Date: 03 - 05 - 2021

Expt. No. 11

Implementation of DAG

Aim: To implement the concept of DAG.

Algorithm:

1. Step 1:
    a. If y operand is undefined then create node(y).
    b. If z operand is undefined then for case(i) create node(z).
2. Step 2:
    a. For case(i), create node(OP) whose right child is node(z) and left child is node(y).
    b. For case(ii), check whether there is node(OP) with one child node(y).
    c. For case(iii), node n will be node(y).
3. Output
    a. 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&lt;stdio.h&gt;
#include&lt;string.h&gt;
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];

int main()
```

```
{
        printf(&quot;\t\tINTERMEDIATE CODE GENERATION OF DAG\n\n&quot;);
        scanf(&quot;%s&quot;,str);
        printf(&quot;The intermediate code:\t\tExpression\n&quot;);
        findopr();
        explore();
}

void findopr()
{
        for(i=0;str[i]!=&#39;\0&#39;;i++)
        if(str[i]==&#39;:&#39;)
        {
                k[j].pos=i;
                k[j++].op=&#39;:&#39;;
        }
        for(i=0;str[i]!=&#39;\0&#39;;i++)
                if(str[i]==&#39;/&#39;)
                {
                        k[j].pos=i;
                        k[j++].op=&#39;/&#39;;
                }
        for(i=0;str[i]!=&#39;\0&#39;;i++)
                if(str[i]==&#39;*&#39;)
                {
                        k[j].pos=i;
                        k[j++].op=&#39;*&#39;;
                }
        for(i=0;str[i]!=&#39;\0&#39;;i++)
                if(str[i]==&#39;+&#39;)
                {
```

```
                k[j].pos=i;
                k[j++].op=&#39;+&#39;;
            }
        for(i=0;str[i]!=&#39;\0&#39;;i++)
            if(str[i]==&#39;-&#39;)
            {
                k[j].pos=i;
                k[j++].op=&#39;-&#39;;
            }
        }

void explore()
{
        i=1;
        while(k[i].op!=&#39;\0&#39;)
        {
            fleft(k[i].pos);
            fright(k[i].pos);
            str[k[i].pos]=tmpch--;
            printf(&quot;\t%c := %s%c%s\t\t&quot;,str[k[i].pos],left,k[i].op,right);
            for(j=0;j &lt;strlen(str);j++)
                if(str[j]!=&#39;$&#39;)
                    printf(&quot;%c&quot;,str[j]);
            printf(&quot;\n&quot;);
            I++;
        }
        fright(-1);
        if(no==0)
        {
            fleft(strlen(str));
            printf(&quot;\t%s := %s&quot;,right,left);
```

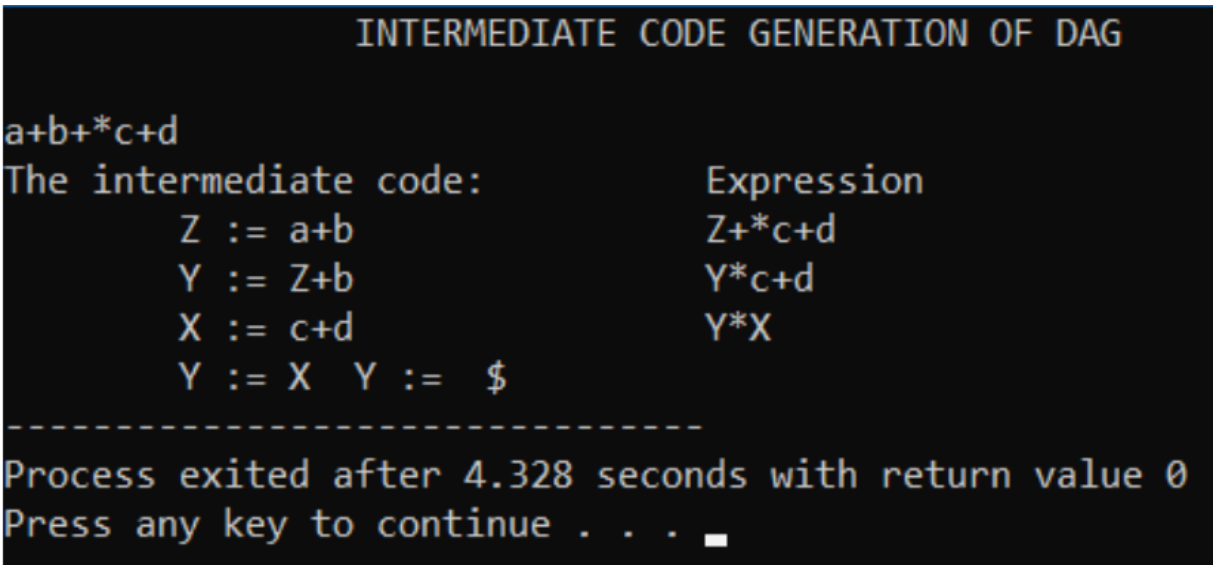
```
        }
        printf(&quot;\t%s := %c&quot;,right,str[k[--i].pos]);
}

void fleft(int x)
{
        int w=0,flag=0;
        X--;
        while(x!= -1
&amp;&amp;str[x]!=&#39;+&#39;&amp;&amp;str[x]!=&#39;*&#39;&amp;
&amp;str[x]!=&#39;=&#39;&amp;&amp;str[x]!=&#39;\0&#39;&amp;&amp;str[x]!=&#39;-&#
39; &amp;&amp;str[x]!=&#39;/&#39;&amp;&amp;str[x]!=&#39;:&#39;)
        {
                if(str[x]!=&#39;$&#39;&amp;&amp; flag==0)
                {
                        left[w++]=str[x];
                        left[w]=&#39;\0&#39;;
                        str[x]=&#39;$&#39;;
                        flag=1;
                }
                x--;
        }
}

void fright(int x)
{
        int w=0,flag=0;
        x++;
        while(x!= -1 &amp;&amp; str[x]!=&#39;+&#39;&amp;&amp;str[x]!=&#39;*&#39;
&amp;&amp;str[x]! =&#39;\0&#39;&amp;&amp;str[x]!=&#39;=&#39;&amp;&amp;str[x]!
=&#39;:&#39;&amp;&amp;str[x]!=&#39;-&#39;&amp;&amp;str[x]!=&#39;/&#39;)
        {
                if(str[x]!=&#39;$&#39;&amp;&amp; flag==0)
```

```
                {
                        right[w++]=str[x];
                        right[w]=&#39;\0&#39;;
                        str[x]=&#39;$&#39;;
                        flag=1;
                }
                x++;
        }
}
```

Input /Output:



Result: The code was successfully implemented and verified.