**Ex.no.8**

Computation of Follow and Trailing Sets

**AIM:** To write a C program to Compute of Follow and Trailing Sets

**ALGORITHM:**

tep tep1: Start.

Step1 : Start the Program: Start.

Step2: Get the no of production and calculate the length of each production.

Step3: With a variable val for checking the valid non terminals if they are duplicated get all Non terminals in an array.

Step4: In each production check the first accurate of terminals and take that terminal and add it to Follow of non terminal in array and exitthe loop.

Step5: Scan the production and find the last terminal and add it to respective trailing array of associated non terminal &exit the loop.

Step6: Consider production with a non terminal on right side and check the Follow of that non terminal associated production and also trailing and add it to the Follow and trailing of left side non terminal.

Step7: Write the Follow and trailing terminals for each non terminal.

**SOURCE CODE:\_**

#include<iostream.h>

#include<stdio.h>

#include<conio.h>

#include<string.h>

#include<ctype.h>

char nt[5],p[5],q[5],a[5][5][5],b[5][5][5],fi[5][5],left[5],right[5],lead[5][10],trail[5][10];

int n1,n[5],c[5][5],m,l[5],f[5],k,a1;

void leading(char,int);

void trailing(char,int);

void main()

{

clrscr();

cout<<"Enter the number of non-terminals ";

cin>>n1;

cout<<"Enter the set of non-terminals ";

for(int i=0;i<n1;i++)

cin>>nt[i];

for(i=0;i<n1;i++)

{

cout<<endl;

cout<<"Enter the number of productions for "<<nt[i]<<" ";

cin>>n[i];

for(int j=0;j<n[i];j++)

{

cout<<"\nEnter the productions ";

gets(p);

//q=strrev(p);

c[i][j]=strlen(p);

for(int k=0;k<c[i][j];k++)

{

a[i][j][k]=p[k];

b[i][j][k]=p[c[i][j]-k-1];

} }

}

for(i=0;i<n1;i++)

{

cout<<endl;

cout<<nt[i]<<"--->";

for(int j=0;j<n[i];j++)

{

for(int k=0;k<c[i][j];k++)

cout<<a[i][j][k];

cout<<"/";

} }

cout<<"\n\n";

char x;

for(i=0;i<n1;i++)

{

l[i]=0;

f[i]=0;

x=nt[i];

k=i;

leading(x,k);

trailing(x,k);

}

/\*for(intmn=0;mn<=n1;mn++)

{

cout<<right[mn]<<" ";

} \*/

int count=0;

char z;

for(intmn=0;mn<n1;mn++)

{

for(i=0;i<n1;i++)

{

z=right[i];

for(int k=0;k<n1;k++)

{

if(z==nt[k])

{

for(int d=0;d<l[k];d++)

{

for(int g=0;g<l[i];g++)

{

if(lead[i][g]==lead[k][d])

count=1;

}

if(count==0)

{

lead[i][l[i]-1]=lead[k][d];

l[i]++;

}

count=0;

}

for(d=0;d<f[k];d++)

{

for(int g=0;g<f[i];g++)

{

if(trail[i][g]==trail[k][d])

count=1;

}

if(count==0)

{

trail[i][f[i]-1]=trail[k][d];

f[i]++;

}

count=0;

}

}

}

count=0;

}

}

for(i=0;i<n1;i++)

{

cout<<"Leading("<<nt[i]<<")= {";

for(int j=0;j<l[i];j++)

{

cout<<lead[i][j]<<" ";

}

cout<<"}\t\t\t\t";

cout<<"Trailing("<<nt[i]<<")= {";

for(j=0;j<f[i];j++)

{

cout<<trail[i][j]<<" ";

}

cout<<"}\n";

}

getch();

}

void leading(char x,int i)

{

char z;

for(int j=0;j<n[i];j++)

{

if(isupper(a[i][j][0]))

{

left[i]=nt[i];

right[i]=a[i][j][0];

if(!isupper(a[i][j][1]))

{

//cout<<a[i][j][1]<<" ";

lead[i][l[i]]=a[i][j][1];

l[i]++;

}

}

else

{

//cout<<a[i][j][0]<<" ";

lead[i][l[i]]=a[i][j][0];

l[i]++;

}

}

}

void trailing(char x,int i)

{

char z;

for(int j=0;j<n[i];j++)

{

if(isupper(b[i][j][0]))

{

if(!isupper(b[i][j][1]))

