ASSIGNMENT-9

Considering the following grammar G

S -> E$

E -> E+E

E -> E*E

E -> (E)

E -> I

I -> I digit

I -> digit

1. Design a syntax directed translator for G.
2. Design a syntax directed translator for G that will perform infix-postfix translation.
3. Design a mechanism to generate three address codes for a statement/expression belonging to L(G).

Solution:

```
#include<bits/stdc++.h>

#include<string>


using namespace std;


char stac[20],val1[20],sym[20];

int val[20];

int top1=-1,top2=-1,top3=-1;

string input;


void print_stac(){

        for(int i=0;i<=top1;i++){

                cout<<stac[i];

        }
```

```
}

void print_val(){
        for(int i=0;i<=top2;i++){
                cout<<val[i]<<" ";
        }
}

void print_val1(){
        for(int i=0;i<=top2;i++){
                cout<<val1[i];
        }
}

void sdt(){
        stac[0] = '$';top1=0;
        input[input.length()]='$';
        cout<<"Stack\tValue\n---------------\n";
        for(int i=0;i<input.length();i++){
                if(input[i]>='0' && input[i]<='9'){
                        stac[top1+1] = 'd';
                        top1+=1;
                        val[top2+1] = int(input[i])-48;
                        top2+=1;
                        print_stac();
                        cout<<"\t";
                        print_val();
                        cout<<endl;
                }
                else{
                        stac[top1+1] = input[i];
                        top1+=1;
                        print_stac();
                        cout<<"\t";
                        print_val();
                        cout<<endl;
                }
                if(stac[top1]=='d' && stac[top1-1]=='I'){
                        stac[top1-1] = 'I';
                        top1-=1;
                        val[top2-1] = 10*val[top2-1] + val[top2];
                        top2-=1;
                        print_stac();
                        cout<<"\t";
                        print_val();
                        cout<<endl;
                }
                if(stac[top1]=='d'){
                        stac[top1]='I';

                        print_stac();
                        cout<<"\t";
                        print_val();
                        cout<<endl;
                }
                if(stac[top1]=='I' && (input[i+1]<'0' || input[i+1]>'9')){
                        stac[top1] = 'E';
                        print_stac();
                        cout<<"\t";
                        print_val();
                        cout<<endl;
                }
```

```
if(stac[top1]==')' && stac[top1-1]=='E' && stac[top1-2]=='('){
        stac[top1-2]='E';
        top1-=2;
        print_stac();
        cout<<"\t";
        print_val();
        cout<<endl;
}
if(stac[top1]=='E' && stac[top1-1]=='+' && stac[top1-2]=='E'){
        stac[top1-2]='E';
        top1-=2;
        val[top2-1] = val[top2]+val[top2-1];
        top2-=1;
        print_stac();
        cout<<"\t";
        print_val();
        cout<<endl;
}
if(stac[top1]=='E' && stac[top1-1]=='*' && stac[top1-2]=='E'){
        stac[top1-2]='E';
        top1-=2;
        val[top2-1] = val[top2]*val[top2-1];
        top2-=1;
        print_stac();
        cout<<"\t";
        print_val();
        cout<<endl;
}
if(stac[top1]=='$' && stac[top1-1]=='E'){
        stac[top1-1]='S';
        top1-=1;
```

```
                print_stac();
                cout<<"\t";
                print_val();
                cout<<endl;
            }

        }
        cout<<val[top2]<<endl;
}

void convert(){
        stac[0] = '$';top1=0;top2=-1;
        input[input.length()]='$';
        cout<<"Stack\tPost-fix\n---------------\n";
        for(int i=0;i<input.length();i++){
            if(input[i]>='0' && input[i]<='9'){
                stac[top1+1] = 'd';
                top1+=1;
                val1[top2+1] = input[i];
                top2+=1;
                print_stac();
                cout<<"\t";
                print_val1();
                cout<<endl;
            }
            else{
                stac[top1+1] = input[i];
                if(input[i]=='*' || input[i]=='+'){
                    sym[top3+1]=input[i];
                    top3+=1;
                }
        top1+=1;
        print_stac();
        cout<<"\t";
        print_val1();
        cout<<endl;
}
if(stac[top1]=='d' && stac[top1-1]=='I'){
        stac[top1-1] = 'I';
        top1-=1;
        // val1[top2-1] = 10*val[top2-1] + val[top2];
        // top2-=1;
        print_stac();
        cout<<"\t";
        print_val1();
        cout<<endl;
}
if(stac[top1]=='d'){
        stac[top1]='I';

        print_stac();
        cout<<"\t";
        print_val1();
        cout<<endl;
}
if(stac[top1]=='I' && (input[i+1]<'0' || input[i+1]>'9')){
        stac[top1] = 'E';
        print_stac();
        cout<<"\t";
        print_val1();
        cout<<endl;
}
```

```
if(stac[top1]==')' && stac[top1-1]=='E' && stac[top1-2]=='('){
        stac[top1-2]='E';
        top1-=2;
        val1[top2+1]=sym[top3];
        top3-=1;top2+=1;
        print_stac();
        cout<<"\t";
        print_val1();
        cout<<endl;
}
if(stac[top1]=='E' && stac[top1-1]=='+' && stac[top1-2]=='E'){
        stac[top1-2]='E';
        top1-=2;
        // val1[top2+1] = '+';
        // top2+=1;
        print_stac();
        cout<<"\t";
        print_val1();
        cout<<endl;
}
if(stac[top1]=='E' && stac[top1-1]=='*' && stac[top1-2]=='E'){
        stac[top1-2]='E';
        top1-=2;
        // val1[top2+1] = '*';
        // top2+=1;
        print_stac();
        cout<<"\t";
        print_val1();
        cout<<endl;
}
if(stac[top1]=='$' && stac[top1-1]=='E'){
```

