

# Compiler Design Lab

**NAME:** Priyanka Srinivas

**REGISTER NUMBER:** RA1911026010014

**SECTION:** K1

## Exercise 12: Directed Acyclic Graph

**AIM:** Intermediate Code Generation of DAG

### INTRODUCTION:

The Directed Acyclic Graph (DAG) is used to represent the structure of basic blocks, to visualize the flow of values between basic blocks, and to provide optimization techniques in the basic block. To apply an optimization technique to a basic block, a DAG is a three-address code that is generated as the result of an intermediate code generation.

Directed acyclic graphs are a type of data structure and they are used to apply transformations to basic blocks.

The Directed Acyclic Graph (DAG) facilitates the transformation of basic blocks.

DAG is an efficient method for identifying common sub-expressions.

It demonstrates how the statement's computed value is used in subsequent statements.

### ALGORITHM:

1. The leaves of a graph are labelled by a unique identifier and that identifier can be variable names or constants.
2. Interior nodes of the graph are labeled by an operator symbol.
3. Nodes are also given a sequence of identifiers for labels to store the computed value.
4. If y operand is undefined then create node(y).
5. If z operand is undefined then for case(i) create node(z).
6. For case(i), create node(OP) whose right child is node(z) and left child is node(y).
7. For case(ii), check whether there is node(OP) with one child node(y).
8. For case(iii), node n will be node(y).
9. For node(x) delete x from the list of identifiers. Append x to attached identifiers list for the node n found in step 2. Finally set node(x) to n.

### PROGRAM:

```
#include<stdio.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 OF DAG\n\n");

    scanf("%s",str);
    printf("The intermediate code:\t\tExpression\n");
    findopr();
    explore();

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

```

```

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

```

```

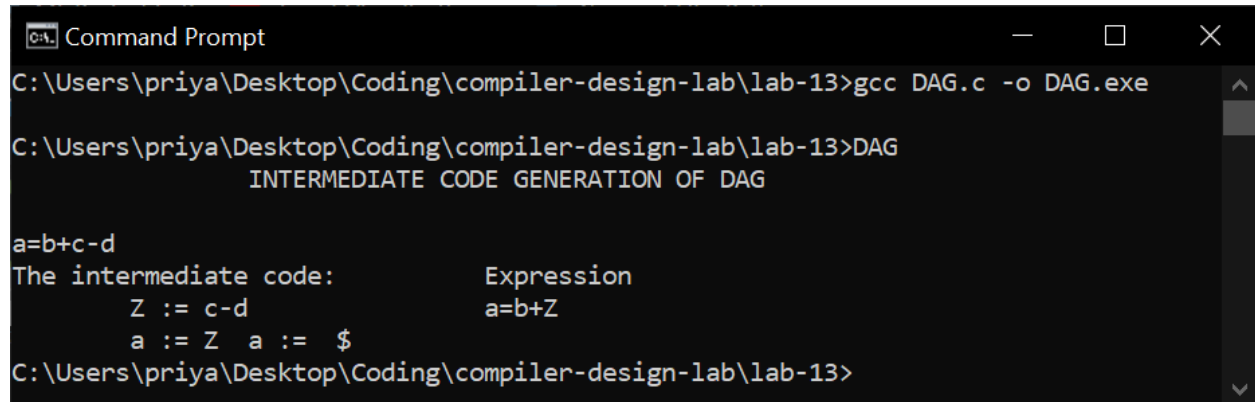
}
void fleft(int x)
{
    int w=0,flag=0;
    x--;
    while(x!= -1 &&str[x]!= '+'
&&str[x]!='*&&str[x]!='='&&str[x]!='\0'&&str[x]!='-'&&str[x]!='/'&&
tr[x]!=':')
    {
        if(str[x]!='$'&& flag==0)
        {
            left[w++]=str[x];
            left[w]='\0';
            str[x]='$';
       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

```

### INPUT:

$a=b+c-d$

### OUTPUT:



```
Command Prompt
C:\Users\priya\Desktop\Coding\compiler-design-lab\lab-13>gcc DAG.c -o DAG.exe
C:\Users\priya\Desktop\Coding\compiler-design-lab\lab-13>DAG
INTERMEDIATE CODE GENERATION OF DAG

a=b+c-d
The intermediate code:      Expression
      Z := c-d              a=b+Z
      a := Z  a := $
C:\Users\priya\Desktop\Coding\compiler-design-lab\lab-13>
```

### RESULT:

Intermediate code generation was performed through DAG.