Assignment Objective: Build skills on C class creation and integration while implementing a **Binary Search Tree (BST)**.

Requirements:

* Create a class called BST with the following BST.h and BST.cpp contents:
  + Our standard Key-Value type definition:

typedef struct key\_value {

int key;

int value;

} KEY\_VALUE;

* + A node class:
    - private:
      * Variables:
        + KEY\_VALUE kv; // holds the nodes value
        + int h; // for future use
        + node \*left, \*right; // left and right children pointers
    - public:
      * The constructor for node, which should be of the form node(KEY\_VALUE kv)
    - Friends class BST
  + A BST class:
    - Private members:
      * An integer “nCount” to record the number of entries in the BST
      * A root pointer, node \*root, to point to the root node of the entire tree.
      * KEY\_VALUE findMin(node \*ptr) which returns the minimum key in the subtree whose root node is pointed to by ptr.
      * Private methods to support insert(), remove(), getValue(), printIt(), and clear().
    - Public members:
      * constructor BST() that causes the object to be initialized
      * destructor ~BST() –deletes all the nodes in the BST in preparation for the BST to be destroyed.
      * bool insert(KEY\_VALUE kv) – Inserts kv into the tree; returns true for success; returns false if the key is already in the tree.
      * bool remove(int key) – removes a node with the matching key, deleting the node; returns true on success; returns if no node has that key.
      * bool getValue(int key, int &value) const – returns true if the key is found is in the tree, also sets the value to the value of the node’s kv pair; otherwise it returns false.
      * void printIt() const – results in the BST’s key-value pairs being printed in ascending order by the keys, one pair per line.
      * int count() const – returns the number of nodes in the tree
      * void clear() – removes all nodes from the BST, making the BST empty.
* You must not use any other data structure, whether built-in or otherwise.
* Compile your program: make
* Demonstrate that the BST data structure works:
  + Run your program as follows:

make

./BST < BSTinput.txt > BSToutput.txt

* + Compare your BSToutput.txt to the posted BSTcorrectOutput.txt file
* Deliverables:
  + Into D2L put a zip file containing the BST.h, BST.cpp, BSToutput.txt, and Makefile files.
  + Turned into class, a hardcopy of your BST.h, BST.cpp, and BSToutput.txt files.