Name: **Muhammad Anas Baig**

Enrollment No.: **01-134152-037**

Section: **BS(CS)-4A**

LAB-JOURNAL-5

Exercise 1:

Implement the class Linked List to create a list of integers. You need to provide the implementation of the member functions as described in the following.

```
class List
{
private:
    Node * head;
public:
    List();
    ~List();

    // Checks if the list is empty or not
    bool emptyList();

    // Inserts a new node with value ‘newV’ after the node containing value ‘oldV’. If a node with value ‘oldV’ does not exist, inserts the new node at the end.
    void insertafter(int oldV, int newV);

    // Deletes the node containing the specified value
    void deleteNode(int value);

    // Inserts a new node at the start of the list
    void insert_begin(int value);

    // Inserts a new node at the end of the list
    void insert_end(int value);

    // Displays the values stored in the list
    void traverse();
};
```

Solution:

node.h File:

```
#pragma once
class Node
{
public:
    int data;
    Node *next;
public:
    Node(void);
};
```

node.cpp File:

```
#include "Node.h"


Node::Node(void)
```

```
{
}
```

list.h File:

```
#include "Node.h"
#pragma once
class List
{
public:  //we took this public to check midInsertNodeWhereAddress() function because we passed address of ‘head’ from main() in this method
Node *head;
public:
    List(void);
    bool isEmpty();

    void beginInsert(int);
    void midInsertWhereData(int, int); //insert to same location where data == X
    void midInsertNodeToNextWhereData(int, int); //insert Node next to location where data == X
    void midInsertWhereAddress(int, Node *); //insert to same location where address == Y
    void midInsertNodeToNextWhereAddress(int, Node *); //insert Node next to location where address == Y
    void midInsertWhereCount(int, int); //count nodes and inset to count==Z
    void midInsertNodeToNextWhereCount(int, int); //count nodes and insert Node next to count == Z
    void endInsert(int);

    void findNode(int);

    void beginDelete();
    void midDelete(int); //delete node with Data Value == Z
    void endDelete();

    void displayList();
};
```

list.cpp File:

```
#include "List.h"
#include "conio.h"
#include "Node.h"
#include <iostream>
using namespace std;

List::List(void)
{
    head = new Node;
    head->next = '\0';
}

bool List::isEmpty()
{
    if(head->next == '\0')
    {
        return true;
    }
    else
    {
        return false;
    }
}

void List::beginInsert(int value) //inserts node at beginning
{
    Node *ptr;
    ptr = new Node;
    ptr->data = 0;
    ptr->next = '\0';

