```
//**************************************     PROGRAM IDENTIFICATION   **********************************
//*                                                                                                    *
//*     PROGRAM FILE NAME:   Program5.cpp           ASSIGNMENT  #:  5           Grade: _______       *
//*                                                                                                    *
//*     PROGRAM AUTHOR:       ____________________________________                                     *
//*                                    Adrian Beloqui                                                  *
//*                                                                                                    *
//*     COURSE #:   CSC  36000  11                                  DUE DATE:    Mar 31, 2017          *
//*                                                                                                    *
//****************************************************************************************************

//**************************************    PROGRAM DESCRIPTION   **************************************
//*                                                                                                    *
//*      PROCESS: This program is designed to read a file with up to 50 records of items in an inventory. *
//*               It is to sort the inventory three times using different algorithms and "keys" for    *
//*               sorting. Each time the inventory is sorted the inventory is to be printed. Also, it  *
//*               it is to print the initial list of items in the inventory as it is read from the file. *
//*                                                                                                    *
//*      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::Read - Reads an item from the input file.                      *
//*                     InventoryCLASS::Print - Prints all the items from the inventory.               *
//*                     InventoryCLASS::GetCurrentIndex - Gets the value of the current index.         *
//*                     InventoryCLASS::ExchangeSort - Sorts the inventory into descending order       *
//*                                                    according to the Quantity on Hand.              *
//*                     InventoryCLASS::ShellSort - Sorts the inventory into descending order according *
//*                                                    to the Selling Price.                           *
//*                     InventoryCLASS::QuickSort - Sorts the inventory into ascending order according *
//*                                                    to the Inventory Number.                        *
//*                                                                                                    *
//****************************************************************************************************

    //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 ItemTYPE{
    int number;
    char description[26];
    int quantity;
    int reorderNum;
    float cost;
    float price;
};

    //Definition of classes
class InventoryCLASS{
public:
        // Constructor
    InventoryCLASS() { currentIndex = 0; };

        // Functions
```

```
    void Read(ifstream &, int);
    void Print(ofstream &, int &);
    void ExchangeSort();
    void ShellSort();
    void QuickSort(int, int);
    int GetCurrentIndex() { return currentIndex; }

private:
    ItemTYPE stock[50];
    int currentIndex;
};

    //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, sentinel;
    bool endOfFile = false;
    InventoryCLASS inventory;

        // Open the input file
    InFile.open("data5.txt", ios::in);

        // Create the output file
    OutFile.open("output5.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 value that is used as sentinel
        InFile >> sentinel;
            // Check if the sentinel indicates the end of file
        if (sentinel != -999){
                // Read the input file and store the item
            inventory.Read(InFile, sentinel);
        }
        else {
                // Indicate the end of file
            endOfFile = true;
        }
    } while (NOT endOfFile);

        // Print the labels
    OutFile << "The Original Inventory Array:" << endl << endl;
    OutFile << "Inventory" << setw(7) << "Item" << setw(35) << "Quantity" << setw(12)
        << "Reorder" << setw(10) << "Cost of" << setw(12) << "Selling" << endl;
    OutFile << setw(7) << "Number" << setw(16) << "Description" << setw(27) << "on hand"
        << setw(12) << "Number" << setw(8) << "Item" << setw(13) << "Price" << endl;
