**CSCI 1730 - Programming Assignment 5 - 100 pts.**

**Due Date: May 8, 2019**

1. Write a **class** **template** List for modeling a list of five elements which will have values from one of four different data types (int, float, char, Distance). This class should support these list manipulation tasks:

* Initialize a list to “zero” values.
* Initialize a list to the values stored in a list of the same type.
* Allow for user input of a list.
* Display a list.
* Sort a list into ascending order.

Then, write a **template function** demo, that will accept a flag of the type data you would like and then create a list of that type to demonstrate the list class – it will do these tasks:

* Create a list of the selected type and display it, showing the initialized “zero” values it contains.
* Have the user enter values into the list and then display the values.
* Create a second list of the selected type, initializing it to the values stored in the first list, and then display the second list.
* Sort the values in the first list.
* Display the sorted list.

**Program Requirements:**

* To perform the sorting, use either a **selection** or **bubble sort algorithm**. See the document posted with the assignments “sort and search.doc” for details on these algorithms.
* The Distance class is given below:

class Distance //English Distance class

{

private:

int feet;

float inches;

public:

Distance() : feet(0), inches(0.0) //constructor (no args)

{ }

//constructor (two args)

Distance(int ft, float in) : feet(ft), inches(in)

{ }

Distance( float fltfeet ) //constructor (one arg)

{ //convert float to Distance

feet = int(fltfeet); //feet is integer part

inches = 12\*(fltfeet-feet); //inches is what's left

}

bool operator < (Distance) const; //compare distances

friend istream& operator >> (istream& s, Distance& d);

friend ostream& operator << (ostream& s, Distance& d);

};

bool Distance::operator < (Distance d2) const

{

float bf1 = feet + inches/12;

float bf2 = d2.feet + d2.inches/12;

return (bf1 < bf2) ? true : false;

}

//--------------------------------------------------------------

istream& operator >> (istream& s, Distance& d) //get Distance

{ //from user

cout << "\nEnter feet: "; s >> d.feet; //using

cout << "Enter inches: "; s >> d.inches; //overloaded

return s; //>> operator

}

//--------------------------------------------------------------

ostream& operator << (ostream& s, Distance& d) //display

{ //Distance

s << d.feet << "\'-" << d.inches << '\"'; //using

return s; //overloaded

} //<< operator

**Note:** A char value can be initialized to zero – when displayed, no output appears.

To test your templates, try using the following main driver:

int main()

{

int sel;

bool end=false;

int iFlag=0;

float fFlag=0;

char cFlag=0;

Distance dFlag;

cout << "TEMPLATE DEMO PROGRAM\n";

do{

cout << "Enter list type (1=int 2=float 3=char 4=Distance 5=exit): ";

cin >> sel;

switch (sel)

{

case 1:

demo(iFlag);

break;

case 2:

demo(fFlag);

break;

case 3:

demo(cFlag);

break;

case 4:

demo(dFlag);

break;

default:

end=true;

cout << "Bye...\n";

break;

}

}while(!end);

return 0;

}

**Note:** To see how this program works, copy [S:\Coursework\Liu\a6-1.exe](file:///S:\Coursework\Liu\a6-1.exe) to your own folder to run the program.

Here is a sample run of the program (user input is bold):