```

```
32.     ptr->data = value;
33.
34.     ptr->next = head->next;
35.     head->next = ptr;
36. }
37.
38. void List::midInsertWhereData(int value, int dataLocation) //insert to same location where data == X
39. {
40.     if(!isEmpty())
41.     {
42.         Node *tempPtr;
43.         tempPtr = head->next;
44.
45.         while(tempPtr->data != dataLocation && tempPtr->next != '\0')
46.         {
47.             tempPtr = tempPtr->next;
48.         }
49.             if(tempPtr->data == dataLocation) //if data is found in node in list
50.             {
51.                 tempPtr->data = value;
52.             }
53.             else
54.             {
55.                 cout<<"Sorry!!! No Node found with Data Value = "<<dataLocation<<"."<<endl;
56.             }
57.     }
58.     else
59.     {
60.         cout<<"Sorry!!! No data inserted-No value found-List is empty"<<endl;
61.     }
62. }
63.
64. void List::midInsertNodeToNextWhereData(int value, int dataLocation) //insert Node next to location where data ==
    X
65. {
66.     if(!isEmpty())
67.     {
68.         Node *tempPtr;
69.         tempPtr = head;
70.
71.         while(tempPtr->data != dataLocation && tempPtr->next != '\0')
72.         {
73.             tempPtr = tempPtr->next;
74.         }
75.             if(tempPtr->data == dataLocation) //if data is found in node in list
76.             {
77.                 Node *ptr;
78.                 ptr = new Node;
79.                 ptr->data = 0;
80.                 ptr->next = '\0';
81.
82.                 ptr->data = value;
83.                 ptr->next = tempPtr->next;
84.                 tempPtr->next = ptr;
85.             }
86.             else
87.             {
88.                 cout<<"Sorry!!! No Node found whith Data Value = "<<dataLocation<<"."<<endl;
89.             }
90.     }
91.     else
92.     {
93.         cout<<"Sorry!!! No data inserted-No value found-List is empty"<<endl;
94.     }
95. }
96.
97. void List::midInsertWhereAddress(int value, Node *addressLocation) //insert to same location where address == Y

98. {
99.     if(!isEmpty())
```

```
100.          {
101.              Node *tempPtr;
102.              tempPtr = head;
103.              int check=0;
104.
105.              while(tempPtr->next != addressLocation && tempPtr->next != '\0')
106.              {
107.                  tempPtr = tempPtr->next;
108.              }
109.                  if(tempPtr->next == addressLocation) //if address is found in node in list
110.                  {
111.                      addressLocation->data = value;
112.                  }
113.                  else
114.                  {
115.                      cout<<"Sorry!!! No Node found with Address = "<<addressLocation<<"."<<endl;
116.                  }
117.          }
118.          else
119.          {
120.              cout<<"Sorry!!! No data inserted-No address found-List is Empty"<<endl;
121.          }
122.      }
123.
124.      void List::midInsertNodeToNextWhereAddress(int value, Node *addressLocation) //insert Node next to locatio
    n where address == Y
125.      {
126.          if(!isEmpty())
127.          {
128.              Node *tempPtr;
129.              tempPtr = head;
130.
131.              while(tempPtr != addressLocation && tempPtr->next != '\0')
132.              {
133.                  tempPtr = tempPtr->next;
134.              }
135.                  if(tempPtr == addressLocation ) //if address is found in node in list
136.                  {
137.                      Node *ptr;
138.                      ptr = new Node;
139.                      ptr->data = 0;
140.                      ptr->next = '\0';
141.
142.                      ptr->data = value;
143.                      ptr->next = addressLocation->next;
144.                      addressLocation->next = ptr;
145.                  }
146.                  else
147.                  {
148.                      cout<<"Sorry!!! No Node found whith Address = "<<addressLocation<<"."<<endl;
149.                  }
150.          }
151.          else
152.          {
153.              cout<<"Sorry!!! No data inserted-No address found-List is empty"<<endl;
154.          }
155.      }
156.
157.      void List::midInsertWhereCount(int value, int countLocation) //count Nodes and insert data to count==Z
158.      {
159.          if(!isEmpty())
160.          {
161.              Node *tempPtr;
162.              tempPtr = head;
163.              int count = 0;
164.
165.              while(count != countLocation && tempPtr->next != '\0')
166.              {
167.                  tempPtr = tempPtr->next;
168.                  count++;
```

```
                }
                    if(count == countLocation) //if count position is found in list
                    {
                        tempPtr->data = value;
                    }
                    else
                    {
                        cout<<"Sorry!!! No Node found at Count = "<<countLocation<<"."<<endl;
                    }
            }
            else
            {
                cout<<"Sorry!!! No data inserted-No count ound-List is empty"<<endl;
            }

        }

        void List::midInsertNodeToNextWhereCount(int value, int countLocation) //count Nodes and insert Node next to count==Z)
        {
            if(!isEmpty())
            {
                Node *tempPtr;
                tempPtr = head;
                int count = 0;

                while(count != countLocation && tempPtr->next != '\0')
                {
                    tempPtr = tempPtr->next;
                    count++;
                }
                    if(count == countLocation) //if count position is found in list
                    {
                        Node *ptr;
                        ptr = new Node;
                        ptr->data = 0;
                        ptr->next = '\0';

                        ptr->data = value;
                        ptr->next = tempPtr->next;
                        tempPtr->next = ptr;
                    }
                    else
                    {
                        cout<<"Sorry!!! No Node found at Count = "<<countLocation<<"."<<endl;
                    }
            }
            else
            {
                cout<<"Sorry!!! No data inserted-No count found-List is empty"<<endl;
            }
        }

        void List::endInsert(int value)
        {
            Node *tempPtr;
            tempPtr = head;

            while(tempPtr->next != '\0')
            {
                tempPtr = tempPtr->next;
            }

            Node *ptr;
            ptr = new Node;
            ptr->data = 0;
            ptr->next = '\0';

            ptr->data = value;

