# **CIS023 Laboratory 4 Assignment Mockup**

## **Exercise L4-1 Chapter 19 – Binary Trees**

Write a program to do the following:

1. First define and implement a templated binary tree class, name it "**binaryTreeType**". Use the "**binaryTreeType**" example from the book for your starting point and modify it as needed. Store it as "**binaryTreeType.h**".
2. Next define and implement a templated binary search tree class name it "**binarySearchTreeType**", and derive it from the "**binaryTreeType**" class you created in step a. Store it as "**binarySearchTreeType.h**".
3. Have your program create three instances of your " **binarySearchTreeType** " named ***T1,*** ***T2,*** ***T3***.
4. Do a postorder traversal of ***T1*** and, while doing the postorder traversal, insert each node into ***T2***.
5. Do a preorder traversal of ***T2*** and, while doing the preorder traversal, insert each node into ***T3***.
6. Do an inorder traversal of ***T3***.
7. Call the **swapSubTrees()** function in ***T3***, then do an inorder traversal of ***T3***.
8. Output the heights and the number of leaves in each of the three **binarySearchTreeType** instances.

Submit only the header files (**binaryTreeType.h**, and **binarySearchTreeType.h**) and your documentation files. The header files must include all function definitions. Validate compilation of your header files with the following code. The test program will exercise all functions in the classes listed above.

//Data: 65 55 22 44 61 19 90 10 78 52 -999

#include <iostream>

#include "binarySearchTreeType.h"

using namespace std;

int main()

{

bSearchTreeType<int> treeRoot;

int num;

cout << "Enter integers ending with -999" << endl;

cin >> num;

while (num != -999)

{

treeRoot.insert(num);

cin >> num;

}

cout << endl << "Tree nodes in inorder: ";

treeRoot.inorderTraversal();

cout << endl << "Tree nodes in preorder: ";

treeRoot.preorderTraversal();

cout << endl << "Tree nodes in postorder: ";

treeRoot.postorderTraversal();

cout << endl;

cout << "Tree Height: " << treeRoot.treeHeight() << endl;

treeRoot.swapSubtrees();

return 0;

}

Your program output must look like the example below.

**G:\Coursework\CIS 23 - Data Structures and Algorithms\2020\_Fall ONLINE\\_SolutionsLab4\L4-1>L4-1Soln.exe**

**Enter numbers ending with -999**

**16 08 24 04 12 20 28 02 06 10 14 18 22 26 30 01 03 05 07 09 11 13 15 17 19 21 23 25 27 29 31 -999**

**tree1 nodes in inorder: 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28 29 30 31**

**tree1 nodes in postorder: 1 3 2 5 7 6 4 9 11 10 13 15 14 12 8 17 19 18 21 23 22 20 25 27 26 29 31 30 28 24 16**

**tree2 nodes in preorder: 1 3 2 5 4 7 6 9 8 11 10 13 12 15 14 17 16 19 18 21 20 23 22 25 24 27 26 29 28 31 30**

**tree3 nodes in inorder: 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28 29 30 31**

**tree1 height: 5**

**tree1 leaves: 16**

**tree2 height: 16**

**tree2 leaves: 15**

**tree3 height: 16**

**tree3 leaves: 15**

**swapSubtree for tree3**

**tree3 nodes in inorder: 31 30 29 28 27 26 25 24 23 22 21 20 19 18 17 16 15 14 13 12 11 10 9 8 7 6 5 4 3 2 1**

## **Exercise L4-2 Chapter 20 - Graphs**

The algorithm to determine the minimal spanning tree given in this chapter is of the order ***O(n3)***. The following is an alternative to *Prim’s*algorithm that is of the order ***O(n2)***.

**Input:** a connected weighted graph *G = (V, E)* of *n* vertices, numbered 0, 1, … n-1; starting with vertex ***s***, with a weight matrix of ***W***.

**Output:** the minimal spanning tree.

***Prim2(G, W, n, s).***

Let ***T*** = (V, E), where E =

for (j = 0; j < n; J++)  
{  
 edgeWeights[j] = W(s,j);  
 edges[j] = s;  
 visited[s] = false;  
}

edgeWeights[s] = 0;  
visited[s] = true;

while (*not all nodes are visited*)  
{  
 visited [k] = true;

E = E { (k, edges[k]) }

V = V {k}

for *each node* j *that is not visited*

{ if (W(k,j) < edgeWeights[k])  
 {  
 edgeWeights[k] = W(k.j);  
 edges[j] = k;  
 }

}

return ***T***;

Write a definition of the function for ***Prim2*** to implement this algorithm, also add this function to the class msTreeType. Write a program to test this version of Prim’s algorithm.