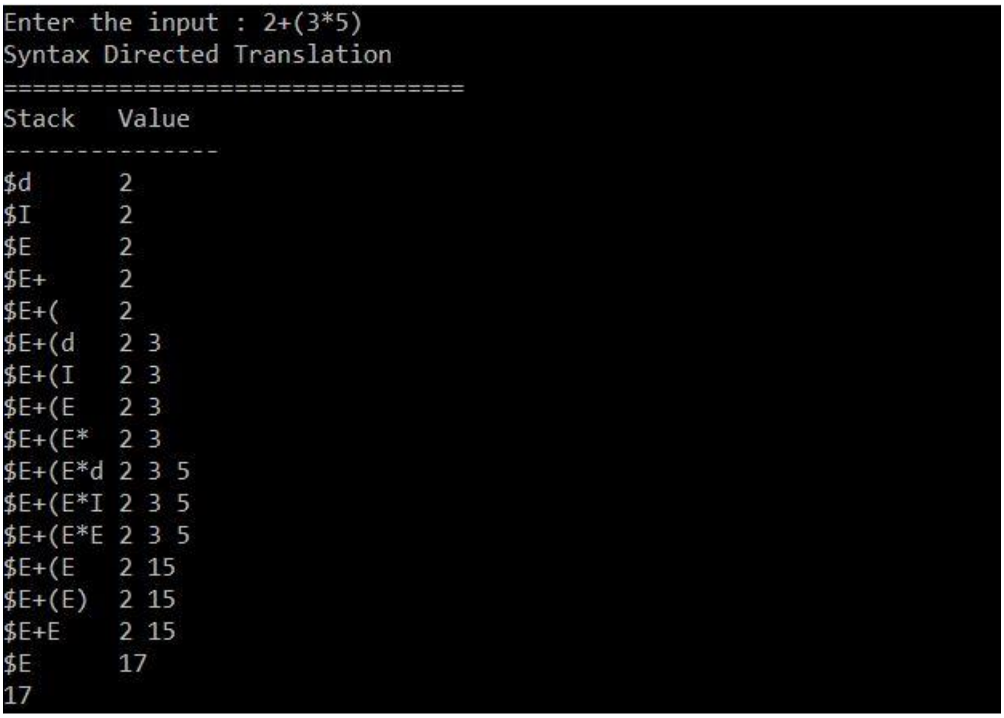
```
            stac[top1-1]='S';
            top1-=1;
            print_stac();
            cout<<"\t";
            print_val1();
            cout<<endl;
        }

    }
    print_stac();
    cout<<"\t";
    print_val1();
    while(top3!=-1){
        cout<<sym[top3];
        top3-=1;
    }
    cout<<endl;

}

void threeAddressCode(){
    stac[0] = '$';top1=0;top2=-1;
    int x=1;
    vector<vector<string> > v;
    input[input.length()]='$';
    cout<<"Stack\tPlace\tGenerated Code\n-----------------------------\n";
    for(int i=0;i<input.length();i++){
        if(input[i]>='0' && input[i]<='9'){
            stac[top1+1] = 'd';
            top1+=1;
            val[top2+1] = int(input[i])-48;
```

```
		top2+=1;
		print_stac();
		cout<<"\t";
		print_val();
		cout<<endl;
	}
	else{
		stac[top1+1] = input[i];
		top1+=1;
		print_stac();
		cout<<"\t";
		print_val();
		cout<<endl;
	}
	if(stac[top1]=='d' && stac[top1-1]=='l'){
		stac[top1-1] = 'l';
		top1-=1;
		val[top2-1] = 10*val[top2-1] + val[top2];
		top2-=1;
		print_stac();
		cout<<"\t";
		print_val();
		cout<<endl;
	}
	if(stac[top1]=='d'){
		stac[top1]='l';

		print_stac();
		cout<<"\t";
		print_val();
		cout<<endl;
```

