///---------------------------------------------------------------  
/// File: SortedList.h  
/// Purpose: Header file for a demonstration of a sorted list  
/// implemented as an array.  
/// Programming Language: C++  
///---------------------------------------------------------------  
  
#include <iostream>  
using namespace std;  
  
/// Define a structure to use as the list item  
struct ListItem  
{  
 int key;  
 float theData;  
};  
  
#define MAX\_SIZE 50 /// Define maximum length of the list  
  
class SortedList  
{  
 private:  
 int head; /// Index to head of the list  
 ListItem theList[MAX\_SIZE]; /// The list  
  
 public:  
 SortedList(); /// Class constructor  
 ~SortedList(); /// Class destuctor  
 void ClearList(); /// Remove all items from the list  
 int Insert(int key, float f); /// Add an item to the list  
 int Delete(int key); /// Delete an item from the list  
 int Search(int key, float \*retVal); /// Search for an item in the list  
 int ListLength(); /// Return number of items in list  
 int isEmpty(); /// Return true if list is empty  
 int isFull(); /// Return true if list is full  
 void PrintList(); /// Print all items in the list  
};  
  
  
  
///---------------------------------------------------------------  
/// File: SortedList.cpp  
/// Purpose: Implementation file for a demonstration of a sorted  
/// list implemented as an array.  
/// Programming Language: C++  
///---------------------------------------------------------------  
///#include "SortedList.h"  
  
///--------------------------------------------  
/// Function: SortedList()  
/// Purpose: Class constructor  
/// Returns: void  
//--------------------------------------------  
SortedList::SortedList()  
{  
 head = -1;  
}  
  
///--------------------------------------------  
/// Function: SortedList()  
/// Purpose: Class destructor  
/// Returns: void  
///--------------------------------------------  
SortedList::~SortedList()  
{  
 /// Nothing to do here since the memory used  
 /// for the list is freed automatically.  
}  
  
///--------------------------------------------  
/// Function: ClearList()  
/// Purpose: Remove all items from the list  
/// Returns: void  
///--------------------------------------------  
void SortedList::ClearList()  
{  
 head = -1; // Reset count to start over  
}  
  
///--------------------------------------------  
/// Function: Insert()  
/// Purpose: Insert an item into the list in  
/// the next open slot.  
/// Returns: true if insertion was successful  
/// or false if the insertion failed.  
///--------------------------------------------  
int SortedList::Insert(int key, float f)  
{  
 int idx, i;  
  
 /// Check to see if the list is full  
 if(isFull()) return false;  
  
 /// Locate where to insert this item in the list  
 idx = 0;  
 while((idx <= head) && (theList[idx].key < key)) idx++;  
  
 /// Move all other items up 1 in list to make room for the  
 /// new item to be inserted in the correct place  
 head++; // Increment head index  
 for(i=head; i>idx; i--)  
 theList[i] = theList[i-1];  
  
 /// Insert the item into the list  
 theList[idx].key = key;  
 theList[idx].theData = f;  
 return true;  
}  
  
///--------------------------------------------  
/// Function: Delete()  
/// Purpose: Delete an item from the list and  
/// move all others up to close up the  
/// empty space.  
/// Returns: true if deletion was successful  
/// or false if the deletion failed.  
///--------------------------------------------  
int SortedList::Delete(int key)  
{  
 int i, d = 0;  
  
 /// Check for empty list  
 if(isEmpty()) return false;  
  
 /// Search the list for the item to delete  
 while((d <= head) && (key != theList[d].key))  
 {  
 d++;  
 }  
  
 /// Check to see if the item was found  
 if(d > head) return false; // Not found so return false  
 else  
 {  
 /// Move all other items toward the front of the array  
 /// This also overwrites and "deletes" the task searched for  
 for(i = d; i < head; i++)  
 {  
 theList[i] = theList[i+1]; // Using whole structure copy  
 }  
 head--; /// Reset head  
 }  
 return true;  
}  
  
  
///--------------------------------------------  
/// Function: Search()  
/// Purpose: Search for an item by key and copy  
/// the value into the variable pointed to  
/// by \*retVal.  
/// Returns: true if search was successful  
/// or false if the search failed.  
///--------------------------------------------  
int SortedList::Search(int key, float \*retVal)  
{  
 int s = 0;  
  
 while((s <= head) && (key != theList[s].key))  
 {  
 s++;  
 }  
  
 /// If item not found return false  
 if(s > head) return false;  
 else  
 \*retVal = theList[s].theData; // Copy the data  
 return true;  
}  
  
