COLEMAN UNIVERSITY

NAME: Amrit Panesar

STUDENT #: 77260

CLASS #: COM 390

8-12AM_____ **1-5PM**_____ **6-10PM** X

PROJECT: 5b - Linked List

INSTRUCTOR: Scott Mayer

```c
#include <stdlib.h>
#include <string.h>

#ifndef boolean

typedef int bool_t;

#define boolean bool_t

#define true 1
#define false 0

#endif
```

```
1
2   //include node manager
3   #include "NodeManager.h"
4
5   // standard libraries
6   #include <stdlib.h>
7   #include <stdio.h>
8   #include <conio.h>
9   #include <Windows.h>
```

```c
#include "Lab3.h"

int main(int argc, char** argv) {
    NodeManagerP m = NewNodeManager();
}
```

```
1   #include "Node.h"
2   #include "common.h"
3
4   #ifndef H_NODEMANAGER
5   #define H_NODEMANAGER
6
7   typedef struct {
8   private:
9       int allocatedCount;
10      int nodeCount;
11
12  public:
13      NodeP *Nodes;
14      NodeP startNode;
15
16      void Init(int initialSize);
17      int  Size();
18      int  Length();
19
20      void    Add(int aIn, float bIn, char *nameIn);
21      NodeP   GetAt(int index);
22      boolean InsertAt(int index, NodeP nodeIn);
23      boolean DeleteAt(int index);
24
25      void Sort();
26      boolean Swap(int id1, int id2);
27
28      void Destroy();
29      void ReorderRefrence();
30
31  } NodeManager, *NodeManagerP;
32
33  #define N_INITIALSIZE 10
34  NodeManagerP NewNodeManager(int size = N_INITIALSIZE);
35
36  #endif
```

```
1   #include "NodeManager.h"
2
3   void NodeManager::Init(int initialSize) {
4       Nodes = (NodeP*)malloc(sizeof(NodeP) * initialSize);
5   }
6
7   int NodeManager::Length() {
8       return nodeCount;
9   }
10
11  int NodeManager::Size() {
12      return allocatedCount;
13  }
14
15  void NodeManager::Sort() {
16      // A-Za-z0-9 sort
17      int i = 0;
18
19      if (Size() < 2) {
20          return;
21      }
22
23      for (i = 0; i < Size()-1; i++) {
24          if (strcmp(Nodes[i]->name, Nodes[i+1]->name) > 0) {
25              Swap(i, i+1);
26          }
27      }
28  }
29
30  void NodeManager::ReorderRefrence() {
31      int i;
32      NodeP curNode = startNode;
33      for (i = 0; i < nodeCount; i++) {
34          Nodes[i] = curNode;
35          curNode = curNode->right;
36      }
37  }
38
39  boolean NodeManager::Swap(int id1, int id2) {
40      NodeP one, oneL, oneR;
41      NodeP two, twoL, twoR;
42
43      one = GetAt(id1);
44      two = GetAt(id2);
45      if (one == NULL || two == NULL) {
46          return false;
47      }
48
49      //left node
50      oneL = one->left;
51      oneR = one->right;
52
53      //right node
```

```
54        twoL = two->left;
55        twoR = two->right;
56
57        //swap
58        one->left = twoL;
59        one->right = twoR;
60        two->left = oneL;
61        two->right = twoR;
62
63        return true;
64    }
65
66    NodeP NodeManager::GetAt(int index) {
67        int i;
68        NodeP curNode;
69        if (index > nodeCount) {
70            return NULL;
71        }
72        for (i = 0; i < nodeCount; i++) {
73            curNode = Nodes[0];
74        }
75        return curNode;
76    }
77
78    boolean NodeManager::InsertAt(int index, NodeP nodeIn) {
79        int i;
80        NodeP leftNode;
81        NodeP curNode;
82        NodeP rightNode;
83
84        // make more room if we don't have any more
85        if (nodeCount + 1 > allocatedCount) {
86            Nodes = (NodeP*)realloc(Nodes, sizeof(Node) * (nodeCount + (nodeCount /2)));
87        }
88
89        curNode = GetAt(index);
90        leftNode = curNode->left;
91        rightNode = curNode->right;
92
93        nodeCount++;
94    }
95
96    boolean NodeManager::DeleteAt(int index) {
97        NodeP node = GetAt(index);
98        if (node == NULL) {
99            return false;
100       }
101
102       NodeP left = node->left;
103       NodeP right = node->right;
104
105       left->right = right;
106       right->left = left;
```

```
107
108         free(node);
109         return true;
110     }
111
112     void NodeManager::Destroy() {
113         int i;
114         for (i = 0; i < Length() - 1; i++) {
115             free(Nodes[i]);
116         }
117     }
118
119     void NodeManager::Add(int aIn, float bIn, char *nameIn) {
120         NodeP out;
121
122         out = (NodeP)malloc(sizeof(Node));
123         out->New(aIn, bIn, nameIn);
124         InsertAt(nodeCount - 1, out);
125     }
126
127     NodeManagerP NewNodeManager(int size) {
128         NodeManagerP nodem;
129
130         nodem = (NodeManagerP)malloc(sizeof(NodeManager));
131         nodem->Init(size);
132
133         return nodem;
134     }
```

```h
#include "common.h"

#ifndef H_NODE
#define H_NODE

typedef struct {
    int a;
    float b;
    char *name;

    NodeP left;
    NodeP right;
    NodeP self;

    void New(int aIn, float bIn, char *nameIn);

}Node, *NodeP;

#endif
```

```
1  #include "Node.h"
2
3  void Node::New(int aIn, float bIn, char *nameIn) {
4      a = aIn;
5      b = bIn;
6      name = nameIn;
7  }
8
```