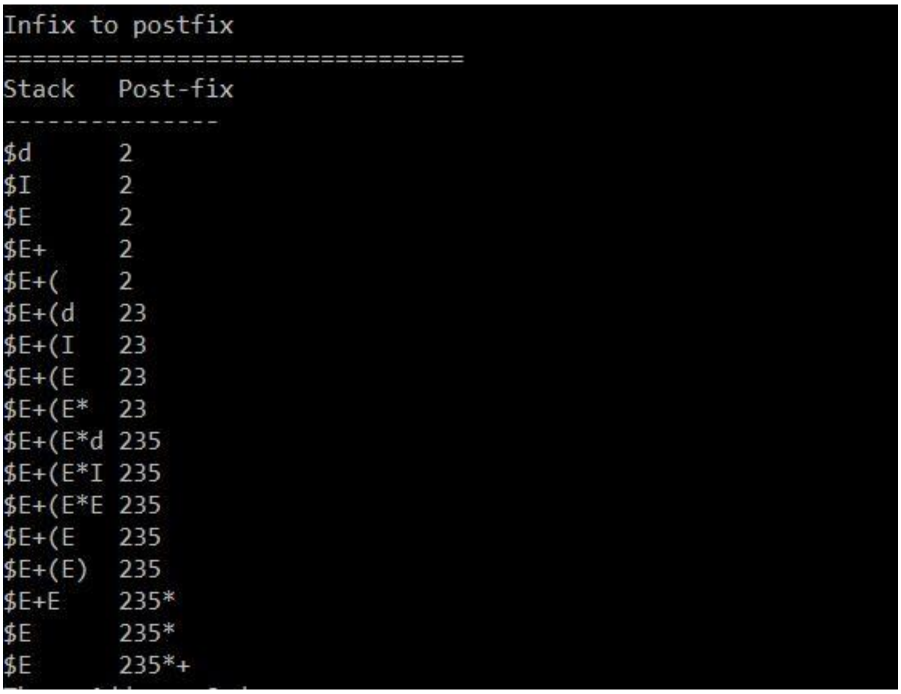
```
}
if(stac[top1]=='I' && (input[i+1]<'0' || input[i+1]>'9')){
    stac[top1] = 'E';
    print_stac();
    cout<<"\t";
    print_val();
    cout<<endl;
}
if(stac[top1]==')' && stac[top1-1]=='E' && stac[top1-2]=='('){
    stac[top1-2]='E';
    top1-=2;
    print_stac();
    cout<<"\t";
    print_val();
    cout<<endl;
}
if(stac[top1]=='E' && stac[top1-1]=='+' && stac[top1-2]=='E'){
    print_stac();
    cout<<"\t";
    print_val();
    if(x>1)
        cout<<"\tT"<<x<<" := "<<val[top2-1]<<" + T"<<x-1;
    else
        cout<<"\tT"<<x<<" := "<<val[top2-1]<<" + "<<val[top2];
    x++;
    cout<<endl;
    stac[top1-2]='E';
    top1-=2;
    val[top2-1] = val[top2]+val[top2-1];
    top2-=1;
    print_stac();
```

```
        cout<<"\t";
        print_val();
        cout<<endl;
}
if(stac[top1]=='E' && stac[top1-1]=='*' && stac[top1-2]=='E'){
        print_stac();
        cout<<"\t";
        print_val();
        if(x>1)
                cout<<"\tT"<<x<<" := "<<val[top2-1]<<" * T"<<x-1;
        else
                cout<<"\tT"<<x<<" := "<<val[top2-1]<<" * "<<val[top2];
        x++;
        cout<<endl;
        stac[top1-2]='E';
        top1-=2;
        val[top2-1] = val[top2]*val[top2-1];
        top2-=1;
        print_stac();
        cout<<"\t";
        print_val();

        cout<<endl;
}
if(stac[top1]=='$' && stac[top1-1]=='E'){
        stac[top1-1]='S';
        top1-=1;
        print_stac();
        cout<<"\t";
        print_val();
        cout<<endl;
```