TEMPLATE DEMO PROGRAM

Enter list type (1=int 2=float 3=char 4=Distance 5=exit): **1**

New blank list created

List values -> 0 0 0 0 0

Enter values into the list

Enter element 1: **5**

Enter element 2: **4**

Enter element 3: **3**

Enter element 4: **2**

Enter element 5: **1**

List entered -> 5 4 3 2 1

Create a second list initialized to the first

List created -> 5 4 3 2 1

Sort the first list

Sorted list --> 1 2 3 4 5

Enter list type (1=int 2=float 3=char 4=Distance 5=exit): **2**

New blank list created

List values -> 0 0 0 0 0

Enter values into the list

Enter element 1: **5.6**

Enter element 2: **-3.4**

Enter element 3: **-6.8**

Enter element 4: **10.6**

Enter element 5: **3.9**

List entered -> 5.6 -3.4 -6.8 10.6 3.9

Create a second list initialized to the first

List created -> 5.6 -3.4 -6.8 10.6 3.9

Sort the first list

Sorted list --> -6.8 -3.4 3.9 5.6 10.6

Enter list type (1=int 2=float 3=char 4=Distance 5=exit): **3**

New blank list created

List values ->

Enter values into the list

Enter element 1: **p**

Enter element 2: **z**

Enter element 3: **o**

Enter element 4: **b**

Enter element 5: **w**

List entered -> p z o b w

Create a second list initialized to the first

List created -> p z o b w

Sort the first list

Sorted list --> b o p w z

Enter list type (1=int 2=float 3=char 4=Distance 5=exit): **4**

New blank list created

List values -> 0'-0" 0'-0" 0'-0" 0'-0" 0'-0"

Enter values into the list

Enter element 1:

Enter feet: **7**

Enter inches: **5**

Enter element 2:

Enter feet: **2**

Enter inches: **11**

Enter element 3:

Enter feet: **10**

Enter inches: **4**

Enter element 4:

Enter feet: **3**

Enter inches: **10**

Enter element 5:

Enter feet: **8**

Enter inches: **9**

List entered -> 7'-5" 2'-11" 10'-4" 3'-10" 8'-9"

Create a second list initialized to the first

List created -> 7'-5" 2'-11" 10'-4" 3'-10" 8'-9"

Sort the first list

Sorted list --> 2'-11" 3'-10" 7'-5" 8'-9" 10'-4"

Enter list type (1=int 2=float 3=char 4=Distance 5=exit):

2. Write a **class** LinkList, which implements a **sorted** linked list of floats. The class should have member functions that handle the following tasks:

* Initialize a new linked list to being empty.
* Add a float value to the list – this function should add the value to the list so that the list is always in sorted order from smallest to largest value.
* Display the list.
* Find a float value in the list and return the **position number** of the value in the list (assume position number **counting starts with one**). If the value is not found, this function should return zero.
* Find the ***n*th link** in the list and **return the address of the link**. If the list does not have an *n*th link, or if the list is empty, this function should return NULL.
* Delete the ***n*th link** in the list. If *n* is more than the number of links in the list, delete nothing, and give the user an error message.
* Show these list statistics – the total number of values in the list, the mean value, and the median value. Note: If there are an odd number of values in the list, the median is the middle value; otherwise, it is the average of the two middle values.

**Note**: **The structure** you create to represent the nodes of your linked list should contain **only two member variables**: **a variable to store a floating-point number** and **a pointer to the structure**.

**Note**: The **LinkList class** you create should contain **only one member variable**: one **pointer** to the structure for holding the address of the first item in the linked list.

Using your LinkList class, write a **C++ program** that creates one LinkList object and then repeatedly offers the user these options for working on this linked list:

1. Add a value to the list – the value is added so that the list is **sorted from smallest to largest**.
2. Search for a value in the list – if found, displays the position number of the value in the list; if not found, displays a message indicating this.
3. Display the *n*th value in the list – if there is no *n*th value, display a message indicating this.
4. Delete the *n*th value in the list – if there is no *n*th value, display a message indicating this.
5. Display list statistics (count, mean, median).
6. Display the entire list.
7. Exit the program.

**Note:** To see how this program works, copy [S:\Coursework\Liu\a6-2.exe](file:///S:\Coursework\Liu\a6-2.exe) to your own folder to run the program.

Here is a sample run of the program (user input is in bold):

LINKED LIST MANAGER

Make a selection:

1 - Add a value (a decimal number)

2 - Search for a value

3 - Find the nth value

4 - Delete the nth value

5 - Display list statistics (count, mean, median)

6 - Display the list

7 - Exit the program

=> **1**

Enter data value to add: **3.14159**

Make a selection:

1 - Add a value (a decimal number)

2 - Search for a value

3 - Find the nth value

4 - Delete the nth value

5 - Display list statistics (count, mean, median)

6 - Display the list

7 - Exit the program

=> **6**

3.14159

Make a selection:

1 - Add a value (a decimal number)

2 - Search for a value

3 - Find the nth value

4 - Delete the nth value

5 - Display list statistics (count, mean, median)

6 - Display the list

7 - Exit the program

=> **1**

Enter data value to add: **-3.14159**

Make a selection:

1 - Add a value (a decimal number)

2 - Search for a value

3 - Find the nth value

4 - Delete the nth value

5 - Display list statistics (count, mean, median)