{

//cout<<a[i][j][1]<<" ";

trail[i][f[i]]=b[i][j][1];

f[i]++;

}

}

else

{

//cout<<a[i][j][0]<<" ";

trail[i][f[i]]=b[i][j][0];

f[i]++;

}

}

}

**OUTPUT-**

Enter the no. of production:5

E->TE’

E’->+TE’/e

T->FT’

T’->\*FT’/e

F->(E) /id

Leading

Leading (E)={(,id}

Leading (E’)={+,e}

Leading (T)={(,id}

Leading (T’)={\*,e}

Leading (F)={(,id}

Trailing

Trailing (E)={),$}

Trailing (E’)={),+,$}

Trailing (T)={),+,$}

Trailing (T’)={\*,+,),$}

F Trailing (F)={\*,+,),$}

**RESULT:** Thus the Leading and Trailing was executed and verified Successfully

**EX.NO.9**

Computation of LR (0) items

**AIM:To write a code for LR(0) Parser for Following Production:**

**E->E+T**

**T->T\*F/F**

**F->(E)/char**

**ALGORITHM:**

**1.Initialize the stack with the start state.**

**2. Read an input symbol**

**3. while true do**

**3.1 Using the top of the stack and the input symbol determine the next state.**

**3.2 If the next state is a stack state**

**then3.2.1 stack the state**

**3.2.2 get the next input symbol**

**3.3 else if the next state is a reduce state**

**then**

**3.3.1 output reduction number, k**

**3.3.2 pop RHSk -1 states from the stack where RHSk is the right hand side of production k.**

**3.3.3 set the next input symbol to the LHSk**

**3.4 else if the next state is an accept state**

**then**

**3.4.1 output valid sentence**

**3.4.2 return**

**else**

**3.4.3 output invalid sentence**

**3.4.4 return**

**SOURCE CODE:\_**

**#include<string.h>**

**#include<conio.h>**

**#include<stdio.h>**

**intaxn[][6][2]={**

**{{100,5},{-1,-1},{-1,-1},{100,4},{-1,-1},{-1,-1}},**

**{{-1,-1},{100,6},{-1,-1},{-1,-1},{-1,-1},{102,102}},**

**{{-1,-1},{101,2},{100,7},{-1,-1},{101,2},{101,2}},**

**{{-1,-1},{101,4},{101,4},{-1,-1},{101,4},{101,4}},**

**{{100,5},{-1,-1},{-1,-1},{100,4},{-1,-1},{-1,-1}},**

**{{100,5},{101,6},{101,6},{-1,-1},{101,6},{101,6}},**

**{{100,5},{-1,-1},{-1,-1},{-1,-1},{-1,-1},{-1,-1}},**

**{{100,5},{-1,-1},{-1,-1},{100,4},{-1,-1},{-1,-1}},**

**{{-1,-1},{100,6},{-1,-1},{-1,-1},{100,11},{-1,-1}},**

**{{-1,-1},{101,1},{100,7},{-1,-1},{101,1},{101,1}},**

**{{-1,-1},{101,3},{101,3},{-1,-1},{101,3},{101,3}},**

**{{-1,-1},{101,5},{101,5},{-1,-1},{101,5},{101,5}}**

**};**

**int gotot[12][3]={1,2,3,-1,-1,-1,-1,-1,-1,-1,-1,-1,8,2,3,-1,-1,-1,-1,**

**9,3,-1,-1,10,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1};**

**int a[10];**

**char b[10];**

**int top=-1,btop=-1,i;**

**void push(int k)**

**{**

**if(top<9)**

**a[++top]=k;**

**}**

**voidpushb(char k)**

**{**

**if(btop<9)**

**b[++btop]=k;**

**}**

**char TOS()**

**{**

**return a[top];**

**}**

**void pop()**

**{**

**if(top>=0)**

**top--;**

**}**

**voidpopb()**

**{**

**if(btop>=0)**

**b[btop--]='\0';**

**}**

**void display()**

**{**

**for(i=0;i<=top;i++)**

**printf("%d%c",a[i],b[i]);**

**}**

**void display1(char p[],int m)**

**{**

**int l;**

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

