```
//*************************************     PROGRAM IDENTIFICATION   **********************************
//*                                                                                                   *
//*     PROGRAM FILE NAME:   Program6.cpp           ASSIGNMENT  #:  6           Grade: _______      *
//*                                                                                                   *
//*     PROGRAM AUTHOR:       ____________________________________                                    *
//*                                    Adrian Beloqui                                                 *
//*                                                                                                   *
//*     COURSE #:   CSC  36000  11                                  DUE DATE:    Apr 14, 2017       *
//*                                                                                                   *
//*****************************************************************************************************

//*************************************    PROGRAM DESCRIPTION   **************************************
//*                                                                                                   *
//*      PROCESS: This program is designed to read a file and store the records as items of an inventory  *
//*               into a binary tree structure. It is to read commands and perform different actions      *
//*               depending on the command that is executed. It is to insert, delete, update and print    *
//*               the items from the inventory.                                                           *
//*                                                                                                   *
//*      USER DEFINED                                                                                 *
//*      MODULES      : main - Controlls the flow of the entire program, calling functions is the        *
//*                            right sequence and printing the labels into the output file.           *
//*                     Header - Prints a header in the output file.                                  *
//*                     Footer - Prints a footer in the output file.                                  *
//*                     PageBreak - Adds end lines to the output file.                                *
//*                     InventoryCLASS::InventoryCLASS - Initializes the private members of the class.    *
//*                     InventoryCLASS::ReadNode- Read an item from the input file.                   *
//*                     InventoryCLASS::Print - Prints all the items from the inventory or only one.      *
//*                     InventoryCLASS::InsertNode - Inserts a node into the binary tree.             *
//*                     InventoryCLASS::GetRoot - Gets the pointer of the root of the binary tree         *
//*                     InventoryCLASS::CheckExistance - Checks if a node exists in the binary tree and   *
//*                                                      returns it.                                  *
//*                     InventoryCLASS::DeleteNode - Deletes a node from the binary tree.             *
//*                     InventoryCLASS::PatchParent - Adjusts the child nodes of a deleted node.          *
//*                     InventoryCLASS::UpdateNode - Updates a node depending on the command executed.    *
//*                                                                                                   *
//*****************************************************************************************************

    //Imports
#include <string>
#include <fstream>
#include <iomanip>

    //Definition of constants
#define NOT !
#define LinesPerPage 66

    //Definition of namespace
using namespace std;

    //Definition of a node structure
struct NodeType{
    char id[6];
    char name[21];
    int quantityOnHand;
    int quantiyOnOrder;

    NodeType *LPtr;
    NodeType *RPtr;
};

    //Definition of classes
class InventoryCLASS{
public:
        // Constructor
    InventoryCLASS() { Root = NULL; };
```

```
        // Functions
    bool InsertNode(NodeType &);
    void ReadNode(ifstream &, NodeType &, char);
    bool Print(ofstream &, int &, NodeType *, char, NodeType *);
    NodeType* GetRoot() { return Root; }
    NodeType* CheckExistance(NodeType *);
    bool DeleteNode(ofstream &, NodeType &);
    void PatchParent(NodeType *, NodeType *, NodeType *);
    bool UpdateNode(NodeType *, char);
private:
    NodeType *Root;
};

    //Function prototypes definitions
void Header(ofstream &);
void Footer(ofstream &);
void PageBreak(ofstream &, int &);

//****************************************** FUNCTION MAIN  ***********************************************
int main()
{
    ifstream InFile;
    ofstream OutFile;

        //Set initial variables
    int linesWritten = 0;
    bool endOfFile = false;
    char command, printingType;
    NodeType tempNode;
    InventoryCLASS inventory;

        // Open the input file
    InFile.open("tree_in.txt", ios::in);

    // Create the output file
    OutFile.open("output6.txt", ios::out);

        // Print the header in the output file.
    Header(OutFile);
        // Add amount of lines written into the output file
    linesWritten += 3;
        // Print separator line
    OutFile << "================================================="
        << "============================================================" << endl << endl;
        // Add amount of lines written into the output file
    linesWritten += 2;

        // Read the input file
    do {
            // Read the command character
        InFile >> command;
            // Execute the right intructions depending on the command executed
        switch (command){
        case 'I':
                // Read a node
            inventory.ReadNode(InFile, tempNode, command);
                // Insert a node
            if (inventory.InsertNode(tempNode)){
                    // Print success message
                OutFile << "Item ID Number " << tempNode.id << " successfully entered into database."