Submit only the following header files and your documentation files. The header files must include all function definitions.



You must the open and read the data from file **"L4-2Data.txt"** to populate your spanning graph. The **layout** of this file follows:

| Description | **Values** |
| --- | --- |
| The number of vertices | **3** |
| The vertex followed by its adjacent vertices. -999 is the terminator.  Vertex 0 directly connects to vertex 1, 2 and 3 | **0 1 2 3 -999**  **1 0 4 6 -999**  **2 0 5 6 -999**  ***(Independent Example data)*** |
| Blank line separator |  |
| The vertex followed by its adjacent vertex and the weight of the edge  Vertex 0 🡪 Vertex 1 weight is 6  Vertex 0 🡪 Vertex 2 weight is 5  Vertex 0 🡪 Vertex 3 weight is 2  -999 is the terminator | **0 1 6 2 5 3 2 -999**  **1 0 6 4 2 6 4 -999**  **2 0 5 5 7 6 5 -999**  ***(Independent Example data)*** |

Validate compilation of your header files with the following code. The test program will exercise all functions in the classes listed above.

#include <iostream>

#include "minimalSpanTreeType.h"

using namespace std;

int main()

{

msTreeType spanTree(50);

cout << "\*\*\*\*\*\*\* Using Prim2 Algorithm \*\*\*\*\*\*\*\*\*" << endl;

spanTree.createSpanningGraph();

spanTree.prim2(0);

spanTree.printTreeAndWeight();

system("PAUSE");

return 0;

}

**G:\Coursework\CIS 23 - Data Structures and Algorithms\2020\_Fall ONLINE\\_SolutionsLab4\L4-2>L4-2Soln.exe**

**\*\*\*\*\*\*\* Using Prim2 Algorithm \*\*\*\*\*\*\*\*\***

**Enter input file name: L4-2Data.txt**

**Source Vertex: 0**

**Edges Weight**

**(6, 1) 4**

**(0, 2) 5**

**(0, 3) 2**

**(1, 4) 2**

**(2, 5) 7**

**(2, 6) 5**

**Minimal Spanning Tree Weight: 25**

See the Example Graph Data the next page

**G:\Coursework\CIS 23-Data Structures and Algorithms\2020\_Fall ONLINE\\_SolutionsLab4\L4-2>L4-2TestData.txt**

**9**

**0 1 4 7 -999**

**1 0 2 4 -999**

**2 1 4 5 -999**

**3 4 6 -999**

**4 0 1 2 3 6 7 8 -999**

**5 2 7 -999**

**6 3 4 7 8 -999**

**7 0 4 5 6 -999**

**8 4 6 -999**

**0 1 5 4 3 7 2 -999**

**1 0 5 2 2 4 2 -999**

**3 4 8 6 1 -999**

**4 0 3 1 2 2 5 3 8 6 2 4 3 8 15 -999**

**5 2 9 7 5 -999**

**6 3 1 4 2 7 7 8 4 -999**

**7 0 2 4 3 5 5 6 7 -999**

**8 4 15 6 4 -999**

## **Exercise L4-3 Chapter 21 – Standard Template Library**

In lab three you submitted Programming Exercise L3-2, which converted an infix expression to a postfix expression. Redo the postfix expression calculator program so that it uses the STL **class stack** to evaluate the postfix expressions.

Submit the header file (infixToPostfix.h), a separate implementation file (infixToPostfix.cpp), and your documentation files. The header file must include all function declarations and the infix expression must be terminated by a semicolon. Validate compilation of your header file with the following code. The test program will exercise all functions in the classes listed above.

//Main program infix to postfix

#include <iostream>

#include <fstream>

#include <string>

#include "infixToPostfix.h"

using namespace std;

int main()

{

infixToPostfix InfixExp;

string infix = "A/B+C\*D/E-(A/((B-C\*D)/A)\*A);”

InfixExp.getInfix(infix);

InfixExp.showInfix();

InfixExp.showPostfix();

cout << endl;

return 0;

}

**Example of sample program output:**

**Infix Expression: A+B-C;**

**Postfix Expression: AB+C-**

**Infix Expression: (A+B)\*C;**

**Postfix Expression: AB+C\***

**Infix Expression: (A+B)\*(C-D);**

**Postfix Expression: AB+CD-\***

**Infix Expression: A+((B+C)\*(E-F)-G)/(H-I);**

**Postfix Expression: ABC+EF-\*G-HI-/+**

**Infix Expression: A+B\*(C+D)-E/F\*G+H;**

**Postfix Expression: ABCD+\*+EF/G\*-H+**

**Infix Expression: (A\*(B\*(C/(D/E))/F)\*G)+H;**

**Postfix Expression: ABCDE//\*F/\*G\*H+**