    OutFile << "--------    -------------------------      --------     -------   -------" <<
```

```
        "     ------- " << endl;
        // Add amount of lines written into the output file
    linesWritten += 5;
        // Print the Inventory
    inventory.Print(OutFile, linesWritten);
        // Print page break
    PageBreak(OutFile, linesWritten);

        // Print the labels
    OutFile << "The Inventory Array sorted in descending order according to the Quantity"
        << " on Hand using the Exchange Sort:" << endl << endl;
    OutFile << "Inventory" << setw(7) << "Item" << setw(35) << "Quantity" << setw(12)
        << "Reorder" << setw(10) << "Cost of" << setw(12) << "Selling" << endl;
    OutFile << setw(7) << "Number" << setw(16) << "Description" << setw(27) << "on hand"
        << setw(12) << "Number" << setw(8) << "Item" << setw(13) << "Price" << endl;
    OutFile << "--------    -------------------------      --------     -------   -------" <<
        "     ------- " << endl;
        // Add amount of lines written into the output file
    linesWritten += 5;
        // Sort in descending order according to the Quantity on Hand using the Exchange Sort
    inventory.ExchangeSort();
        // Print the Inventory
    inventory.Print(OutFile, linesWritten);
        // Print page break
    PageBreak(OutFile, linesWritten);

        // Print the labels
    OutFile << "The Inventory Array sorted in descending order according to the Selling"
        << " Price using the Shell Sort:" << endl << endl;
    OutFile << "Inventory" << setw(7) << "Item" << setw(35) << "Quantity" << setw(12)
        << "Reorder" << setw(10) << "Cost of" << setw(12) << "Selling" << endl;
    OutFile << setw(7) << "Number" << setw(16) << "Description" << setw(27) << "on hand"
        << setw(12) << "Number" << setw(8) << "Item" << setw(13) << "Price" << endl;
    OutFile << "--------    -------------------------      --------     -------   -------" <<
        "     ------- " << endl;
        // Add amount of lines written into the output file
    linesWritten += 5;
        // Sort in descending order according to the Selling Price using the Shell Sort
    inventory.ShellSort();
        // Print the Inventory
    inventory.Print(OutFile, linesWritten);
        // Print page break
    PageBreak(OutFile, linesWritten);

        // Print the labels
    OutFile << "The Inventory Array sorted in ascending order according to the Inventory"
        << " Number using the Quick Sort:" << endl << endl;
    OutFile << "Inventory" << setw(7) << "Item" << setw(35) << "Quantity" << setw(12)
        << "Reorder" << setw(10) << "Cost of" << setw(12) << "Selling" << endl;
    OutFile << setw(7) << "Number" << setw(16) << "Description" << setw(27) << "on hand"
        << setw(12) << "Number" << setw(8) << "Item" << setw(13) << "Price" << endl;
    OutFile << "--------    -------------------------      --------     -------   -------" <<
        "     ------- " << endl;
        // Add amount of lines written into the output file
    linesWritten += 5;
        // Sort in ascending order according to the Inventory Number using the Quick Sort
    inventory.QuickSort(0, inventory.GetCurrentIndex()-1);
        // Print the Inventory
    inventory.Print(OutFile, linesWritten);
        // Print page break
    PageBreak(OutFile, linesWritten);

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

    return 0;
```

```
}
//***********************************  END OF FUNCTION MAIN  ****************************************

//************************************* FUNTION QUICKSORT  ****************************************
void InventoryCLASS::QuickSort(int First, int Last)
{
        // Receives - An integer that indicates the beginning of the array, and another integer that
        //            indicates the end of the array
        // Task - Sort the inventory items in ascending order according to the Inventory Number
        // Returns - Nothing

    int I, J, MID, Pivot_Value;
    ItemTYPE Temp;
    I = First;
    J = Last;
    MID = (First + Last) / 2;    // Find the middle indexed
    Pivot_Value = stock[MID].number; // Find the Pivot Point


    while (I <= J)
    {
        while (stock[I].number < Pivot_Value)
            ++I;
        while (stock[J].number > Pivot_Value)
            --J;
        if (I <= J)
        {
            Temp = stock[I];      //  switch  Item[I] and Item[J]
            stock[I] = stock[J];
            stock[J] = Temp;
            ++I;
            --J;
        }
    }

    if (J >= First)
    {
        QuickSort(First, J);        //  Partition left subarray
    }

    if (I <= Last)
    {
        QuickSort(I, Last);         //  Partition right subarray
    }

}
