CptS 122 – Data Structures

Programming Assignment 8: Data Analysis using Binary Search Trees

Assigned: Friday, November 9, 2018
Due: Friday, November 16, 2018 by midnight

I. Learner Objectives:

At the conclusion of this assignment, participants should be able to:

- Analyze a basic set of requirements for implementing and testing a solution to a problem
- Design, implement and test classes in C++
- Design and apply inheritance
- Design with polymorphism
- Design and implement a dynamically linked binary search tree

II. Prerequisites:

Before starting this assignment, participants should be able to:

- Analyze a basic set of requirements for a problem
- Compose basic C++ language programs
- Create basic test cases for a program
- Apply arrays, strings, and pointers
- Declare and define *constructors*
- Declare and define *destructors*
- Compare and contrast *public* and *private* access specifiers in C++
- Describe what is an *attribute* or data member of a class
- Describe what is a *method* of a class
- Apply and implement *overloaded* functions
- Distinguish between pass-by-*value* and pass-by-*reference*
- Discuss *classes* versus *objects*

III. Overview & Requirements:

Summary:
For this assignment you are implementing a system for detecting trends in consumer products over a 48 hour period. We are interested in knowing which products are purchased and sold, the least and most, by various retail stores throughout the United States. When a product is tagged as *purchased* it indicates that a certain retail store bought units of the product from a supplier. When a product is tagged as *sold* it indicates that a certain retail store sold that many units of a product. Your system must read product data from a .csv file, and store the data in a way that inserts data in better than linear time (O(n)) in most cases. Since, a binary search tree (BST) is a reasonably efficient data structure for inserting and searching data (O (log n) for balanced trees), you must create two BSTs; one tree represents the products that were sold and the other tree represents the products that were bought. The system must leverage the organization of the trees to display,

which products were least bought and sold, and which were most bought and sold for that 48 hour period. Your system is only required to output the following to the screen:

- The contents of the two BSTs, which will be printed in order
- The product that type and number of units that sold the most
- The product that type and number of units that sold the least
- The product that type and number of units that were purchased the most
- The product that type and number of units that were purchased the least

Class Design:

For this assignment you are required to implement a dynamically linked binary search tree. You will first need to start by defining an *abstract* base class `Node`, which encapsulates the following:

Data members:

mData : std::string // # denotes protected
mpLeft : Node *
mpRight : Node *

Member functions:

+ virtual destructor // + denotes public
+ constructor which accepts a string to set the data in the node; each pointer in the node is set to NULL
+ setters – one for each data member (3 total should be defined)
+ getters – one for each data member (3 total should be defined, the 2 defined to get the pointers should return a reference to the pointer, i.e. Node *&)
+ pure virtual printData () function, which must be overridden in class `TransactionNode`

Next define a class `TransactionNode`, which *publically* inherits from abstract base class `Node`. Class `TransactionNode` must encapsulate the following:

New Data members:

- mUnits : int // - denotes private

New Member functions:

+ destructor // + denotes public
+ constructor which accepts a string to set the data and an integer to set the number of units in the node; should invoke class `Node`'s constructor
+ setter
+ getter
+ printData (), which overrides the pure virtual function in class `Node`

Now define a class `BST`, which encapsulates the following:

Data members:

- mpRoot : Node * // yes, we want a pointer to a `Node`, not `TransactionNode` here!

Member functions:

+ destructor // calls destroyTree ()
- destroyTree () // yes, it's private, and it should visit each node in postOrder to delete them
+ default constructor
+ setter
+ getter
+ insert () // public used to hide pointer information, i.e. won't pass in the root of the tree into this function, only the private insert () function

- insert () // yes, it's private, and it dynamically allocates a `TransactionNode` and inserts recursively in the correct subtree based on mUnits; should pass in a reference to a pointer (i.e. Node *& pT)

+ inOrderTraversal () // yes, once again it's private to hide pointer information

- inOrderTraversal (), which recursively visits and prints the contents (mData and mUnits) of each node in the tree in order; each node's printData () should be called

contents should be printed on a separate line; must call the printData () function associated with the `TransactionNode`

+ findSmallest (), which returns a reference to a `TransactionNode` (i.e `TransactionNode &`) with the smallest mUnits

+ findLargest (), which returns a reference to a `TransactionNode` with the largest mUnits

Lastly, define a class `DataAnalysis`, which encapsulates the following:

Data members:

- mTreeSold : BST
- mTreePurchased : BST
- mCsvStream : ifstream

Member functions:

- A function that opens data.csv // yes, it's private, and must use mCsvStream to open the file
- A function that reads one line from the file and splits the line into units, type, and transaction fields
- A function that loops until all lines are read from the file; calls the function below, and then displays the current contents of both BSTs; use inOrderTraversal () to display the trees
- A function that compares the transaction field and inserts the units and type into the appropriate tree (mTreeSold or mTreePurchased) // note with the way the data.csv file is organized the trees will be fairly balanced
- A function that writes to the screen the trends we see in our tree; the function must display the type and number of units that are least purchased and sold, and the most purchased and sold

+ runAnalysis (), which is the only public function in the class, aside from possibly a constructor and/or destructor; this function calls the other private functions

What should go in main ()?

```
DataAnalysis obj;
obj.runAnalysis ();
```

Questions to Ponder:

Questions for you to consider (you do not need to submit answers to these questions):

- We understand that a BST is most efficient when it is balanced. If the data.csv file was not already organized to provide a fairly balanced tree, how would you balance the tree as you insert?
- Do you think there are other data structures that would be better suited for this type of problem? Why?
- What would happen to our program if we found duplicate products or # of units sold/purchased in the file? Would we need a data structure to efficiently combine the products and # of units? Hash table?

IV. Submitting Assignments:

1. Using the OSBLE+ MS VS plugin, please submit your solution. Please visit https://github.com/WSU-HELPLAB/OSBLE/wiki/Submitting-an-Assignment for more information about submitting using OSBLE+.
2. Your project must contain at least one header file (.h files) and two C++ source files (which must be .cpp files).
3. Your project must build properly. The most points an assignment can receive if it does not build properly is 120 out of 200.

V. Grading Guidelines:

This assignment is worth 100 points. Your assignment will be evaluated based on adherence to the requirements. We will grade according to the following criteria:

- 5 pts – Appropriate design, style, and commenting according to class standards
- 5 pts – Node class and all functions described above
- 10 pts – TransactionNode class, which inherits from the Node class, and all functions described above
- 40 pts – BST class
  - 5 pts - destroyTree ()
  - 10 pts – insert () functions: 8 pts private one, 2 points public one
  - 7 pts – inOrderTraversal () functions: 5 pts private one, 2 points public one
  - 7 pts – findSmallest ()
  - 7 pts – findLargest ()
  - 4 pts – other functions
- 35 Pts – DataAnalysis class
  - 2 pts - for opening data.csv
  - 8 pts – for reading a line and splitting it
  - 10 pts – for reading all lines in the file and inserting into the appropriate tree (mTreeSold and mTreePurchased)
  - 10 pts – for determining the trends and displaying them to the screen
  - 5 pts – other functions
- 5 pts – main ()