                    << endl;
                OutFile << "------------------------------------------------------------" << endl;
                linesWritten += 2;
            }
            else {
                    // Print error duplicate message
```

```
            OutFile << "ERROR - Attempt to insert a duplicate item " << tempNode.id
                << " into the database." << endl;
            OutFile << "-------------------------------------------------------------" << endl;
            linesWritten += 2;
        }
        break;
    case 'D':
            // Read a node
        inventory.ReadNode(InFile, tempNode, command);
            // Delete a node
        if (inventory.DeleteNode(OutFile, tempNode)){
                // Print success message
            OutFile << "Item ID Number " << tempNode.id << " successfully deleted from database."
                << endl;
            OutFile << "-------------------------------------------------------------" << endl;
            linesWritten += 2;
        }
        else {
                // Print error message
            OutFile << "ERROR --- Attempt to delete an item " << tempNode.id
                << " not in the database list." << endl;
            OutFile << "-------------------------------------------------------------" << endl;
            linesWritten += 2;
        }
        break;
    case 'P':
            // Read printing type
        InFile >> printingType;
            // Print page break
        PageBreak(OutFile, linesWritten);
            // Print labels
        OutFile << setw(40) << "JAKE'S HARDWARE INVENTORY REPORT" << endl;
        OutFile << "Item" << setw(16) << "Item" << setw(26) << "Quantity"
            << setw(13) << "Quantity" << endl;
        OutFile << "ID Number" << setw(18) << "Description" << setw(19) << "On Hand"
            << setw(13) << "On Order" << endl;
        OutFile << "-------------------------------------------------------------" << endl;
        if (printingType == 'N'){
                // Read node
            inventory.ReadNode(InFile, tempNode, printingType);
                // Print node
            if (NOT inventory.Print(OutFile, linesWritten, inventory.GetRoot(),
                printingType, &tempNode)){
                    // Print error message
                OutFile << "Item " << tempNode.id << " not in database. Print failed."
                    << endl;
                OutFile << "-------------------------------------------------------------"
                    << endl;
                linesWritten += 2;
            }
        }
        else {
                // Print tree
            inventory.Print(OutFile, linesWritten, inventory.GetRoot(), printingType, NULL);
        }
        OutFile << endl << endl;
        linesWritten += 6;
        break;
    case 'S':
            // Read node
        inventory.ReadNode(InFile, tempNode, command);
            // Update node
        if (inventory.UpdateNode(&tempNode, command)){
                // Print success message
            OutFile << "Quantity on Hand for item " << tempNode.id
                << " successfully updated." << endl;
```

```
                OutFile << "------------------------------------------------------------" << endl;
                linesWritten += 2;
            }
            else {
                    // Print error message
                OutFile << "Item " << tempNode.id << " not in database. Data not updated." << endl;
                OutFile << "------------------------------------------------------------" << endl;
                linesWritten += 2;
            }
            break;
        case 'O':
                // Read Node
            inventory.ReadNode(InFile, tempNode, command);
                // Update Node
            if (inventory.UpdateNode(&tempNode, command)){
                    // Print success message
                OutFile << "Quantity on Order For item  " << tempNode.id
                    << " successfully updated." << endl;
                OutFile << "------------------------------------------------------------" << endl;
                linesWritten += 2;
            }
            else {
                    // Print error message
                OutFile << "Item " << tempNode.id << " not in database. Data not updated." << endl;
                OutFile << "------------------------------------------------------------" << endl;
                linesWritten += 2;
            }
            break;
        case 'R':
                // Read node
            inventory.ReadNode(InFile, tempNode, command);
                // Update node
            if (inventory.UpdateNode(&tempNode, command)){
                    // Print success message
                OutFile << "Quantity on Hand for item " << tempNode.id
                    << " successfully updated." << endl;
                OutFile << "------------------------------------------------------------" << endl;
                linesWritten += 2;
            }
            else {
                    // Print error message
                OutFile << "Item " << tempNode.id << " not in database. Data not updated." << endl;
                OutFile << "------------------------------------------------------------" << endl;
                linesWritten += 2;
            }
            break;
        case 'Q':
            endOfFile = true;
            break;
        }
    } while (NOT endOfFile);

        // Print page break
    PageBreak(OutFile, linesWritten);

        // Print the footer into the output file.