//***********************************  END OF FUNCTION QUICKSORT  ***********************************

//************************************* FUNTION SHELLSORT  ****************************************
void InventoryCLASS::ShellSort()
{
        // Receives - Nothing
        // Task - Sorts the inventory in descending order according to the Selling Price
        // Returns - Nothing

    int count = 0;
    int NumOfStages = 3;
    int KValues[3] = { 7, 3, 1 };
    int i, j, k, Stage;
    bool  Found;
    ItemTYPE Temp;

        // Execute the stages for the sorting algorithm
    for (Stage = 0; Stage < NumOfStages; Stage++)
    {
        k = KValues[Stage];
```

```
            // Traverse the subarrays
        for (i = k; i < currentIndex; i++)
        {
            Temp = stock[i];
            j = i - k;
            Found = false;
                // Compare the values in the subarrays
            while ((j >= 0) && (!Found))
            {
                if (Temp.price > stock[j].price)
                {
                    stock[j + k] = stock[j];
                    j -= k;
                }
                else {
                    Found = true;
                }
            }   // end while
            stock[j + k] = Temp;
        }
    }
}
//***********************************  END OF FUNCTION SHELLSORT  ***************************************

//*************************************** FUNTION EXCHANGESORT  ***************************************
void InventoryCLASS::ExchangeSort()
{
        // Receives - Nothing
        // Task - Sorts the inventory in descending order according to the Quantity on Hand
        // Returns - Nothing

    ItemTYPE temp;
    int  Max, IndexA, IndexB;
        // IndexA controls how many times we pass thru the
        // array the array
    for (IndexA = 0; IndexA < currentIndex; IndexA++)
    {
        Max = IndexA;
            // IndexB controls at which end of the array we begin and
            // which elements are compared and swapped if needed.
        for (IndexB = currentIndex - 1; IndexB > IndexA; IndexB--)
        {
                // Search for the maximum value for the quantity
            if (stock[IndexB].quantity > stock[Max].quantity){
                Max = IndexB;
            }
        }
            // Swap the values
        temp = stock[IndexA];
        stock[IndexA] = stock[Max];
        stock[Max] = temp;
    }
    return;

}
//***********************************  END OF FUNCTION EXCHANGESORT  ***********************************

//*************************************** FUNTION PRINT  ***************************************************
void InventoryCLASS::Print(ofstream &OutFile, int &linesWritten)
{
        // Receives - The output file and an integer that indicates the amount of lines written into the
        //            output file
        // Task - Print the items in the inventory
        // Returns - The output file and an integer that indicates the amount of lines written into the
        //           output file
```

```
        // Print each item in the inventory
    for (int i = 0; i < currentIndex; i++){
        OutFile << setw(6) << stock[i].number;
        OutFile << setw(31) << stock[i].description;
        OutFile << setw(12) << stock[i].quantity;
        OutFile << setw(12) << stock[i].reorderNum;
        OutFile << setw(12) << stock[i].cost;
        OutFile << setw(12) << stock[i].price << endl;
            // Add amount of lines written into the output file
        linesWritten += 1;
    }

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

//************************************** FUNTION READ  ********************************************
void InventoryCLASS::Read(ifstream &InFile, int inventoryNumber)
{
        // Receives - The input file and an integer
        // Task - Read the values from the input file into a ItemTYPE and add it to the inventory
        // Returns - The input file

        // Allocate memory for the new item
    ItemTYPE *item = new ItemTYPE();

        // Add the inventory number to the item
    item->number = inventoryNumber;
    InFile >> ws;
        // Read input line
    InFile.getline(item->description, 26);
        // Read values and store them into the item
    InFile >> item->quantity;
    InFile >> item->reorderNum;
    InFile >> item->cost;
    InFile >> item->price;

        // Add the item to the inventory
    stock[currentIndex++] = *item;

}
//***********************************  END OF FUNCTION READ  ****************************************

//************************************** 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 #5" << 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  ****************************************
```