```

```
238.            ptr->next = tempPtr->next;
239.            tempPtr->next = ptr;
240.        }
241.
242.        void List::findNode(int value)
243.        {
244.            if(!isEmpty())
245.            {
246.                Node *tempPtr;
247.                tempPtr = head;
248.                int count = 0;
249.
250.                while(tempPtr->data != value && tempPtr->next != '\0')
251.                {
252.                    tempPtr = tempPtr->next;
253.                    count++;
254.                }
255.                    if(tempPtr->data == value) //if data is found in node in list
256.                    {
257.                        cout<<"NODE FIND RESULT:"<<endl;
258.                        cout<<"================="<<endl;
259.                        cout<<"Node Address: "<<tempPtr<<endl;
260.                        cout<<"Node Data Value: "<<tempPtr->data<<endl;
261.                        cout<<"Node Count Postion: "<<count<<endl;
262.                    }
263.                    else
264.                    {
265.                        cout<<"Sorry!!! No Node found with Data Value = "<<value<<"."<<endl;
266.                    }
267.            }
268.            else
269.            {
270.                cout<<"Sorry!!! List is empty."<<endl;
271.            }
272.        }
273.
274.        void List::beginDelete()
275.        {
276.            if(!isEmpty())
277.            {
278.                head = head->next;
279.            }
280.            else
281.            {
282.                cout<<"Sorry!!! List is empty."<<endl;
283.            }
284.        }
285.
286.        void List::midDelete(int dataLocation)
287.        {
288.            if(!isEmpty())
289.            {
290.                Node *predPtr;
291.                Node *tempPtr;
292.                predPtr = head;
293.                tempPtr = head;
294.
295.                while(tempPtr->data != dataLocation && tempPtr->next != '\0')
296.                {
297.                    predPtr = tempPtr;
298.                    tempPtr = tempPtr->next;
299.                }
300.                    if(tempPtr->data == dataLocation) //if data is found in node in list
301.                    {
302.                        predPtr->next = tempPtr->next;
303.                    }
304.                    else
305.                    {
306.                        cout<<"Sorry!!! No Node found with Data Value = "<<dataLocation<<"."<<endl;
307.                    }
```

```
308.            }
309.            else
310.            {
311.                cout<<"Sorry!!! List is empty."<<endl;
312.            }
313.        }
314.
315.        void List::endDelete()
316.        {
317.            if(!isEmpty())
318.            {
319.
320.                Node *tempPtr;
321.                Node *predPtr;
322.                tempPtr = head;
323.                predPtr = head;
324.
325.                while(tempPtr->next != '\0')
326.                {
327.                    predPtr = tempPtr;
328.                    tempPtr = tempPtr->next;
329.                }
330.
331.                predPtr->next = '\0';
332.            }
333.            else
334.            {
335.                cout<<"Sorry!!! List is Empty."<<endl;
336.            }
337.
338.        }
339.
340.        void List::displayList()
341.        {
342.            if(!isEmpty())
343.            {
344.                Node *tempPtr;
345.                tempPtr = head;
346.
347.                while(tempPtr->next != '\0')
348.                {
349.                    tempPtr = tempPtr->next;
350.                    cout<<tempPtr->data<<"  ";
351.                }
352.                cout<<endl;
353.            }
354.            else
355.            {
356.                cout<<"Sorry!!! List is Empty."<<endl;
357.            }
358.        }
```