**for(l=m;p[l]!='\0';l++)**

**printf("%c",p[l]);**

**printf("\n");**

**}**

**void error()**

**{**

**printf("\n\nSyntax Error");**

**}**

**void reduce(int p)**

**{**

**intlen,k,ad;**

**charsrc,\*dest;**

**switch(p)**

**{**

**case 1:dest="E+T";**

**src='E';**

**break;**

**case 2:dest="T";**

**src='E';**

**break;**

**case 3:dest="T\*F";**

**src='T';**

**break;**

**case 4:dest="F";**

**src='T';**

**break;**

**case 5:dest="(E)";**

**src='F';**

**break;**

**case 6:dest="i";**

**src='F';**

**break;**

**default:dest="\0";**

**src='\0';**

**break;**

**}**

**for(k=0;k<strlen(dest);k++)**

**{**

**pop();**

**popb();**

**}**

**pushb(src);**

**switch(src)**

**{**

**case 'E': ad=0;**

**break;**

**case 'T': ad=1;**

**break;**

**case 'F': ad=2;**

**break;**

**default: ad=-1;**

**break;**

**}**

**push(gotot[TOS()][ad]);**

**}**

**int main()**

**{**

**intj,st,ic;**

**charip[20]="\0",an;**

**clrscr();**

**printf("Enter any String :- ");**

**gets(ip);**

**push(0);**

**display();**

**printf("\t%s\n",ip);**

**for(j=0;ip[j]!='\0';)**

**{**

**st=TOS();**

**an=ip[j];**

**if(an>='a'&an<='z')**

**ic=0;**

**else if(an=='+')**

**ic=1;**

**else if(an=='\*')**

**ic=2;**

**else if(an=='(')**

**ic=3;**

**else if(an==')')**

**ic=4;**

**else if(an=='$')**

**ic=5;**

**else**

**{**

**error();**

**break;**

**}**

**if(axn[st][ic][0]==100)**

**{**

**pushb(an);**

**push(axn[st][ic][1]);**

**display();**

**j++;**

**display1(ip,j);**

**}**

**if(axn[st][ic][0]==101)**

**{**

**reduce(axn[st][ic][1]);**

**display();**

**display1(ip,j);**

**}**

**if(axn[st][ic][1]==102)**

**{**

**printf("Given String is Accepted");**

**break;**

**}**

**}**

**getch();**

**return 0;**

**}**

**OUTPUT**

**Enter any String :-  a+b\*c**

**0                       a+b\*c**

**0a5                  +b\*c**

**0F3                  +b\*c**

**0T2                  +b\*c**

**0E1                  +b\*c**

**0E1+6               b\*c**

**0E1+6b5            \*c**

**0E1+6F3            \*c**

**0E1+6T9            \*c**

**0E1+6T9\*7          c**

**0E1+6T9\*7c5**

**RESULT:** Thus the LR(0) Program was executed and verified Successfully.

**EX.NO.10**

**Intermediate Code Generation**

**AIM**:To write a C program to implementation of code generation

**Program**

#include<stdio.h>

#include<conio.h>

#include<string.h>

**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\n");

printf("Enter the Expression :");

scanf("%s",str);

printf("The intermediate code:\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);

printf("\n");

i++;

}

fright(-1);

**if**(no==0)

{

fleft(strlen(str));

printf("\t%s := %s",right,left);

getch();

exit(0);

}

printf("\t%s := %c",right,str[k[--i].pos]);

getch();

}

**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]!='\0'&&str[x]!='='&&str[x]!=':'&&str[x]!='-'&&str[x]!='/')

{

**if**(str[x]!='$'&& flag==0)

{

right[w++]=str[x];

right[w]='\0';

str[x]='$';

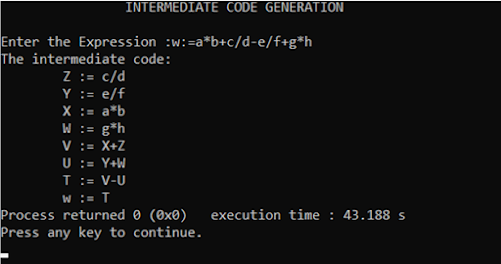
flag=1;

}

x++;

}

}

[](https://lh3.googleusercontent.com/-9nbeUxFSS8I/X-v3u0CTf-I/AAAAAAAAfhI/GY4vLcf4m8EjwtzfNGkZrAD1R7FBjGbwwCLcBGAsYHQ/image.png)

**EX.NO : 11**

Intermediate Code Generation

**AIM:** To write a C program to implementation of code generation

**ALGORITHM:**

step 1: Start.

Step 2: Enter the three address codes.

Step 3: If the code constitutes only memory operands they are moved to

the register and according to the operation the corresponding

assembly code is generated.