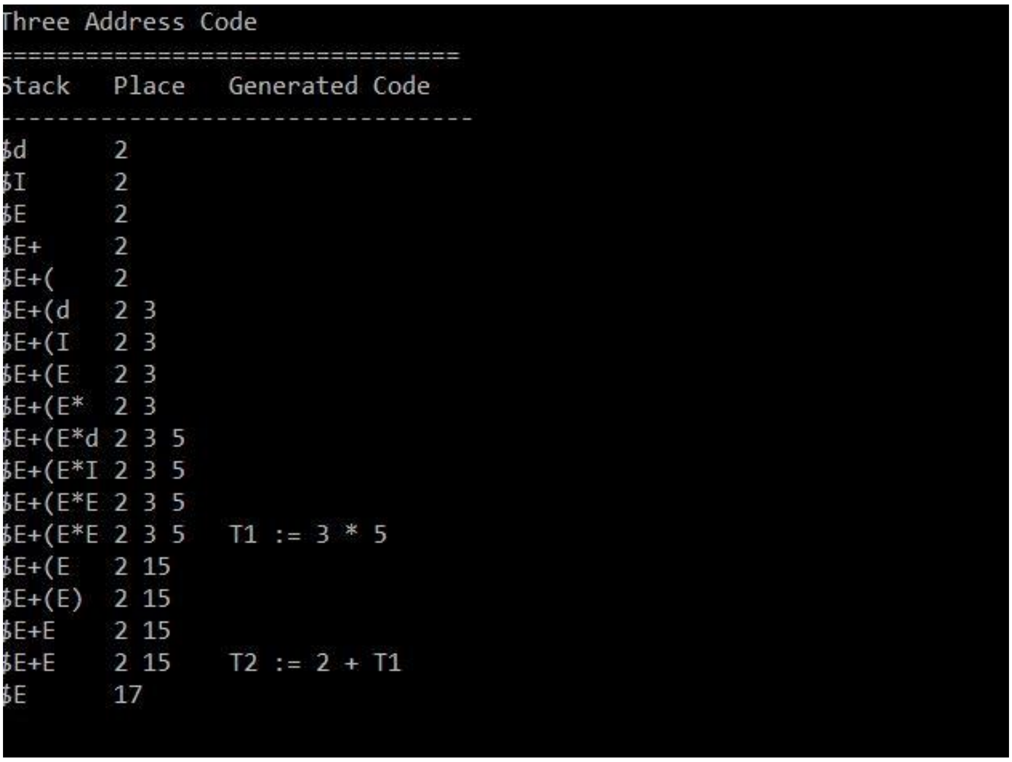
```
                        }


            }
}

int main(){
            //get_gram();
            cout<<"Enter the input : ";
            cin>>input;
            cout<<"Syntax Directed Translation\n================================\n";
            sdt();
            cout<<"Infix to postfix\n================================\n";
            convert();
            cout<<"Three Address Code\n================================\n";
            threeAddressCode();
            return 0;
}
```



```
Enter the input : 2+(3*5)
Syntax Directed Translation
================================
Stack   Value
---------------
$d      2
$I      2
$E      2
$E+     2
$E+(    2
$E+(d   2 3
$E+(I   2 3
$E+(E   2 3
$E+(E*  2 3
$E+(E*d 2 3 5
$E+(E*I 2 3 5
$E+(E*E 2 3 5
$E+(E   2 15
$E+(E)  2 15
$E+E    2 15
$E      17
17
```



```
Infix to postfix
================================
Stack    Post-fix
---------------
$d       2
$I       2
$E       2
$E+      2
$E+(     2
$E+(d    23
$E+(I    23
$E+(E    23
$E+(E*   23
$E+(E*d  235
$E+(E*I  235
$E+(E*E  235
$E+(E    235
$E+(E)   235
$E+E     235*
$E       235*
$E       235*+
```



```
Three Address Code
================================
Stack    Place    Generated Code
--------------------------------
$d       2
$I       2
$E       2
$E+      2
$E+(     2
$E+(d    2 3
$E+(I    2 3
$E+(E    2 3
$E+(E*   2 3
$E+(E*d  2 3 5
$E+(E*I  2 3 5
$E+(E*E  2 3 5
$E+(E*E  2 3 5    T1 := 3 * 5
$E+(E    2 15
$E+(E)   2 15
$E+E     2 15
$E+E     2 15     T2 := 2 + T1
$E       17
```

