**ASSIGNMENT NO.**

NAME: SHRUTI DILIP BHUJANGE

CLASS: BE COMP-1 ROLL NO.: 402006

PROGRAM:

#include <iostream>

#include <string.h>

using namespace std;

class node

{

char key;

node \*left;

node \*right;

int label, flag, visited;

friend class tree;

public:

node()

{

key='0';

left=NULL;

right=NULL;

label=0;

flag=1;

visited=0;

}

node(char a)

{

key=a;

left=NULL;

right=NULL;

label=0;

flag=1;

visited=0;

}

void getdata(char a, node\* l, node\* r)

{

key=a;

left=l;

right=r;

}

};

class reg

{

int number;

int flag;

friend class tree;

public:

reg()

{

number=0;

flag=1;

}

reg(int number, int flag)

{

this->number=number;

this->flag=flag;

}

};

class stack

{

node \*nodes[10];

reg \*regs[10];

int top;

int count;

friend class tree;

public:

stack()

{

top=-1;

count=0;

}

void push(node\* a)

{

nodes[++top]=a;

}

void push(reg\* a)

{

regs[++top]=a;

}

node\* pop()

{

if(top>=0)

{

return nodes[top--];

}

else

return NULL;

}

reg\* popreg()

{

if(top>=0)

{

return regs[top--];

}

else

return NULL;

}

};

class tree

{

node \*root;

int i;

stack st;

public:

tree()

{

root=NULL;

i=0;

}

void gentree(node\* root, node\* l, node\* r)

{

root->left=l;

root->left->flag=1;

root->right=r;

root->right->flag=0;

}

void assignroot(node\* root)

{

this->root=root;

}

void postorder(node\* root)

{

//Generating labels for nodes

if(root)

{

postorder(root->left);

postorder(root->right);

if(root->left==NULL)

{

if(root->flag==1)

root->label=1;

else if(root->flag==0)

root->label=0;

}

else

{

if(root->left->label!=root->right->label)

{

if(root->left->label > root->right->label)

root->label=root->left->label;

else

root->label=root->right->label;

}

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

root->label=root->left->label + 1;

}

}

}

void inorder(node\* root)

{

//Displaying labelling tree

if(root)

{

inorder(root->left);

cout<<root->key;

cout<<"("<<root->label<<")";

inorder(root->right);

}

}

void codegen(node\* root)

{

//Recursive function to generate code

if(root)

{

if(root->left!=NULL && root->right->label > root->left->label)

codegen(root->right);

else if(root->left!=NULL)

codegen(root->left);

if(root->left!=NULL && root->right->label!=0 && root->right->visited==0)

codegen(root->right);

else if(root->left!=NULL && root->left->label!=0 && root->left->visited==0)

codegen(root->left);

if(root->key!='+' && root->key!='-' && root->key!='\*' && root->key!='/' && root->label==1)

{

root->visited=1;

st.count++;

cout<<"MOV "<<root->key<<","<<"R"<<st.count<<";"<<endl;

st.push(new reg(st.count, root->flag));

}

else if(root->key=='+' || root->key=='-' || root->key=='\*' || root->key=='/')

{

root->visited=1;

if(root->left->label==0 || root->right->label==0)

{

reg \*temp;

temp=st.popreg();

switch(root->key)

{

case '+':

if(temp->flag==1)

{

cout<<"ADD R"<<temp->number<<","<<root->right->key<<";"<<endl;

temp->flag=root->flag;

st.push(temp);

}

else

{

cout<<"ADD "<<root->left->key<<","<<"R"<<temp->number<<";"<<endl;

temp->flag=root->flag;

st.push(temp);

}

break;

case '-':

if(temp->flag==1)

{

cout<<"SUB R"<<temp->number<<","<<root->right->key<<";"<<endl;

temp->flag=root->flag;

st.push(temp);

}

else

{

cout<<"SUB "<<root->left->key<<","<<"R"<<temp->number<<";"<<endl;

temp->flag=root->flag;

st.push(temp);

}

break;

case '\*':

if(temp->flag==1)

{

cout<<"MUL R"<<temp->number<<","<<root->right->key<<";"<<endl;

temp->flag=root->flag;

st.push(temp);

}

else

{

cout<<"MUL "<<root->left->key<<","<<"R"<<temp->number<<";"<<endl;

temp->flag=root->flag;

st.push(temp);

}

break;

case '/':

if(temp->flag==1)

{

cout<<"DIV R"<<temp->number<<","<<root->right->key<<";"<<endl;

temp->flag=root->flag;

st.push(temp);

}

else

{

cout<<"DIV "<<root->left->key<<","<<"R"<<temp->number<<";"<<endl;

temp->flag=root->flag;

st.push(temp);

}

break;

}

}

else

{

reg \*temp1, \*temp2;

temp1=st.popreg();

temp2=st.popreg();

st.count--;

switch(root->key)

{

case '+':

if(temp1->flag==1)

{

cout<<"ADD R"<<temp1->number<<","<<"R"<<temp2->number<<";"<<endl;

}

else

{

cout<<"ADD R"<<temp2->number<<","<<"R"<<temp1->number<<";"<<endl;

}

break;

case '-':

if(temp1->flag==1)

{

cout<<"SUB R"<<temp1->number<<","<<"R"<<temp2->number<<";"<<endl;

}

else

{

cout<<"SUB R"<<temp2->number<<","<<"R"<<temp1->number<<";"<<endl;

}

break;

case '\*':

if(temp1->flag==1)

{

cout<<"MUL R"<<temp1->number<<","<<"R"<<temp2->number<<";"<<endl;

}

else

{

cout<<"MUL R"<<temp2->number<<","<<"R"<<temp1->number<<";"<<endl;

}

break;

case '/':

if(temp1->flag==1)

{

cout<<"DIV R"<<temp1->number<<","<<"R"<<temp2->number<<";"<<endl;

}

else

{

cout<<"DIV R"<<temp2->number<<","<<"R"<<temp1->number<<";"<<endl;

}

break;

}

if(temp1->number<temp2->number)

{

temp1->flag=root->flag;

st.push(temp1);

}

else

{

temp2->flag=root->flag;

st.push(temp2);

}

}

}

}

}

};

int main() {

node \*temp, \*l, \*r;

char str[20];

stack st;

tree t;

cout<<"Enter the postfix notation of the expression:"<<endl;

cin>>str;

for(int i=0;i<(int)strlen(str);i++)

{

if(str[i]!='+' && str[i]!='-' && str[i]!='\*' && str[i]!='/')

{

st.push(new node(str[i]));

}

else

{

temp=new node(str[i]);

r=st.pop();

l=st.pop();

t.gentree(temp, l, r);

st.push(temp);

}

}

t.assignroot(st.pop());

t.postorder(temp);

cout<<"\nLabeled tree for given expression "<<str<<" (representation using inorder traversal) :\n";

t.inorder(temp);

cout<<endl;

cout<<"\nAssembly language code for given expression "<<str<<" :\n";

t.codegen(temp);

return 0;

}