    Footer(OutFile);

    return 0;
}
//***********************************  END OF FUNCTION MAIN  ***************************************

//************************************** FUNTION DELETENODE  ***************************************
bool InventoryCLASS::DeleteNode(ofstream &OutFile, NodeType &node)
{
        // Receives - The output file and a node
```

```
        // Task - Deletes a node from the tree
        // Returns - The output file, a node and a boolean

    NodeType *delnode, *parnode, *node1, *node2, *node3;
        // Declare a flag to indicate the node to be deleted  is found
    bool found = false;
        // Set the pointers to start at the root
    delnode = Root;
    parnode = NULL;
        // Search the tree until we find the node to be deleted or until there
        //  are no more nodes to examine
    while ((found == false) && (delnode != NULL)) {
            // Set flag to true if we find the node
        if (strcmp(node.id, delnode->id) == 0) {
            found = true;
        }
        else     // Otherwise keep track of the parent node and move down
            // the appropriate branch of the tree
        {
            parnode = delnode;
            if (strcmp(node.id, delnode->id) < 0) {
                delnode = delnode->LPtr;
            }
            else {
                delnode = delnode->RPtr;
            }
        }
    }

    if (found == false)
        return found;
    else
    {
        if (delnode->LPtr == NULL) {
            if (delnode->RPtr == NULL) {  // CASE 2 – Node has NO children
                PatchParent(NULL, parnode, delnode);
            }
            else {           // CASE 3 – Node has ONE right child
                PatchParent(delnode->RPtr, parnode, delnode);
            }
        }
        else
        {
            if (delnode->RPtr == NULL)  // CASE 4 – Node has ONE left child
                PatchParent(delnode->LPtr, parnode, delnode);
            else {           // CASE 5 – Node has TWO children
                node1 = delnode;
                node2 = delnode->LPtr;
                node3 = node2->RPtr;
                while (node3 != NULL) {
                    node1 = node2;
                    node2 = node3;
                    node3 = node3->RPtr;
                }
                if (node1 != delnode){
                    node1->RPtr = node2->LPtr;
                    node2->LPtr = delnode->LPtr;
                }
                node2->RPtr = delnode->RPtr;
                PatchParent(node2, parnode, delnode);
            }  /* end else */
        } /* end else  */
    } /* end else */
    node = *delnode;
    return found;
}
```

```
//***********************************  END OF FUNCTION DELETENODE  ************************************

//*************************************** FUNTION PATCHPARENT  ***************************************
void InventoryCLASS::PatchParent(NodeType *Newparnode, NodeType *parnode, NodeType *delnode)
{
        // Receives - Three nodes
        // Task - Assign the child nodes of a delete node to a parent node
        // Returns - Nothing

    if (parnode == NULL)
        Root = Newparnode;
    else {
        if (parnode->LPtr == delnode) {
            parnode->LPtr = Newparnode;
        }
        else {
            parnode->RPtr = Newparnode;
        }
    }

}
//***********************************  END OF FUNCTION PATCHPARENT  ************************************

//*************************************** FUNTION UPDATENODE  ***************************************
bool InventoryCLASS::UpdateNode(NodeType *node, char command) {

        // Receives - A node and a character indicating the command executed
        // Task - Updates the members of a node depending on the command executed
        // Returns - A boolean

        // Check if the node exists
    NodeType *nodeToUpdate = CheckExistance(node);
    if (nodeToUpdate == NULL){
        return false;
    }

    if (node != NULL)
    {
            // Update a node depending on the command executed
        switch (command){
        case 'S':
            if (strcmp(node->id, nodeToUpdate->id) == 0){
                nodeToUpdate->quantityOnHand -= node->quantityOnHand;
                return true;
            }
            break;
        case 'O':
            if (strcmp(node->id, nodeToUpdate->id) == 0){
                nodeToUpdate->quantiyOnOrder += node->quantityOnHand;
                return true;
            }
            break;
        case 'R':
            if (strcmp(node->id, nodeToUpdate->id) == 0){
                nodeToUpdate->quantityOnHand += node->quantityOnHand;