ASSIGNMENT-10

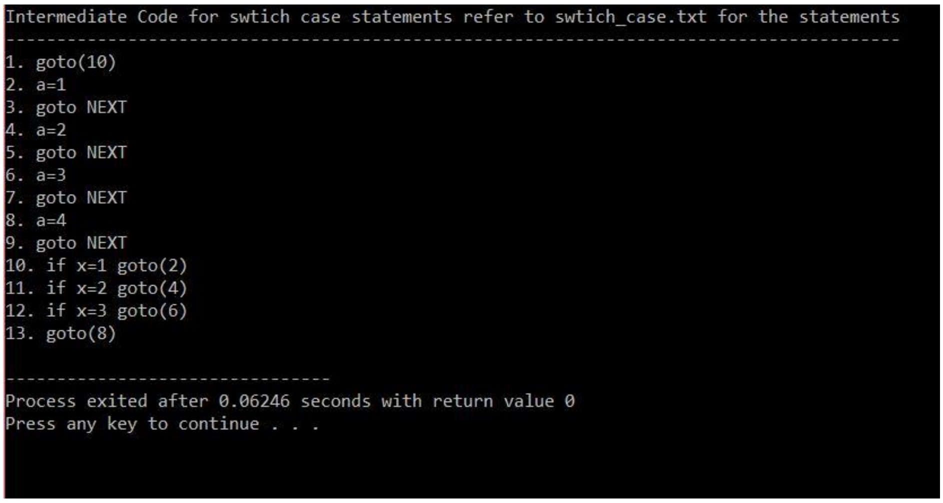
1.Design a syntax directed translator that will generate intermediate code for switch-case construct in C language.

