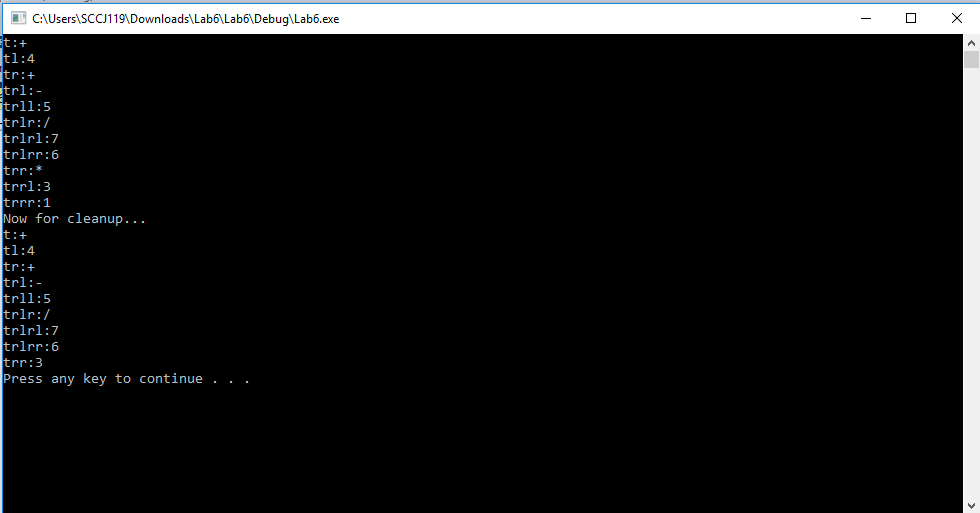
Name: William A. Brannon

Assignment: Lab Assignment Report #6

Date Due: Monday, March, 18, 2019

Class: Fundamentals of Programming III Section #1

**Program Output**



**Lab6.cpp**

// Lab6.cpp : expression trees

// By William A. Brannon on 03/01/2019

#include "stdafx.h"

#include <iostream>

#include <string>

#include <vector>

using namespace std;

char precedence[14] = { '+','-','\*','/','9','8','7','6','5','4','3','2','1','0'};

class Node {

public:

Node\* left;

Node\* right;

char id;

string name;

int origIndex;

Node() {

}

Node(char c, int i) {

id = c;

origIndex = i;

}

int valOf() {

switch (id)

{

case '+':

return left->valOf() + right->valOf();

case '-':

return left->valOf() - right->valOf();

case '\*':

return left->valOf() \* right->valOf();

case '/':

return left->valOf() / right->valOf();

default:

return id;

}

}

void clean() {

switch (id)

{

case '+':

if (hasChildren()) {

if (left->id == '0' && isdigit(right->id)) {

id = right->id;

left = NULL;

right = NULL;

}

if (right->id == '0' && isdigit(left->id)) {

id = left->id;

left = NULL;

right = NULL;

}

}

case '-':

if (hasChildren()) {

if (right->id == '0' && isdigit(left->id)) {

id = left->id;

left = NULL;

right = NULL;

}

}

case '\*':

if (hasChildren()) {

if (right->id == '1' && isdigit(left->id)) {

id = left->id;

left = NULL;

right = NULL;

}

}

case '/':

if (hasChildren()) {

if (right->id == '1' && isdigit(left->id)) {

id = left->id;

left = NULL;

right = NULL;

}

}

default:

break;

}

if (left) {

left->clean();

}

if (right) {

right->clean();

}

}

void display() {

cout << name << ":" << id << endl;

if (left) {

left->display();

}

if (right) {

right->display();

}

}

void setname(string s) {

//t denotes root, lr denote path from root

name = s;

if (left) {

left->setname(s + "l");

}

if (right) {

right->setname(s + "r");

}

}

bool hasChildren() const {

return left && right;

}

};

void validSpot(int i, Node\* parent, string s) {

if (!parent->left && i < parent->origIndex) {

parent->left = new Node(s[i], i);

}

else if (!parent->right && i > parent->origIndex) {

parent->right = new Node(s[i], i);

}

else if (parent->left && i < parent->origIndex) {

validSpot(i, parent->left, s);

}

else if (parent->right && i > parent->origIndex) {

validSpot(i, parent->right, s);

}

}

int calcMidIndex(int i) {

//calculates the middle index of a given set

//making it its own separate function for bug testing

if (i % 2 == 0) {

//even

return calcMidIndex(i - 1);

}

else {

//odd

return (i - 1) / 2;

}

}

class ExpTree{

public:

Node\* root;

int numNodes;

ExpTree(string s) {

//accepted string format is alternating between numbers and operators

//example: 4+5-7/6+3\*1

numNodes = s.length();

vector<int> tempStore;

for (char& c : precedence) {

int i = 0;

for (char& c2 : s) {

if (c == c2) {

tempStore.push\_back(i);

}

i++;

}

int k = tempStore.size();

for (int L = 0; L < k; L++) {

int j = tempStore.at(calcMidIndex(tempStore.size()));

if (!root) {

root = new Node(s[j], j);

}

else {

validSpot(j, root, s);

}

tempStore.erase(tempStore.begin() + calcMidIndex(tempStore.size()));

}

}

setname();

}

void display() {

root->display();

}

void clean() {

root->clean();

}

void setname() {

root->setname("t");

}

};

int main()

{

ExpTree tester("4+5-7/6+3\*1");

tester.display();

cout << "Now for cleanup..." << endl;

tester.clean();

tester.display();

system("pause");

return 0;

}