                nodeToUpdate->quantiyOnOrder -= node->quantityOnHand;
                return true;
            }
            break;
        }
    }

    return false;
}
//***********************************  END OF FUNCTION UPDATENODE  ************************************
```

```
//************************************** FUNTION PRINT  ********************************************
bool InventoryCLASS::Print(ofstream &OutFile, int &linesWritten, NodeType *root, char printingType,
    NodeType *node) {

        // Receives - The output file, the amount of lines written, the root node, the printing
        //            character, and a node
        // Task - Print the entire tree or just one node
        // Returns - The output file, the lines written and a boolean

        // Check what type of printing is
    switch (printingType){
    case 'E':
        if (root != NULL)
        {
                // Traverse the left subtree
            Print(OutFile, linesWritten, root->LPtr, printingType, NULL);

                // Print node
            OutFile << setw(6) << root->id;
            OutFile << setw(30) << root->name;
            OutFile << setw(10) << root->quantityOnHand;
            OutFile << setw(10) << root->quantiyOnOrder << endl;
            linesWritten++;

                // Traverse the right tree
            Print(OutFile, linesWritten, root->RPtr, printingType, NULL);
        }
        break;
    case 'N':
        if (root != NULL)
        {
                // Check if the node exists
            NodeType *tempNode = CheckExistance(node);

                // Check if the node exists
            if (tempNode == NULL){
                return false;
            }

                // Print the node
            OutFile << setw(6) << root->id;
            OutFile << setw(30) << root->name;
            OutFile << setw(10) << root->quantityOnHand;
            OutFile << setw(10) << root->quantiyOnOrder << endl;
            linesWritten++;
            return true;
        }
        break;
    }

    if (printingType == 'E'){
        return true;
    }
    else {
        return false;
    }

}
//**********************************  END OF FUNCTION PRINT  ***************************************

//************************************** FUNTION INSERTNODE  ***************************************
bool InventoryCLASS::InsertNode(NodeType &node)
{
        // Receives - A node
        // Task - Inserts a node into a tree with an "in order" format
        // Returns - A node and a boolean
```

```
    bool inserted = false;
    NodeType *tempNode = CheckExistance(&node);

        // Check if the node is already in the tree
    if (tempNode != NULL){
        return inserted;
    }

    NodeType  *newPtr, *CurrPtr;
    newPtr = new NodeType();

    if (newPtr != NULL)
    {
            // Copy the node into a new node to be inserted
        strcpy_s(newPtr->id, node.id);
        strcpy_s(newPtr->name, node.name);
        newPtr->quantityOnHand = node.quantityOnHand;
        newPtr->quantiyOnOrder = node.quantiyOnOrder;
        newPtr->LPtr = NULL;
        newPtr->RPtr = NULL;
        CurrPtr = Root;
        while (inserted == false)
        {
                // Check if the tree is empty
            if (CurrPtr == NULL)
            {
                Root = newPtr;
                inserted = true;
            }
            else
            {
                    // Check to which subtree the node should be added
                if (strcmp(node.id, CurrPtr->id) < 0) {
                    if (CurrPtr->LPtr != NULL)
                        CurrPtr = CurrPtr->LPtr;
                    else
                    {
                        CurrPtr->LPtr = newPtr;
                        inserted = true;
                    }
                }
                else {
                    if (CurrPtr->RPtr != NULL)
                        CurrPtr = CurrPtr->RPtr;
                    else
                    {
                        CurrPtr->RPtr = newPtr;
                        inserted = true;
                    }
                }
            }
        }
    }
    return inserted;
}
//***********************************  END OF FUNCTION INSERTNODE  ************************************

//*************************************** FUNTION CHECKEXISTANCE  ************************************
NodeType* InventoryCLASS::CheckExistance(NodeType *mainNode)
{
        // Receives - A node
        // Task - Checks if a node already exists and returns the node if it exists
        // Returns - A node

    bool found = false;
```

```
    NodeType *comparingNode = Root;
    NodeType *returningNode = new NodeType();

