**Practical 7**

**Implementation of Directed Acyclic Graph**

**Introduction**

In graph theory, a graph is a series of vertexes connected by edges. In a directed graph, the edges are connected so that each edge only goes one way. A directed acyclic graph means that the graph is not cyclic, or that it is impossible to start at one point in the graph and traverse the entire graph. Each edge is directed from an earlier edge to a later edge. This is also known as a topological ordering of a graph.

The code optimization is required to produce an efficient target code. These are two important issues that used to be considered while applying the techniques for code optimization.

They are:

* The semantics equivalences of the source program must not be changed.
* The improvement over the program efficiency must be achieved without changing the algorithm.

**Algorithm**

1. Start the program
2. Include all the header files
3. Check for postfix expression and construct the in order DAG representation
4. Print the output
5. Stop the program

**Program**

#include<stdio.h>

#include<string.h>

#include<ctype.h>

#include<conio.h>

void main()

{

struct da

{

int ptr,left,right;

char label;

}dag[25];

int ptr,l,j,change,n=0,i=0,state=1,x,y,k;

char store,\*input1,input[25],var;

clrscr();

for(i=0;i<25;i++)

{

dag[i].ptr=NULL;

dag[i].left=NULL;

dag[i].right=NULL;

dag[i].label=NULL;

}

printf("\n\nENTER THE EXPRESSION\n\n");

scanf("%s",input1);

/\*EX:((a\*b-c))+((b-c)\*d)) like this give with paranthesis.limit

is 25 char ucan change that\*/

for(i=0;i<25;i++)

input[i]=NULL;

l=strlen(input1);

a:

for(i=0;input1[i]!=')';i++);

for(j=i;input1[j]!='(';j--);

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

if(isalpha(input1[x]))

input[n++]=input1[x];

else

if(input1[x]!='0')

store=input1[x];

input[n++]=store;

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

input1[x]='0';

if(input1[0]!='0')goto a;

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

{

dag[i].label=input[i];

dag[i].ptr=i;

if(!isalpha(input[i])&&!isdigit(input[i]))

{

dag[i].right=i-1;

ptr=i;

var=input[i-1];

if(isalpha(var))

ptr=ptr-2;

else

{

ptr=i-1;

b:

if(!isalpha(var)&&!isdigit(var))

{

ptr=dag[ptr].left;

var=input[ptr];

goto b;

}

else

ptr=ptr-1;

}

dag[i].left=ptr;

}

}

printf("\n SYNTAX TREE FOR GIVEN EXPRESSION\n\n");

printf("\n\n PTR \t\t LEFT PTR \t\t RIGHT PTR \t\t LABEL\n\n");

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

{

if((dag[i].label==dag[j].label&&dag[i].left==dag[j].left)&&dag[i].right==dag[j].right)

{

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

{

if(dag[k].left==dag[j].ptr)dag[k].left=dag[i].ptr;

if(dag[k].right==dag[j].ptr)dag[k].right=dag[i].ptr;

}

dag[j].ptr=dag[i].ptr;

}

}

}

printf("\n DAG FOR GIVEN EXPRESSION\n\n");

printf("\n\n PTR \t LEFT PTR \t RIGHT PTR \t LABEL \n\n");

for(i=0;i<n;i++)/\*draw DAG for the following output with

pointer value\*/

printf("\n %dt\t%d\t\t%d\t\t%c\n",dag[i].ptr,dag[i].left,dag[i].right,dag[i].label);

getch();

}for(i=0;i<n;i++)/\* draw the syntax tree for the following

output with pointer value\*/

printf("\n%d\t%d\t%d\t%c\n",dag[i].ptr,dag[i].left,dag[i].right,dag[i].label);

getch();

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

{

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

{

if((dag[i].label==dag[j].label&&dag[i].left==dag[j].left)&&dag[i].right==dag[j].right)

{

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

{

if(dag[k].left==dag[j].ptr)dag[k].left=dag[i].ptr;

if(dag[k].right==dag[j].ptr)dag[k].right=dag[i].ptr;

}

dag[j].ptr=dag[i].ptr;

}

}

}

printf("\n DAG FOR GIVEN EXPRESSION\n\n");

printf("\n\n PTR \t LEFT PTR \t RIGHT PTR \t LABEL \n\n");

for(i=0;i<n;i++)/\*draw DAG for the following output with

pointer value\*/

printf("\n %dt\t%d\t\t%d\t\t%c\n",dag[i].ptr,dag[i].left,dag[i].right,dag[i].label);

getch();

}

**Output**

Enter the expression

((a\*b-c))+((b-c)\*d))

**Syntax tree for given expression**

**PTR             LEFT PTR                RIGHT PTR               LABEL**

0         0        0        a

1       0        0        b

2       0        0        c

3       1        2       -

4       0        3       -

**Dag for given expression**

**PTR     LEFT PTR        RIGHT PTR        LABEL**

0         0        0        a

1       0        0       b

2       0        0       c

3       1        2        -

4       0        3        -

**Result**

Thus the program for implementation of DAG has been successfully executed and output is

verified.