Step 4: If the code constitutes immediate operands then the code will have

a # symbol proceeding the number in code.

Step 5: If the operand or three address code involve pointers then the code

generated will constitute pointer register. This content may be

stored to other location or vice versa.

Step 6: Appropriate functions and other relevant display statements are

executed.

Step 7: Stop.

**SOURCE CODE:\_**

#include<stdio.h>

#include<string.h>

voidpm();

voidplus();

voiddiv();

inti,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%sgoto %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 :

Example Generation of Three Address Project Output Result

1. assignment

2. arithmetic

3. relational

4. Exit

Enter the choice:1

Enter the expression with assignment operator:

a=b

Three address code:

temp=b

a=temp

1.assignment

2.arithmetic

3.relational

4.Exit

Enter the choice:2

Enter the expression with arithmetic operator:

a+b-c

Three address code:

temp=a+b

temp1=temp-c

1.assignment

2.arithmetic

3.relational

4.Exit

Enter the choice:2

Enter the expression with arithmetic operator:

a-b/c

Three address code:

temp=b/c

temp1=a-temp

1.assignment

2.arithmetic

3.relational

4.Exit

Enter the choice:2

Enter the expression with arithmetic operator:

a\*b-c

Three address code:

temp=a\*b

temp1=temp-c

1.assignment

2.arithmetic

3.relational

4.Exit

Enter the choice:2

Enter the expression with arithmetic operator:a/b\*c

Three address code:

temp=a/b

temp1=temp\*c

1.assignment

2.arithmetic

3.relational

4.Exit

Enter the choice:3

Enter the expression with relational operator

a

<=

b

100 if a<=b goto 103

101 T:=0

102 goto 104

103 T:=1

1.assignment

2.arithmetic

3.relational

4.Exit

Enter the choice:4

**RESULT:** Thus the Generation of Three Address was executed and verified Successfully.

**EX.NO : 12**

**Intermediate Code Generation**

AIM:To write a C program to perform the Intermediate Code Generation

**ALGORITHM:**

**PROGRAM:**

#include<stdio.h>  
#include<conio.h>  
#include<string.h>  
char op[2],arg1[5],arg2[5],result[5];  
void main()  
{  
  FILE \*fp1,\*fp2;  
  fp1=fopen("input.txt","r");  
  fp2=fopen("output.txt","w");  
  while(!feof(fp1))  
  {  
  
    fscanf(fp1,"%s%s%s%s",op,arg1,arg2,result);  
    if(strcmp(op,"+")==0)  
    {  
      fprintf(fp2,"\nMOV R0,%s",arg1);  
      fprintf(fp2,"\nADD R0,%s",arg2);  
      fprintf(fp2,"\nMOV %s,R0",result);  
    }  
     if(strcmp(op,"\*")==0)  
    {  
      fprintf(fp2,"\nMOV R0,%s",arg1);  
      fprintf(fp2,"\nMUL R0,%s",arg2);  
      fprintf(fp2,"\nMOV %s,R0",result);  
    }  
    if(strcmp(op,"-")==0)  
    {  
      fprintf(fp2,"\nMOV R0,%s",arg1);  
      fprintf(fp2,"\nSUB R0,%s",arg2);  
      fprintf(fp2,"\nMOV %s,R0",result);  
    }  
       if(strcmp(op,"/")==0)  
    {  
      fprintf(fp2,"\nMOV R0,%s",arg1);  
      fprintf(fp2,"\nDIV R0,%s",arg2);  
      fprintf(fp2,"\nMOV %s,R0",result);  
    }  
if(strcmp(op,"=")==0)  
    {  
      fprintf(fp2,"\nMOV R0,%s",arg1);  
      fprintf(fp2,"\nMOV %s,R0",result);  
    }  
    }  
    fclose(fp1);  
    fclose(fp2);  
    getch();  
  }  
}  
  
input.txt  
  
+ a b t1  
\* c d t2  
- t1 t2 t  
= t ? x

output.txt

MOV R0,a

ADD R0,b

MOV t1,R0

MOV R0,c

MUL R0,d

MOV t2,R0

MOV R0,t1

SUB R0,t2

MOV t,R0

MOV R0,t

MOV x,R0

**EX.NO.13**

**Construction of DAG**

**AIM:** To write a C program to perform the Construction of DAG

**ALGORITHM:**

**PROGRAM CODE:**

//*Construction of DAG*

#include<stdio.h>

#include<stdlib.h>

#include<time.h>

#define MIN\_PER\_RANK 1

#define MAX\_PER\_RANK 5

#define MIN\_RANKS 3

#define MAX\_RANKS 5

#define PERCENT 30

void main()