Solution:

```
#include<bits/stdc++.h>

using namespace std;

string output[100];
int sum=0,top=-1;
int stac[50];

void threeAddressCode(){
        int cou=1,it=1;
        for(int i=0;i<sum;i++){
                if(output[i]=="switch"){
                        cout<<it<<". ";
                        it++;
                        cout<<"goto(10)"<<"\n";
                }
                else if(output[i]=="case"){
                        cout<<it<<". ";
                        it++;
                        cout<<output[i+3]<<"\n";
                        stac[top+1]=it-1;
                        top+=1;
                        cout<<it<<". ";
                        it++;
                        cout<<"goto NEXT"<<"\n";
                cou++;
        }
        else if(output[i]=="default"){
                cout<<it<<". ";
                it++;
                cout<<output[i+2]<<"\n";
                stac[top+1]=it-1;
                top+=1;
                cout<<it<<". ";
                it++;
                cout<<"goto NEXT"<<"\n";
                cou++;
                break;
        }
}

int check=0;
for(int i=0;i<sum;i++){
        if(output[i]=="case"){
                cout<<it<<". ";
                it++;
                cout<<"if x="<<output[i+1]<<" goto("<<stac[check]<<")"<<"\n";
                check+=1;
        }
        else if(output[i]=="default"){
                cout<<it<<". ";
                it++;
                cout<<"goto("<<stac[check]<<")"<<"\n";
                check+=1;
        }
}
```

```
}


int main(){
	ifstream file;
	file.open("switch_case.txt");
	if(file.is_open())
	{
		while(!file.eof())
		{
			file>>output[sum];
			sum++;
		}
	}
	file.close();


	cout<<"Intermediate Code for swtich case statements refer to swtich_case.txt for the statements\n--------------------------------------------------------------------------------------------\n";
	threeAddressCode();


	return 0;
}
```

Output:



```
Intermediate Code for swtich case statements refer to swtich_case.txt for the statements
----------------------------------------------------------------------------------------
1. goto(10)
2. a=1
3. goto NEXT
4. a=2
5. goto NEXT
6. a=3
7. goto NEXT
8. a=4
9. goto NEXT
10. if x=1 goto(2)
11. if x=2 goto(4)
12. if x=3 goto(6)
13. goto(8)

--------------------------------
Process exited after 0.06246 seconds with return value 0
Press any key to continue . . .
```

ASSIGNMENT-11

1.Implementation of the labelling algorithm to generate assembly language code from the labelled tree of intermediate code.

Solution:

```
#include<stdlib.h>
#include<iostream>
using namespace std;

/* We will implement DAG as Strictly Binary Tree where each node has zero or two children */

struct bin_tree
{
char data;
int label;
struct bin_tree *right, *left;
};
typedef bin_tree node;

class dag
{
private:
/* R is stack for storing registers */
int R[10];
int top;

/* op will be used for opcode name w.r.t. arithmetic operator e.g. ADD for + */
char *op;

public:
```

```
void initializestack(node *root)
{
/* value of top = index of topmost element of stack R = label of Root of tree(DAG) minus one */
   top=root->label - 1;

   /* Allocating Stack Registers */
   int temp=top;
   for(int i=0;i<=top;i++)
      {
        R[i]=temp;
        temp--;
      }
}

/* insertnode() and insert() functions are for adding nodes to tree(DAG) */

void insertnode(node **tree,char val)
{
node *temp = NULL;

if(!(*tree))
   {
      temp = (node *)malloc(sizeof(node));
      temp->left = temp->right = NULL;
      temp->data = val;
      temp->label=-1;
      *tree = temp;
   }
}

void insert(node **tree,char val)
```

```
{
    char l,r;
    int numofchildren;

    insertnode(tree, val);

    cout << "\nEnter number of children of " << val <<" :";
    cin >> numofchildren;

  if(numofchildren==2)
   {
    cout << "\nEnter Left Child of " << val <<" :";
    cin >> l;
    insertnode(&(*tree)->left,l);

    cout << "\nEnter Right Child of " << val <<" :";
    cin >> r;
    insertnode(&(*tree)->right,r);

    insert(&(*tree)->left,l);
    insert(&(*tree)->right,r);
   }
}

/* findleafnodelabel() will find out the label of leaf nodes of tree(DAG) */

void findleafnodelabel(node *tree,int val)
{

if(tree->left != NULL && tree->right !=NULL)
{
```

```
findleafnodelabel(tree->left,1);
findleafnodelabel(tree->right,0);
}

else
{
tree->label=val;
}

}

/* findinteriornodelabel() will find out the label of interior nodes of tree(DAG) */

void findinteriornodelabel(node *tree)
{
if(tree->left->label==-1)
{
findinteriornodelabel(tree->left);
}

else if(tree->right->label==-1)
{
findinteriornodelabel(tree->right);
}

else
{

if(tree->left != NULL && tree->right !=NULL)
{
```

```
if(tree->left->label == tree->right->label)
{

tree->label=(tree->left->label)+1;
}

else
{

if(tree->left->label > tree->right->label)
{
tree->label=tree->left->label;
}
else
{
tree->label=tree->right->label;
}

}
}
}
}

/* function print_inorder() will print inorder of nodes. Here we are also printing label of each node
of tree(DAG) */

void print_inorder(node * tree)
{
   if (tree)
   {
      print_inorder(tree->left);
```