///--------------------------------------------  
/// Function: ListLength()  
/// Purpose: Return the number of items in the  
/// list.  
/// Returns: Number of items in list.  
/// Note: head is the index of the last filled  
/// slot. To get the number of items you must  
/// add 1 to the last filled index.  
///--------------------------------------------  
int SortedList::ListLength()  
{  
 return head+1;  
}  
  
///--------------------------------------------  
/// Function: isEmpty()  
/// Purpose: Return true if the list is empty  
/// Returns: true if empty, otherwise false  
/// Note: C has no boolean data type so we use  
/// the defined int values for true and false  
/// instead.  
///--------------------------------------------  
int SortedList::isEmpty()  
{  
 return (head == -1);  
}  
  
///--------------------------------------------  
/// Function: isFull()  
/// Purpose: Return true if the list is full  
/// Returns: true if full, otherwise false  
/// Note: C has no boolean data type so we use  
/// the defined int values for true and false  
/// instead.  
///--------------------------------------------  
int SortedList::isFull()  
{  
 return (head >= MAX\_SIZE);  
}  
  
  
///--------------------------------------------  
/// Function: PrintList()  
/// Purpose: Print all items in the list with  
/// their priority.  
/// Returns: void  
///--------------------------------------------  
void SortedList::PrintList()  
{  
 int i;  
  
 cout << "\n\nItems in the List\n";  
 cout << "-----------------------------------------------------------\n";  
 cout << "Key\t\tData\n";  
 cout << "-----------------------------------------------------------\n";  
  
 for(i=0; i<=head; i++)  
 {  
 cout << theList[i].key << "\t\t" << theList[i].theData << "\n";  
 }  
 cout << "-----------------------------------------------------------\n\n";  
}  
///\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_  
///Main file used to test the list class  
//---------------------------------------------------------------  
// File: ListMain.cpp  
// Purpose: Main file with tests for a demonstration of a sorted  
// list implemented as an array.  
// Programming Language: C++  
//---------------------------------------------------------------  
///#include "SortedList.h"  
  
int main()  
{  
 float f;  
 SortedList \*theList;  
  
 cout << "Simple List Demonstration\n";  
 cout << "Create a list and add a few tasks to the list";  
  
 theList = new SortedList(); /// Instantiate a list object  
  
 theList->Insert(5, 3.1f); /// Note: The argument to the funtion should be a float  
 theList->Insert(1, 5.6f); /// A constant real number like 3.1 is interpreted as  
 theList->Insert(3, 8.3f); /// a double unless it is explicitly defined as a float  
 theList->Insert(2, 7.4f); /// by adding an 'f' to the end of the number.  
 theList->Insert(4, 2.5f);  
  
 /// Show what is in the list  
 theList->PrintList();  
  
 /// Test the list length function  
 cout << "\nList now contains " << theList->ListLength() << "items.\n\n";  
  
 /// Test delete function  
 cout << "Testing delete of last item in list.\n";  
 theList->Delete(5);  
 theList->PrintList();  
  
 /// Test delete function  
 cout << "Testing delete of first item in list.\n";  
 theList->Delete(1);  
 theList->PrintList();  
  
 /// Test delete function  
 cout << "Testing delete of a middle item in list.\n";  
 theList->Delete(3);  
 theList->PrintList();  
  
 /// Test delete function with a known failure argument  
 cout << "Testing failure in delete function.\n";  
 if(theList->Delete(5))  
 cout << "Oops! Should not have been able to delete.\n\n";  
 else  
 cout << "Unable to locate item to delete.\n\n";  
  
 /// Test search (known failure)  
 cout << "Testing Search function. Search for key 3\n";  
 if(theList->Search(3, &f))  
 cout << "Search result: theData = " << f << "\n\n";  
 else  
 cout << "Search result: Unable to locate item in list\n\n";  
  
 /// Test search (known success)  
 cout << "Testing Search function. Search for key 2\n";  
 if(theList->Search(2, &f))  
 cout << "Search result: theData = " << f << "\n";  
 else  
 cout << "Search result: Unable to locate item in list\n";  
  
 cout << "\n\nEnd list demonstration...";  
  
 return 0;  
}