Name_____

1. Let's extend our Node data structure and out linked list by adding a second pointer. Let a Node now contain the following members,

```
void initialize(int aValue);          // set myData to aValue, nextPtr to NULL
void show(void);                      // print out myData
int get(void);                        // return myData
void add(Node* aNode, int aPosition);
int myData;
Node* forwardPtr;
Node* backwardPtr;
```

   This add function is to accept an argument of type pointer to Node and an index into our linked list where the new Node is to be inserted.

   Demonstrate your doubly linked list by inserting several nodes at different indices.

2. Let's now take the same data structure and make the following modifications…

```
void initialize(int aValue);          // set myData to aValue, nextPtr to NULL
void show(void);                      // print out myData
int get(void);                        // return myData
void add(Node* aNode);
int myData;
Node* leftChild;
Node* rightChild;
```

   We'll use the data structure to build a more complex type, a binary tree, as follows. Declare and define several nodes containing the following data:

{ 2, 7, 3, 9, 1, 4, 6, 8 }

Begin with a single instance of a Node, call it head. Modify the add method to support the following:

1. If the value of the data in the new node is greater than that of the node head and the right child pointer of head is NULL, add the new node as the right child of head.
2. If the value of the data in the new node is less than that of the node head and the left child pointer of head is NULL, add the new node as the left child of head.
3. If the value of the data in the new node is equal to that of the node head, it is rejected.
4. If the right (left) child pointer is not NULL, repeat steps 1 or 2 on the right (left) child until the new node can be entered.

Display your tree when complete…