HW12

Create a template for a *Node* class and a *LinkedList* class. The classes should be parameterized by a single *typename* (*class*), which we will call *T* in this document, and which can be either a primitive type or a class.

Four files are provided for you. *main.cpp*, which can be used to test our templates. *MyClass.h*, *MyClass.cpp*, which contain sufficient functionality to work with the templates and *output.txt*, which contains the output with my code and the *main.cpp* method. Your output does not have to be exactly like my output.

Hint: When writing your template, put the .h stuff and the .cpp stuff in the .h file. Not doing this can lead to problems where the template is not instantiated. If that is the case you will get link time error messages saying that functions defined in your templates cannot be found.

The following functions and fields should be defined for each class. You may define other functions as convenient.

Node:

data fields.

data: this field contains the data associated with each node and is of *T*.
prev and *next*: these are pointers to a node, and represent the forward and back pointers in the linked list.

functions:

Node(T data): A constructor that takes a single argument of type *T*. The *prev* and *next* fields should be set to NULL.
~Node( ): a destructor. It can do nothing.
setNext, *setPrev*: set the *prev* and *next* fields described above.
getNext, *getPrev*: get and return the *prev* and *next* fields described above.
getdata: returns the data field above.
Overloaded <. and *==* operators: These will compares two nodes by comparing their data fields. When *T* specifies a class you can assume that the class defines a < and == operator that evaluates the data fields (see *MyClass.cpp* for an example). When *T* specifies a primitive the data field is a primitive and the built in primitive operations will be used by C++.

Overloaded << operator. This takes an *ostream* and a *Node* as arguments and places *Node* on the *ostream*. In my solution I did this by putting the *data* field of the *Node* on the *ostream*, and assuming the *data* object (of type *T*) defines a << operator. See *MyClass.cpp* for an example of a class that can serve as a data that does this.

LinkedList:

data fields:

head: a pointer to the head of the linked list.

functions:

LinkedList( ): A zero arg constructor that sets *head* to NULL.
LinkedList: constructor that takes a *Node* as an argument. This creates a list with the head of the list pointing to the *Node*.
~LinkedList: a destructor that deletes all of the nodes in the list.
getNodeData: takes a *T* as an argument and returns the matching *T* in the list. This sounds pointless, but as we can see in *main.cpp*, the *T* can be an object, e.g., *MyClass*, which contains a key and an associated datum as a pair. Comparisons of *Nodes* will rely on the *Node* overloading < and == operations. Comparisons of *T* objects will rely on < and == operators implemented in the *T* class.
addNode: takes a data of type *T* as an argument, constructs a *Node* and adds it to the linked list. *Node*s should be added in sorted order, i.e. the given a *Node* X and a *Node* Y such that X.next points to Y, X will be less than Y. The output from *main.cpp* should make this clearer. *addNode* returns a pointer to the *data* field of the node, if found, and NULL otherwise.
deleteNode: takes a data of type *T* as an argument, finds a matching node, if it exists, removes it from the LinkedList and deletes the node. *deleteNode* returns a pointer to the data field of the node, if found, and NULL otherwise.
Overloaded <, == and *<<* operators: compare two Nodes. < and == return true or false, << returns an *ostream*.
print: prints the linked list. In my version I print the result of the "<<" operation on each Node in the linked list.