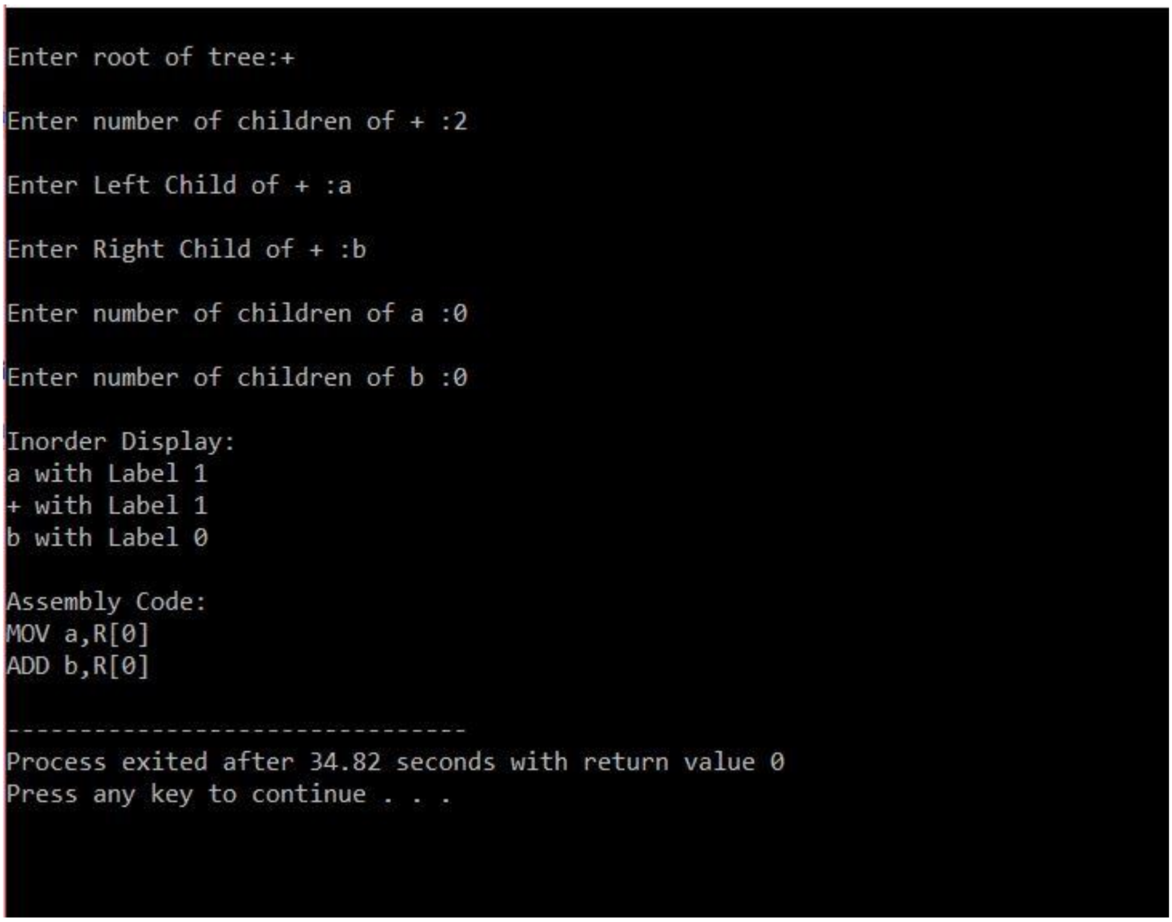
```
    cout << tree->data <<" with Label "<< tree->label << "\n";
    print_inorder(tree->right);
  }
}

/* function swap() will swap the top and second top elements of Register stack R */

void swap()
{
int temp;
temp=R[0];
R[0]=R[1];
R[1]=temp;
}

/* function pop() will remove and return topmost element of stack */

int pop()
{
int temp=R[top];
top--;
return temp;
}

/* function push() will increment top by one and will insert element at top position of Register stack
*/

void push(int temp)
{
top++;
R[top]=temp;
```

```
}

/* nameofoperation() will return opcode w.r.t. arithmetic operator */

void  nameofoperation(char temp)
{
switch(temp)
{
case '+': op =(char *)"ADD"; break;
case '-': op =(char *)"SUB"; break;
case '*': op =(char *)"MUL"; break;
case '/': op =(char *)"DIV"; break;
}
}

/* gencode() will generate Assembly code w.r.t. labels of tree(DAG) */

void gencode(node * tree)
{
if(tree->left != NULL && tree->right != NULL)
{
if(tree->left->label == 1 && tree->right->label == 0 && tree->left->left==NULL && tree->left->right==NULL && tree->right->left==NULL && tree->right->right==NULL)
{
cout << "MOV "<< tree->left->data << "," << "R[" << R[top] << "]\n";
nameofoperation(tree->data);
cout << op << " " << tree->right->data << ",R[" << R[top] << "]\n";
}

else if(tree->left->label >= 1 && tree->right->label == 0)
{
```

```
gencode(tree->left);

nameofoperation(tree->data);

cout << op << " " << tree->right->data << ",R[" << R[top] << "]\n";

}


else if(tree->left->label < tree->right->label)

{

int temp;

swap();

gencode(tree->right);

temp=pop();

gencode(tree->left);

push(temp);

swap();

nameofoperation(tree->data);

cout << op << " " << "R[" << R[top-1] <<"],R[" << R[top] << "]\n";

}


else if(tree->left->label >= tree->right->label)

