Laboratory 8: Cover Sheet

Name: Ernest Landrito Date: 10/23/2013

Section: 1

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	✓	
Analysis Exercise 1	✓	
Analysis Exercise 2	✓	
	Total	

Laboratory 8: Implementation Testing

Name: Ernest Landrito Date: 10/23/2013

Section: 1

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`	`7`	
Nested operators	`*+34/52`	`17.5`	
All operators at start	`-/*9321`	`12.5`	
Uneven nesting	`*4+6-75`	`32`	
Zero dividend	`/02`	`0`	
Single-digit number	`7`	`7`	

Laboratory 8: Programming Exercise 1

Name _____ Date _____

Section _____

Test Plan 8-2 (Logic Expression Tree ADT)			
Test case	**Logic expression**	**Expected result**	**Checked**
One operator	`+10`		
Nested operators	`*+10+01`		
NOT (Boolean value)	`+*10*1-0`		
NOT (subexpression)	`+-1-*11`		
NOT (nested expression)	`-*+110`		
Double negation	`--1`		
Boolean value	`1`		

Test Plan 8-3 (1-bit addition)				
`X`	`Y`	`C = *XY`	`S = +*X-Y*-XY`	**Checked**
`0`	`0`	`*00 =`	`+*0-0*-00 =`	
`0`	`1`	`*01 =`	`+*0-1*-01 =`	
`1`	`0`	`*10 =`	`+*1-0*-10 =`	
`1`	`1`	`*11 =`	`+*1-1*-11 =`	

Laboratory 8: Programming Exercise 2

Name: Ernest Landrito Date: 10/23/2013

Section: 1

Test Plan 8-4 (commute operation)			
Test case	**Arithmetic expression**	**Expected result**	**Checked**
One operator	`+34`	`7`	
Nested operators	`*+34/52`	`2.8`	
All operators at start	`-/*9321`	`0.9259`	
Uneven nesting	`*4+6-75`	`16`	
Zero dividend	`/02`	`inf`	
Single-digit number	`7`	`7`	

Laboratory 8: Programming Exercise 3

Name: Ernest Landrito Date: 10/23/2013

Section: 1

Test Plan 8-5 (isEquivalent operation)				
Test case	**Arithmetic Expression #1**	**Arithmetic Expression #2**	**Expected result**	**Checked**
Commuted plus	`+34`	`+43`	`yes`	
Commuted minus	`-34`	`-43`	`no`	
Commuted mult	`*34`	`*43`	`yes`	
Commuted Div	`/34`	`/43`	`no`	

Laboratory 8: Analysis Exercise 1

Name: Ernest Landrito Date: 10/23/2013

Section: 1

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: Post Order Explanation: The build function creates from the bottom of the tree then points the nodes above to the ones below.

Expression
Traversal: In Order Explanation: The expression function goes in order from left to right and by doing so it prints out each element in the correct order.

Evaluate

Traversal: Post Order

Explanation: The evaluate function post order traverses because it has to evaluate the most nested expressions first.

Clear

Traversal: Post Order

Explanation: The clear function post order traverses because it has to make sure it doesn't lose the data.

Laboratory 8: Analysis Exercise 2

Name: Ernest Landrito Date: 10/23/2013

Section: 1

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?
The Calls will produce the same output because they both follow the same traversal, although the second function will seg fault on an empty tree.

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

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