Laboratory 8: Cover Sheet

Name Catherine Pollock Date 10/6/2014

Section 1001

Place a check mark in the *Assigned* column next to the exercises your instructor has assigned to you. Attach this cover sheet to the front of the packet of materials you submit following the laboratory.

Activities	**Assigned:** Check or list exercise numbers	**Completed**
Implementation Testing	✓	
Programming Exercise 1		
Programming Exercise 2	✓	
Programming Exercise 3	EC	
Analysis Exercise 1	✓	
Analysis Exercise 2	✓	
	Total	

Laboratory 8: Implementation Testing

Name Catherine Pollock Date 10/6/2014

Section 1001

Check with your instructor whether you are to complete this exercise prior to your lab period or during lab.

Test Plan 8-1 (Expression Tree ADT operations)			
Test case	**Arithmetic expression**	**Expected result**	**Checked**
One operator	`+34`	`(3+4) = 7`	
Nested operators	`*+34/52`	`((3+4)*(5/2)) = 17.5`	
All operators at start	`-/*9321`	`(((9*3)/2)-1) = 12.5`	
Uneven nesting	`*4+6-75`	`(4*(6+(7-5))) = 32`	
Zero dividend	`/02`	`(0/2) = 0`	
Single-digit number	`7`	`7 = 7`	

Laboratory 8: Programming Exercise 2

Name Catherine Pollock Date 10/6/2014

Section 1001

Test Plan 8-4 (commute operation)			
Test case	**Arithmetic expression**	**Expected result**	**Checked**
1	1 = 1	1 =1	Yes
2	(1+2) = 3	(2+1) = 3	Yes
3	(1/0) = inf	(0/1) = 0	Yes
4	((4-2)+(0*(6/1))) = 2	(((1/6)*0)+(2-4)) = -2	Yes
5	((5*5)*(2/0)) = inf	((0/2)*(5*5)) = 0	Yes
6	(1+(1+(1+0))) = 3	(((0+1)+1)+1) = 3	Yes
7	(0+0) = 0	(0+0) = 0	Yes
8	((5+3)*(6-4)) = 16	((4-6)*(3+5)) = -16	Yes
9	(((2+3)/(1/7))*0) = 0	(0*((7/1)/(3+2))) = 0	Yes
10	(1+((9-9)+(9*(9/9)))) = 10	((((9/9)*9)+(9-9))+1) = 10	Yes
11	(9-(7-(5-4))) = 3	(((4-5)-7)-9) = -17	Yes
12	(4*(3*2)) = 24	((2*3)*4) = 24	Yes
13	((0+1)+(9+(7/2))) = 13.5	(((2/7)+9)+(1+0)) = 10.2857	Yes
14	(7/(3/4)) = 9.33333	((4/3)/7) = 0.190476	Yes
15	(1+(2+(4+(5+(6+(3+(2/1))))))) = 23	(((((((1/2)+3)+6)+5)+4)+2)+1) = 21.5	Yes

Laboratory 8: Programming Exercise 3

Name Catherine Pollock Date 10/6/2014

Section 1001

Test Plan 8-5 (isEquivalent operation)				
Test case	**Arithmetic Expression #1**	**Arithmetic Expression #2**	**Expected result**	**Checked**
1	1 = 1	1 =1	Yes	Yes
2	(1+2) = 3	(2+1) = 3	Yes	Yes
3	(1/0) = inf	(0/1) = 0	No	Yes
4	((4-2)+(0*(6/1))) = 2	(((1/6)*0)+(2-4)) = -2	No	Yes
5	((5*5)*(2/0)) = inf	((0/2)*(5*5)) = 0	No	Yes
6	(1+(1+(1+0))) = 3	(((0+1)+1)+1) = 3	Yes	Yes
7	(0+0) = 0	(0+0) = 0	Yes	Yes
8	((5+3)*(6-4)) = 16	((4-6)*(3+5)) = -16	No	Yes
9	(((2+3)/(1/7))*0) = 0	(0*((7/1)/(3+2))) = 0	No	Yes
10	(1+((9-9)+(9*(9/9)))) = 10	((((9/9)*9)+(9-9))+1) = 10	Yes	Yes

Laboratory 8: Analysis Exercise 1

Name Catherine Pollock Date 10/6/2014

Section 1001

What type of tree traversal (inorder, preorder, or postorder) serves as the basis of your implementation of each of the following Expression Tree ADT operations? Briefly explain why you used a given traversal to implement a particular operation.

Build
Traversal: Pre order Explanation: The expression given to us is in preorder, and each data value is saved and then buildHelper is called for left and right branches, making it top to bottom or preorder.

Expression
Traversal: In order Explanation: For example, in a tree with a root and two children: calls self to print left branch, then prints operator, then calls self to print right branch

Evaluate

Traversal: Post order

Explanation: Calls helper to do operator function until digit branches are found, then applies values found by digits to root branches and their values.

Clear

Traversal: post order

Explanation: finds end branches, deletes them, and then deletes parent branches.

Laboratory 8: Analysis Exercise 2

Name Catherine Pollock Date 10/6/2014

Section 1001

Consider the functions `writeHelper1()` and `writeHelper2()` given below:

```
void ExprTree<DataType>::writeHelper1 ( ExprTreeNode *p ) const {
   if ( p != 0 ) {
      writeHelper1(p->left);
      cout << p->dataItem;
      writeHelper1(p->right);
   }
}

void ExprTree<DataType>::writeHelper2 ( ExprTreeNode *p ) const {
   if ( p->left != 0 ) writeHelper2(p->left);
   cout << p->dataItem;
   if ( p->right != 0 ) writeHelper2(p->right);
}
```

Let `root` be the pointer to the root node of a nonempty expression tree. Will the following pair of function calls produce the same output?

```
writeHelper1(root); and writeHelper2(root);
```

If not, why not? If so, how do the functions differ and why might this difference be important?

These functions will not produce the same output because of the varying if statement checks. writeHelper1() calls to check if p is null, and if it is executes recursion and cout. writeHelper2() checks for left and right values to be null, but will still cout dataItem. This difference is important because writeHelper2() will always print out data item, without the initial check.