6 - Display the list

7 - Exit the program

=> **1**

Enter data value to add: **7.128**

Make a selection:

1 - Add a value (a decimal number)

2 - Search for a value

3 - Find the nth value

4 - Delete the nth value

5 - Display list statistics (count, mean, median)

6 - Display the list

7 - Exit the program

=> **1**

Enter data value to add: **-7.128**

Make a selection:

1 - Add a value (a decimal number)

2 - Search for a value

3 - Find the nth value

4 - Delete the nth value

5 - Display list statistics (count, mean, median)

6 - Display the list

7 - Exit the program

=> **1**

Enter data value to add: **1.14159**

Make a selection:

1 - Add a value (a decimal number)

2 - Search for a value

3 - Find the nth value

4 - Delete the nth value

5 - Display list statistics (count, mean, median)

6 - Display the list

7 - Exit the program

=> **1**

Enter data value to add: **1.73**

Make a selection:

1 - Add a value (a decimal number)

2 - Search for a value

3 - Find the nth value

4 - Delete the nth value

5 - Display list statistics (count, mean, median)

6 - Display the list

7 - Exit the program

=> **1**

Enter data value to add: **-1.73**

Make a selection:

1 - Add a value (a decimal number)

2 - Search for a value

3 - Find the nth value

4 - Delete the nth value

5 - Display list statistics (count, mean, median)

6 - Display the list

7 - Exit the program

=> **6**

-7.128

-3.14159

-1.73

1.14159

1.73

3.14159

7.128

Make a selection:

1 - Add a value (a decimal number)

2 - Search for a value

3 - Find the nth value

4 - Delete the nth value

5 - Display list statistics (count, mean, median)

6 - Display the list

7 - Exit the program

=> **2**

Enter data value to search for: **5**

5 is not in the list

Make a selection:

1 - Add a value (a decimal number)

2 - Search for a value

3 - Find the nth value

4 - Delete the nth value

5 - Display list statistics (count, mean, median)

6 - Display the list

7 - Exit the program

=> **2**

Enter data value to search for: **1.73**

1.73 is in position 5

Make a selection:

1 - Add a value (a decimal number)

2 - Search for a value

3 - Find the nth value

4 - Delete the nth value

5 - Display list statistics (count, mean, median)

6 - Display the list

7 - Exit the program

=> **3**

Enter position number to find: **6**

Value of position 6 is 3.14159

Make a selection:

1 - Add a value (a decimal number)

2 - Search for a value

3 - Find the nth value

4 - Delete the nth value

5 - Display list statistics (count, mean, median)

6 - Display the list

7 - Exit the program

=> **4**

Enter item position number to delete: **5**

Make a selection:

1 - Add a value (a decimal number)

2 - Search for a value

3 - Find the nth value

4 - Delete the nth value

5 - Display list statistics (count, mean, median)

6 - Display the list

7 - Exit the program

=> **6**

-7.128

-3.14159

-1.73

1.14159

3.14159

7.128

Make a selection:

1 - Add a value (a decimal number)

2 - Search for a value

3 - Find the nth value

4 - Delete the nth value

5 - Display list statistics (count, mean, median)

6 - Display the list

7 - Exit the program

=> **5**

LIST STATISTICS

Number of items in list = 6

Average = -0.0980683

Median = -0.294205

Make a selection:

1 - Add a value (a decimal number)

2 - Search for a value

3 - Find the nth value

4 - Delete the nth value

5 - Display list statistics (count, mean, median)

6 - Display the list

7 - Exit the program

=>

**What you need to turn in:**

1. **Source code listing:** A printed copy of the source code for each problem. Remember to include the name of each group member in a comment at the top of your source code. Be sure to follow the “Code Style Guidelines” specified in class.
2. **Source code files:** E-mail me your source code as attachments.
3. **Working in Groups:** Every student will work with two other students in our class on this assignment; all members of the group must contribute to the solution. **Turn in only one copy of the solution** – **clearly identify the name of each member on everything that you turn in.**
4. **Late Assignments:** Assignments are due before class on the specified due date (both the paper copies and the e-mail copies). If you wish to turn in the paper copy of an assignment after class, place them under my office door. Assignments turned in late will be assessed a 20% penalty per class day late.