    while ((found == false) && (comparingNode != NULL)) {
            // Set flag to true if we find the node
        if (strcmp(mainNode->id, comparingNode->id) == 0) {
            found = true;
            strcpy_s(returningNode->id, comparingNode->id);
            strcpy_s(returningNode->name, comparingNode->name);
            returningNode->quantityOnHand = comparingNode->quantityOnHand;
            returningNode->quantiyOnOrder = comparingNode->quantiyOnOrder;
            returningNode->LPtr = comparingNode->LPtr;
            returningNode->RPtr = comparingNode->RPtr;
            return returningNode;
        }
        else     // Otherwise keep track of the parent node and move down
            // the appropriate branch of the tree
        {
            if (strcmp(mainNode->id, comparingNode->id) < 0) {
                comparingNode = comparingNode->LPtr;
            }
            else {
                comparingNode = comparingNode->RPtr;
            }
        }
    }

    return NULL;
}
//***********************************  END OF FUNCTION CHECKEXISTANCE  ***********************************

//*************************************** FUNTION READNODE  ****************************************************
void InventoryCLASS::ReadNode(ifstream &InFile, NodeType &node, char command)
{
        // Receives - The input file, the node, and the command character
        // Task - Reads the input data into the node depending on the command character
        // Returns - The input file and the node

        // Read input data depending on the command character
    switch (command){
    case 'I':
            // Read input data
        InFile >> ws;
        InFile.getline(node.id, 6);
        InFile.getline(node.name, 21);
        InFile >> node.quantityOnHand;
        InFile >> node.quantiyOnOrder;
        node.LPtr = NULL;
        node.RPtr = NULL;
        break;
    case 'D':
            // Read input data
        InFile >> ws;
        InFile.getline(node.id, 6);
        InFile.getline(node.name, 21);
        break;
    case 'N':
            // Read input data
        InFile >> ws;
        InFile.getline(node.id, 6);
        break;
    case 'S': case 'O': case 'R':
            // Read input data
        InFile >> ws;
        InFile.getline(node.id, 6);
        InFile >> node.quantityOnHand;
```

```
        break;
    }


}
//***********************************  END OF FUNCTION READNODE  ***************************************

//*************************************** FUNTION PAGEBREAK  ***************************************
void PageBreak(ofstream &Outfile, int &limit)
{
        // Receives - The output file and the amount of lines written in the current page.
        // Task - Add end lines to the output file.
        // Returns -  The output file and the amount of lines written in the current page.

        // Calculate amount of blank lines needed for new page
    limit = LinesPerPage - limit;

        // Print blank lines
    for (int i = 0; i < limit; i++){
        Outfile << endl;
    }
        // Reset amount of lines writen in one page
    limit = 0;
}
//***********************************  END OF FUNCTION PAGEBREAK  ***************************************

//*************************************** FUNCTION HEADER  ***************************************
void Header(ofstream &Outfile)
{
        // Receives - The output file
        // Task - Prints the output preamble
        // Returns - The output file

    Outfile << setw(45) << "Adrian Beloqui ";
    Outfile << setw(15) << "CSC 36000";
    Outfile << setw(15) << "Section 11" << endl;
    Outfile << setw(50) << "Spring 2017";
    Outfile << setw(20) << "Assignment #6" << endl;
    Outfile << setw(35) << "-------------------------------------------------";
    Outfile << setw(35) << "-------------------------------------------------------------" << endl;
    return;
}
//***********************************  END OF FUNCTION HEADER  ***************************************

//*************************************** FUNCTION FOOTER  ***************************************
void Footer(ofstream &Outfile)
{
        // Receives - The output file
        // Task - Prints the output salutation
        // Returns - The output file

    Outfile << endl;
    Outfile << setw(35) << "-------------------------------------------------" << endl;
    Outfile << setw(35) << "|               END OF PROGRAM OUTPUT          |" << endl;
    Outfile << setw(35) << "-------------------------------------------------" << endl;
    return;
}
//***********************************  END OF FUNCTION FOOTER  ***************************************
```