{

int i,j,k,nodes=0;

srand(time(NULL));

int ranks=MIN\_RANKS+(rand()%(MAX\_RANKS-MIN\_RANKS+1));

printf("DIRECTED ACYCLIC GRAPH\n");

for(i=1;i<ranks;i++)

{

int new\_nodes=MIN\_PER\_RANK+(rand()%(MAX\_PER\_RANK-MIN\_PER\_RANK+1));

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

for(k=0;k<new\_nodes;k++)

if((rand()%100)<PERCENT)

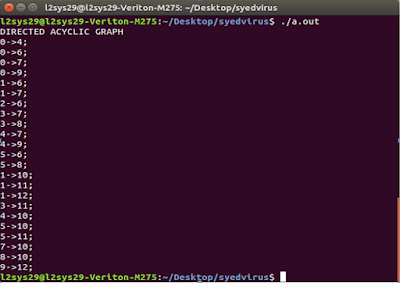
printf("%d->%d;\n",j,k+nodes);

nodes+=new\_nodes;

}

}

**OUTPUT:**

[](https://4.bp.blogspot.com/-L7yHPLytXlQ/Wszx25BBcfI/AAAAAAAAC_8/vH84RS1Yga81qLATV0tccA_QVdsLG0hmgCLcBGAs/s1600/dag.PNG)

**RESULT:** Thus the DAG was executed and verified Successfully.

**EX.NO.14**

**Implementation of Global Data Flow Analysis**

**AIM:**

To write a C program to Implementation of Global Data Flow Analysis.

**ALGORITHM:**

step 1: Start.

Step 2: Enter the Global Data Flow Analysis

Step 3: If the code constitutes only memory operands they are moved to

the register and according to the operation the corresponding

assembly code is generated.

Step 4: If the code constitutes immediate operands then the code will have

a # symbol proceeding the number in code.

Step 5: If the operand or three address code involve pointers then the code

generated will constitute pointer register. This content may be

stored to other location or vice versa.

Step 6: Appropriate functions and other relevant display statements are

executed.

Step 7: Stop.

**PROGRAM :**

#include<stdio.h>

#include<string.h>

#include<ctype.h>

void input();

void output();

void change(int p,int q,char \*res);

void constant();

void expression();

struct expr

{

char op[2],op1[5],op2[5],res[5];

int flag;

}arr[10];

int n;

int main()

{

int ch=0;

input();

constant();

expression();

output();

}

void input()

{

int i;

printf("\n\nEnter the maximum number of expressions:");

scanf("%d",&n);

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

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

{

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

scanf("%s",arr[i].op1);

scanf("%s",arr[i].op2);

scanf("%s",arr[i].res);

arr[i].flag=0;

}

}

void constant()

{

int i;

int op1,op2,res;

char op,res1[5];

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

{

if(isdigit(arr[i].op1[0]) && isdigit(arr[i].op2[0]))

{

op1=atoi(arr[i].op1);

op2=atoi(arr[i].op2);

op=arr[i].op[0];

switch(op)

{

case '+':

res=op1+op2;

break;

case '-':

res=op1-op2;

break;

case '\*':

res=op1\*op2;

break;

case '/':

res=op1/op2;

break;

}

sprintf(res1,"%d",res);

arr[i].flag=1;

change(i,i,res1);

}

}

}

void expression()

{

int i,j;

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

{

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

{

if(strcmp(arr[i].op,arr[j].op)==0)

{

if(strcmp(arr[i].op,"+")==0||strcmp(arr[i].op,"\*")==0)

{

if(strcmp(arr[i].op1,arr[j].op1)==0&&strcmp(arr[i].op2,arr[j].op2)==0 || strcmp(arr[i].op1,arr[j].op2)==0&&strcmp(arr[i].op2,arr[j].op1)==0)

{

arr[j].flag=1;

change(i,j,NULL);

}

}

else

{

if(strcmp(arr[i].op1,arr[j].op1)==0&&strcmp(arr[i].op2,arr[j].op2)==0)

{

arr[j].flag=1;

change(i,j,NULL);

}          }

}          }

}          }

void output()

{

int i=0;

printf("\nOptimized code is : ");

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

{

if(!arr[i].flag)

