# Course Project - Labelling algorithm in association with code optimization using 3 Address code and DAG.


Name: Atharva Dixit Div: C
Roll No. 38 GR No. 1710321

Name: Aditya Holkar Div: C
Roll No. 57 GR No. 1710349

Name: Upendra Kadre Div: C
Roll No. 69 GR No. 1710505

Name: Swapnil Katti Div: C
Roll No. 72 GR No. 1710322

# What is Labelling Algorithm?

The compiler uses Labelling Algorithm in the code generation phase. The algorithm is used to find out how many registers will be required by a program to complete its execution. The labelling algorithms works in bottom-up fashion. The algorithm will label the child nodes first and then it labels the interior nodes.

**Rules of labeling algorithm are:**

- If ‘n’ is a leaf node –
    **a.** If ‘n’ is a left child then its value is 1.
    **b.** If ‘n’ is a right child then its value is 0.
- If ‘n’ is an interior node –
  Let's assume L1 and L2 are left and right child of the interior node respectively.
    **a.** If L1 == L2 then value of ‘n’ is L1 + 1 or L2 + 1
    **b.** If L1 != L2 then value of ‘n’ is MAX(L1, L2)

# Project:

The project consists of code written for the labelling algorithm which takes an infix expression as an input. This input is then converted into a postfix expression. The postfix expression is converted into a 3 address code. Finally, after generating the 3 address code, a tree is constructed and labelled using the labelling algorithm.
The output of the project consists of the generated tree and the number of registers required for the execution of input expression.

# Code:

```
#include<iostream>
```

```
#include<stack>
#include<queue>
#define COUNT 10
using namespace std;

stack <char> stk;
stack <char> stk2;

class node{
public:

char chr;
node *left,*right;

int val,par[5],ct=0;
};


int checkalp(char c){
if(c>='a' && c<='z')
```

```
return 0;
else return 1;
}

int checkop(char c){
if(c=='+' || c=='-' || c=='*' || c=='/')
return 0;
else return 1;
}

void print2DUtil(node *root, int space)
{
    // Base case
    if (root == NULL)
        return;

    // Increase distance between levels
    space += COUNT;

    // Process right child first
```

```
    print2DUtil(root->right, space);

    // Print current node after space
    // count
    cout<<endl;
    for (int i = COUNT; i < space; i++)
        cout<<" ";
    cout<<root->chr;

    if( root->ct >0)
{
cout<<"(";

for(int l=0;l < root->ct;l++)
{

cout<<"t"<<root->par[l]<<" ";

}
cout<<")";
```

```
}
cout<<endl;
    // Process left child
    print2DUtil(root->left, space);
}

// Wrapper over print2DUtil()
void print2D(node *root)
{
    // Pass initial space count as 0
    print2DUtil(root, 0);
}

int main(){

char str[100],post[100];
char add3[10][100];
int h;
int q=0,t=0;
int j=0;
```

```
cout<<"enter exp:\n";
cin>>str;

for(int i=0;str[i]!='\0';i++)
{
if(checkalp(str[i])==0)
{
post[j]=str[i];
j++;
}
else if(checkop(str[i])==0)
{
if(str[i]=='+' || str[i]=='-')
{
if(stk.top()=='*' || stk.top()=='/')
{
post[j]=stk.top();
stk.pop();
stk.push(str[i]);
j++;
```

```
}
else
{
stk.push(str[i]);
}
}
else
{
stk.push(str[i]);
}
}
else if(str[i]==')')
{
while(stk.top()!='(')
{
post[j]=stk.top();
stk.pop();
j++;
}
stk.pop();
```

```
}
else if(str[i]=='(')
{
stk.push(str[i]);
}



}


post[j]='\0';

cout<<"postfix exp:"<<endl<<post;

char x;
for(int i=0;post[i]!='\0';i++)
{
if(checkalp(post[i])==0)
{
stk2.push(post[i]);
```

```
}
else if(checkop(post[i])==0)
{
x=stk2.top();
stk2.pop();
add3[q][t]=stk2.top();
stk2.pop();
t++;
add3[q][t]=post[i];
t++;
add3[q][t]=x;
t++;
add3[q][t]='\0';


stk2.push(q+48);
q++;
t=0;
}
```

```
}

add3[q][t]='\0';

cout<<endl<<"3 address code:"<<endl;

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

cout<<'t'<<i<<'='<<add3[i]<<endl;
}



//dag

node *aray[q];

for(int i=0;i<q;i++)
{
```

```
aray[i]=new node();
aray[i]->chr=add3[i][1];
aray[i]->par[0]=i;
aray[i]->ct=1;

if(checkalp(add3[i][0])==0)
{
aray[i]->left= new node();

aray[i]->left->chr=add3[i][0];
aray[i]->left->right=NULL;
aray[i]->left->left=NULL;
aray[i]->left->val=1;
aray[i]->val=1;
}
else
{
int a=add3[i][0]-'0';
aray[i]->left=aray[a];
aray[a]->left->par[0]=a;
```

```
aray[i]->left->ct=1;


}


if(checkalp(add3[i][2])==0)
{
aray[i]->right= new node();


aray[i]->right->chr=add3[i][2];


aray[i]->right->right=NULL;
aray[i]->right->left=NULL;
aray[i]->right->val=0;
}
else
{
int a=add3[i][2]-'0';
aray[i]->right=aray[a];
aray[i]->right->par[0]=a;
aray[i]->right->ct=1;
```

```
if(aray[i]->left->val==aray[i]->right->val)
{
aray[i]->val=aray[i]->left->val +1;
}
else
{
if(aray[i]->left->val > aray[i]->right->val)
{
aray[i]->val=aray[i]->left->val;
}
else
{
aray[i]->val=aray[i]->right->val;


}
}
}
```