{

int temp;

gencode(tree->left);

temp=pop();

gencode(tree->right);

push(temp);

nameofoperation(tree->data);

cout << op << " " << "R[" << R[top-1] << "],R[" << R[top] <<"]\n";

}


}
```

```
else if(tree->left == NULL && tree->right == NULL && tree->label == 1)
{
cout << "MOV " << tree->data << ",R[" << R[top] << "]\n";
}

}

/* deltree() will free the memory allocated for tree(DAG) */

void deltree(node * tree)
{
   if (tree)
   {
      deltree(tree->left);
      deltree(tree->right);
      free(tree);
   }
}

};

/* Program execution will start from main() function */

int main()
{
   node *root;
   root = NULL;
   node *tmp;
   char val;
   int i,temp;
```

```
    dag d;

    /* Inserting nodes into tree(DAG) */

    cout << "\nEnter root of tree:";
    cin >> val;

    d.insert(&root,val);

    /* Finding Labels of Leaf nodes */
    d.findleafnodelabel(root,1);

    /* Finding Labels of Interior nodes */
    while(root->label == -1)
      d.findinteriornodelabel(root);
    /* Initializing Stack contents and top variable */
    d.initializestack(root);

    /* Printing inorder of nodes of tree(DAG) */
    cout << "\nInorder Display:\n";
    d.print_inorder(root);

    /* Printing assembly code w.r.t. labels of tree(DAG) */
    cout << "\nAssembly Code:\n";
    d.gencode(root);

    /* Deleting all nodes of tree */
    d.deltree(root);

    return 0;
  }
```



```
Enter root of tree:+

Enter number of children of + :2

Enter Left Child of + :a

Enter Right Child of + :b

Enter number of children of a :0

Enter number of children of b :0

Inorder Display:
a with Label 1
+ with Label 1
b with Label 0

Assembly Code:
MOV a,R[0]
ADD b,R[0]

--------------------------------
Process exited after 34.82 seconds with return value 0
Press any key to continue . . .
```