{

printf("\n%s %s %s %s\n",arr[i].op,arr[i].op1,arr[i].op2,arr[i].res);

}

}

}

void change(int p,int q,char \*res)

{

int i;

for(i=q+1;i<n;i++)

{

if(strcmp(arr[q].res,arr[i].op1)==0)

if(res == NULL)

strcpy(arr[i].op1,arr[p].res);

else

strcpy(arr[i].op1,res);

else if(strcmp(arr[q].res,arr[i].op2)==0)

if(res == NULL)

strcpy(arr[i].op2,arr[p].res);

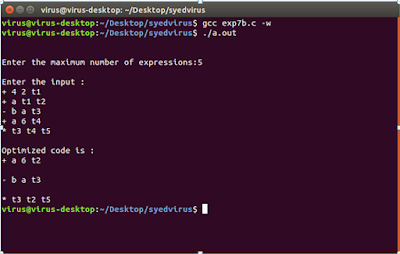
else

strcpy(arr[i].op2,res);

}

}

**OUTPUT:**

[](https://4.bp.blogspot.com/-HVlnlH5Oefw/WszymztYF7I/AAAAAAAADAE/qpl9ntLnUncGwmwEMKxurH9cVXLDmi42wCLcBGAs/s1600/DF.PNG)

**EX.NO.:**15

**Implement any one storage allocation strategies**

**(heap, stack, static)**

**AIM:**

 To implement Stack storage allocation strategies using C program.

**ALGORITHM:**

**Step1:** Initially check whether the stack is empty

**Step2:** Insert an element into the stack using push operation

**Step3:** Insert more elements onto the stack until stack becomes full

**Step4:** Delete an element from the stack using pop operation

**Step5:** Display the elements in the stack

**Step6:** Top the stack element will be displayed

**PROGRAM CODE:**

//*implementation of heap allocation storage strategies//*

#include<stdio.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

{

printf("\nprogram to perform various operations on heap using dynamic memory management");

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("\nenter your chioce(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:

printf("invalid choice,try again");

}

}

while(choice!=5);

}

node\* create()

{

node \*temp,\*New,\*head;

int val,flag;

char ans='y';

node \*get\_node();

temp=NULL;

flag=TRUE;

do

{

printf("\n enter the element:");

scanf("%d",&val);

New=get\_node();

if(New==NULL)

printf("\nmemory is not allocated");

New->data=val;

if(flag==TRUE)

{

head=New;

temp=head;

flag=FALSE;

}

else

{

temp->next=New;

temp=New;

}

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

}

while(ans=='y');

printf("\nthe list is created\n");

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("\nthe list is empty\n");

return;

}

while(temp!=NULL)

{

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

temp=temp->next;

}

printf("NULL");

}

node \*search(node \*head,int key)

{

node \*temp;

int found;

temp=head;

if(temp==NULL)

{

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

return NULL;

}

found=FALSE;

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

{

if(temp->data!=key)

temp=temp->next;

else

found=TRUE;

}

if(found==TRUE)

{

printf("\nthe element is present in the list\n");

return temp;

}

else

{

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

return NULL;

}

}

node \*insert(node \*head)

{

int choice;

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

void insert\_after(node \*);

void insert\_last(node \*);

printf("n1.insert a node as a head node");

printf("n2.insert a node as a head node");

printf("n3.insert a node at intermediate position in t6he 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("\nEnter 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("\nenter 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("\nenter the elements which you want to insert");

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

if(head==NULL)

{

head=New;

}

else

{

printf("\enter the element which you want to insert the node");

scanf("%d",&key);

temp=head;

do

{

if(temp->data==key)

{

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)

{

node \*temp,\*prev;

int key;

temp=\*head;

if(temp==NULL)

{

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

return;

}

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);

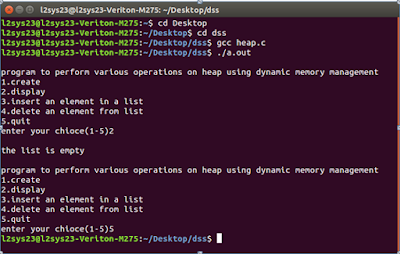
}

printf("\nthe element is deleted\n");

}

}

**OUTPUT:**

[](https://4.bp.blogspot.com/-fcAIHCdfFp0/WszwHvyjrbI/AAAAAAAAC_c/avwCtvN9gM0QSCZ_eOZipNYnnp0l53UZgCLcBGAs/s1600/8.PNG)