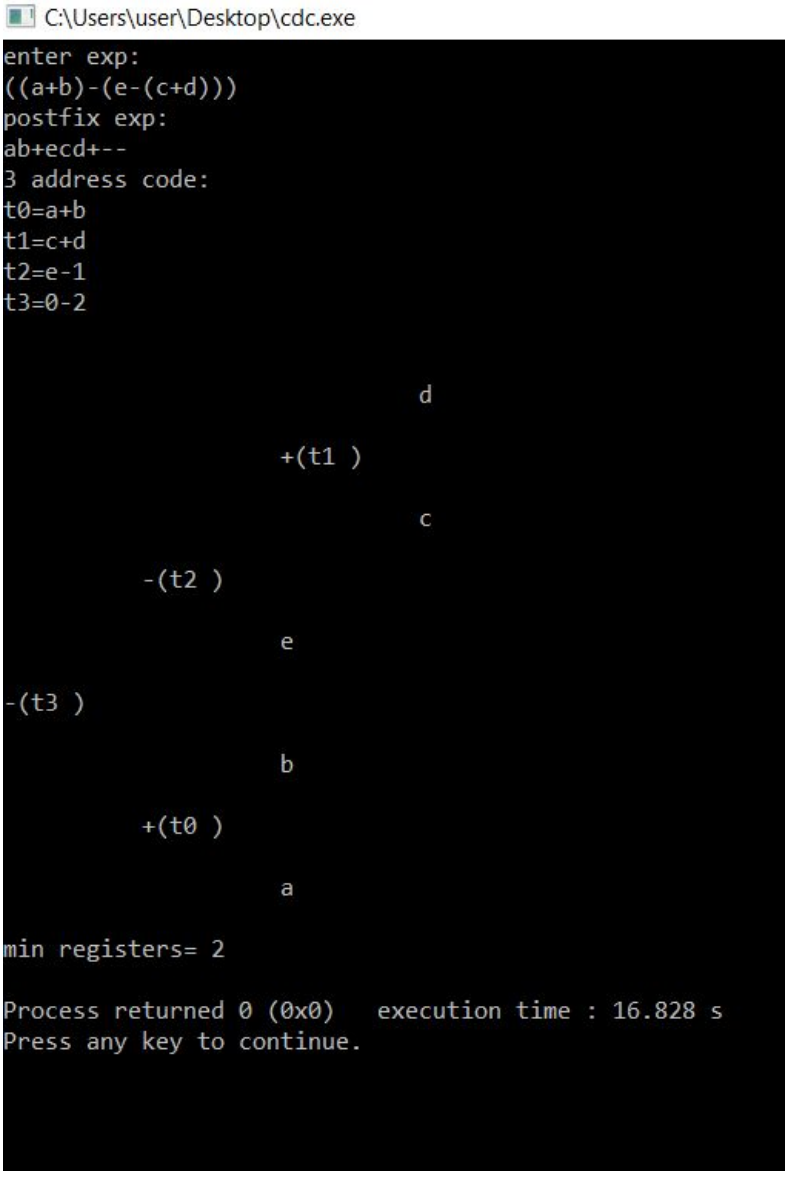
```
}
cout<<endl;



for(int i=0;i<q;i++)
{
for(int j=i+1;j<q;j++)
{
if(aray[i]->chr==aray[j]->chr &&
aray[i]->left->chr==aray[j]->left->chr &&
aray[i]->right->chr==aray[j]->right->chr  )
{
aray[j]=aray[i];
aray[i]->par[aray[i]->ct]=j;
aray[i]->ct =aray[i]->ct +1;


}
}
}


print2D(aray[q-1]);
```

```
/*
int cnt=1,cnt2=0,spa=q*20;
queue <node*> qu;
node *p;
qu.push(aray[q-1]);

while(!qu.empty()){
for(int i=0;i!=cnt;i++)
{

p=qu.front();



for(int d=0; d< spa;d++)
cout<<" ";


if(p!=NULL)
{
```

```
//space

cout<<p->chr;

if( p->ct >0)
{
cout<<"(";

for(int l=0;l < p->ct;l++)
{

cout<<"t"<<p->par[l]<<" ";

}
cout<<")";
}

qu.push(p->left);
cnt2++;
```

```
qu.push(p->right);
cnt2++;
}
qu.pop();



}
cout<<endl;
cout<<endl;

spa=spa/2;
cnt=cnt2;
cnt2=0;
}
*/
cout<<endl<<"min registers= "<<aray[q-1]->val<<endl;

return 0;
}
```

## Output:


C:\Users\user\Desktop\cdc.exe
enter exp:
((a+b)-(e-(c+d)))
postfix exp:
ab+ecd+--
3 address code:
t0=a+b
t1=c+d
t2=e-1
t3=0-2
d
+(t1 )
c
-(t2 )
e
-(t3 )
b
+(t0 )
a
min registers= 2
Process returned 0 (0x0) execution time : 16.828 s
Press any key to continue.