```
/*//
  ------------------------------------------------------------------------------
  ----
Justin Dang
Student ID: 1148267
//
  ------------------------------------------------------------------------------
  ------
FUNCTION OF THE FOLLOWING CODE>>

- Reads the names and weights of 15 people

- Uses a doubly linked list to organize by names(alphabetical) and weight
  (ascending order)

- Doubly linked list contains a node with two pointers; one pointer is for names
  in order, the other for
  weight in order

- Sorts the input into the list, meaning there should be no sort function ever
  called

- Prints out a ordered list in asccending order by name and weight.
//
  ------------------------------------------------------------------------------
  ------
WORKS CITED>>

Idea for how to enter nodes in the middle of a queue: https://
  www.geeksforgeeks.org/insert-node-middle-linked-list/

First time using std::string, learned from: http://www.cplusplus.com/reference/
  string/string/
*///
  ------------------------------------------------------------------------------
  ----
#include <iostream>
#include <string>
using namespace std;

// represents each person
class Node {
public:
    int weight;                    // Weight of each person.
    string name;                   // Name of each person.
    Node* nextWeight;              // address of next Weight Node.
    Node* nextName;                // address of next Name Node.

    // node(int, *weightPtr, *namePtr)
    Node(int info, string info1, Node* namePtr = 0, Node* weightPtr = 0) { //
      Structure for each node
        weight = info;
```

```
39          name = info1;
40          nextWeight = weightPtr;
41          nextName = namePtr;
42      }
43  };
44
45  class DoublyLinked {
46  private:
47      Node* headWeight, * headName;                           // Reference to the start of our "Queue's".
48  public:
49      DoublyLinked() { headWeight = headName = 0; }           // Constructs our list by setting our heads to 0.
50      /*
51      add(string, int)
52      Takes in a string as a user's name and an int for their weight. Sorts the list during input
53      */
54      void add(string data1, int data) {
55          Node* newPerson = new Node(data, data1);            // Creates a new node with data of our new
56                                                              // user.
57          Node* tempHeadWeight = headWeight;                  // Creates a point to our head since we do not want to alter our head here.
58          int targetNode = 0, count = 0;                      // Used to find our "node of interest"(described further into code).
59
60          /*
61          Setting our person's weight into our list
62          ----------------------------------------------------
63          */
64
65
66          if (headWeight == 0) {                              // If our list is empty we just enter the person into our list.
67              headWeight = newPerson;
68          }
69          else {                                              // Otherwise we run an algorithm to find our "node of interest" by counting
70              while (tempHeadWeight->nextWeight != 0)                                 // with integers for every node we pass.
71                  {
                     if (tempHeadWeight->weight > data) {       // We traverse our list and find a node containing a weight larger than
72                       if (targetNode > 0)                    // our new person's.

73                           targetNode--;                      // We then select the node before it(due to inserting behavior for a queue).
74                       break;
75                   }
76                   tempHeadWeight = tempHeadWeight->nextWeight;
77                   targetNode++;
```

```
78                      if (targetNode > 0)
79                  if (tempHeadWeight->nextWeight == 0 && tempHeadWeight->weight >
                      data) {
80                          targetNode--;                       // Handles
                        subtraction if we are adding second to last in our queue
81                  }
82              }
83
84              tempHeadWeight = headWeight;                    // Reset our
                  tempHeadWeight for traversal through the list again.
85
86              while (count < targetNode) {                    // We find the
                  translate our "Node of interest" into an actual Node we can access
                  here.
87                  count++;
88                  tempHeadWeight = tempHeadWeight->nextWeight; // Stores the "node
                      of interest" in tempHeadWeight.
89              }
90
91              if (targetNode == 0 && data < headWeight->weight) {     // We place
                  our new Person at the head of our list
92                  newPerson->nextWeight = headWeight;
93                  headWeight = newPerson;
94              }
95              else if (tempHeadWeight->nextWeight == 0) {             // We place
                  our new Person at the end of our list
96                  tempHeadWeight->nextWeight = newPerson;
97              }
98              else {                                                  // We place
                  our person in the middle of our list(middle-> between two nodes).
99                  newPerson->nextWeight = tempHeadWeight->nextWeight;
100                 tempHeadWeight->nextWeight = newPerson;
101             }
102         }
103         //END(sorting by weight into our list)
104         //-----------------------------------------------------
105
106         /*
107         Setting our person's name into our list
108         -------------------------------------------------------
109         */
110
111         Node* tempHeadName = headName;
112         targetNode = count = 0;
113
114
115         if (headName == 0) {                                    // If our list is
              empty we just enter the person into our list.
116             headName = newPerson;
117         }
118         else {                                                  // Otherwise we run
              an algorithm to find our "node of interest" by counting
```

```
119             while (tempHeadName->nextName != 0) {          // with integers for
                  every node we pass.
120                 if (tempHeadName->name > data1) {          // We traverse our
                      list and find a node containing a name larger than
121                     if (targetNode > 0)                    // our new person's.
122                         targetNode--;                      // We then select the
                        node before it(due to inserting behavior for a queue).
123                     break;
124                 }
125                 tempHeadName = tempHeadName->nextName;
126                 targetNode++;
127                 if (tempHeadName->nextName == 0 && tempHeadName->name > data1)
                      {          // Handles subtraction if we are adding second to
                      last in our queue
128                     if (targetNode > 0)
129                         targetNode--;
130                 }
131             }
132             tempHeadName = headName;                       // Reset our
                  tempHeadName for traversal through the list again.
133
134             while (count < targetNode) {                   // We find the
                  translate our "Node of interest" into an actual Node we can access
                  here.
135                 count++;
136                 tempHeadName = tempHeadName->nextName;     // Stores the "node
                      of interest" in tempHeadName.
137             }
138
139             if (targetNode == 0 && data1 < headName->name) {// We place our new
                  Person at the head of our list
140                 newPerson->nextName = headName;
141                 headName = newPerson;
142             }
143             else if (tempHeadName->nextName == 0) {        // We place our new
                  Person at the end of our list
144                 tempHeadName->nextName = newPerson;
145             }
146             else {                                         // We place our
                  person in the middle of our list(middle-> between two nodes).
147                 newPerson->nextName = tempHeadName->nextName;
148                 tempHeadName->nextName = newPerson;
149             }
150         }
151         // END(sorting by name into our list)
152         //-----------------------------------------------------
153     }// END(AddNode)
154
155     void printOrderNames()
156     {
157         cout << "Names & weights sorted(ascending) by name. : ";
158         for (Node* temp = headName; temp != 0; temp = temp->nextName)       //
```

```
            Traverses through name list, printing each node's name and weight
159             cout << temp->name << " - " << temp->weight << ", ";
160         cout << endl;
161     }
162     void printOrderWeight()
163     {
164         cout << "Names & weights sorted(ascending) by weight. : ";
165         for (Node* temp = headWeight; temp != 0; temp = temp->nextWeight)   //
            Traverses through name list, printing each node's name and weight
166             cout << temp->name << " - " << temp->weight << ", ";
167         cout << endl;
168     }
169
170 };
171
172 int main()
173 {
174     int inputWeight;                                        // Stores user's
        weight here(temp)
175     string inputName;                                       // Stores user's
        name here(temp)
176     DoublyLinked list;
177     for (int x = 1; x < 16; x++) {                          // takes in 15
        users
178         cout << "Please enter user" << x << "'s name: ";
179         getline(cin, inputName);
180         cout << "\nPlease enter user" << x << "'s weight: ";
181         cin >> inputWeight;
182         cin.ignore();
183         list.add(inputName, inputWeight);
184         cout << "\n\n";
185     }
186     list.printOrderNames();
187     list.printOrderWeight();
188 }
189 /*//------------------------------------------------------------Case 1:
190 Please enter user1's name: Mark
191
192 Please enter user1's weight: 150
193
194
195 Please enter user2's name: Tina
196
197 Please enter user2's weight: 115
198
199
200 Please enter user3's name: Zach
201
202 Please enter user3's weight: 55
203
204
205 Please enter user4's name: Amy
```

```
206
207  Please enter user4's weight: 140
208
209
210  Please enter user5's name: Steve
211
212  Please enter user5's weight: 220
213
214
215  Please enter user6's name: Brian
216
217  Please enter user6's weight: 250
218
219
220  Please enter user7's name: Liz
221
222  Please enter user7's weight: 125
223
224
225  Please enter user8's name: Brian
226
227  Please enter user8's weight: 220
228
229
230  Please enter user9's name: Laura
231
232  Please enter user9's weight: 115
233
234
235  Please enter user10's name: Alex
236
237  Please enter user10's weight: 175
238
239
240  Please enter user11's name: Jason
241
242  Please enter user11's weight: 210
243
244
245  Please enter user12's name: Eric
246
247  Please enter user12's weight: 175
248
249
250  Please enter user13's name: Aaron
251
252  Please enter user13's weight: 195
253
254
255  Please enter user14's name: Kim
256
257  Please enter user14's weight: 135
```

```
258
259
260 Please enter user15's name: Brandon
261
262 Please enter user15's weight: 78
263
264
265 Names & weights sorted(ascending) by name. : Aaron - 195, Alex - 175, Amy - 140,
      Brandon - 78, Brian - 250, Brian - 220, Eric - 175, Jason - 210, Kim - 135,
      Laura - 115, Liz - 125, Mark - 150, Steve - 220, Tina - 115, Zach - 55,
266 Names & weights sorted(ascending) by weight. : Zach - 55, Brandon - 78, Tina -
      115, Laura - 115, Liz - 125, Kim - 135, Amy - 140, Mark - 150, Alex - 175, Eric
       - 175, Aaron - 195, Jason - 210, Steve - 220, Brian - 220, Brian - 250,
267 *///------------------------------------------------------------
268
```