main.cpp File:

```
1.  #include "List.h"
2.  #include "conio.h"
3.  #include "Node.h"
4.  #include <iostream>
5.  using namespace std;
6.
7.  void main()
8.  {
9.      List l;
10.
11.     cout<<"LINKED LIST EMPTY CEHCK RESULT:"<<endl;
12.     cout<<"==============================="<<endl;
13.     if(l.isEmpty())
14.     {
15.         cout<<"Yes. List is Empty."<<endl;
```

```
16.     }
17.     else
18.     {
19.         cout<<"No. List is not Empty"<<endl;
20.     }
21.     cout<<endl;
22.
23.     cout<<"DISPLAY AFTER BEGIN NODE INSERTION:"<<endl;
24.     cout<<"==================================="<<endl;
25.     l.beginInsert(1);
26.     l.beginInsert(2);
27.     l.displayList();
28.     cout<<endl;
29.
30.     cout<<"DISPLAY AFTER END NODE INSERTION:"<<endl;
31.     cout<<"================================="<<endl;
32.     l.endInsert(3);
33.     l.endInsert(4);
34.     l.displayList();
35.     cout<<endl;
36.
37.     cout<<"DISPLAY AFTER-MID DATA VALUE-
    DATA VALUE INSERTION:"<<endl; //insert Data Value to same location where data == X
38.     cout<<"================================================="<<endl;
39.     l.midInsertWhereData(99, 3);
40.     l.displayList();
41.     cout<<endl;
42.
43.     cout<<"DISPLAY AFTER-MID DATA VALUE-NODE INSERTION:"<<endl; //insert Node next to location where data == X
44.     cout<<"============================================="<<endl;
45.     l.midInsertNodeToNextWhereData(90, 4);
46.     l.displayList();
47.     cout<<endl;
48.
49.     cout<<"DISPLAY AFTER-MID ADDRESS-
    DATA VALUE INSERTION:"<<endl; //inser Data Value to same location where address == Y
50.     cout<<"================================================"<<endl;
51.     l.midInsertWhereAddress(10, l.head->next->next);
52.     l.displayList();
53.     cout<<endl;
54.
55.     cout<<"DISPLAY AFTER-MID ADDRESS-NODE INSERTION:"<<endl; //insert node next to location where address == Y
56.     cout<<"========================================="<<endl;
57.     l.midInsertNodeToNextWhereAddress(20, l.head->next->next );
58.     l.displayList();
59.     cout<<endl;
60.
61.     cout<<"DISPLAY AFTER-COUNTING NODES-
    DATA VALUE INSERTION:"<<endl;  //count nodes and insert Data Value to count==Z
62.     cout<<"================================================="<<endl;
63.     l.midInsertWhereCount(6, 3);
64.     l.displayList();
65.     cout<<endl;
66.
67.     cout<<"DISPLAY AFTER-COUNTING NODES-
    NODE INSERTION:"<<endl; //count nodes and insert Node next to count == Z
68.     cout<<"============================================="<<endl;
69.     l.midInsertNodeToNextWhereCount(90, 3);
70.     l.displayList();
71.     cout<<endl;
72.
73.     l.findNode(4);
74.     cout<<endl;
75.
76.     cout<<"DISPLAY AFTER-BEGIN NODE-DELETION:"<<endl;
77.     cout<<"==================================="<<endl;
78.     l.beginDelete();
79.     l.displayList();
80.     cout<<endl;
81.
```

```
82.     cout<<"DISPLAY AFTER-MID NODE-DELETION:"<<endl; //delete node with Data Value == Z
83.     cout<<"================================"<<endl;
84.     l.midDelete(6);
85.     l.displayList();
86.     cout<<endl;
87.
88.     cout<<"DISPLAY AFTER-END NODE-DELETION:"<<endl;
89.     cout<<"================================"<<endl;
90.     l.endDelete();
91.     l.displayList();
92.     cout<<endl;
93.
94.     getch();
95. }
```

Output:

Exercise 2:

Linked lists allow efficient implementation of a number of data structures. For instance, Queues can be implemented using a linked list to store the data values. The first node can serve as the 'front' while the last node can be regarded as 'rear' of the queue (Figure 1). The Enqueue() operation is equivalent to adding a node at the end of the list while the Dequeue() operation removes the first element from the list. Implement the 'Queue' class using a linked list for data storage.

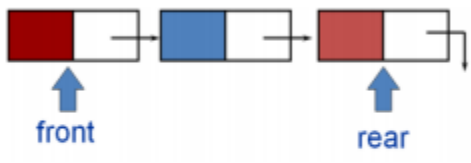


Figure 1 Linked list based implementation of queue

Solution:

queue.h File:

```
1.  #include "List.h"
2.  #include "conio.h"
3.  #include "Node.h"
4.  #include "conio.h"
5.  #include <iostream>
6.  using namespace std;
7.  #pragma once
8.  class _Queue
9.  {
10. private:
11.     List l;
12. public:
13.     _Queue(void);
14.     bool isEmpty();
15.     int getFront();
16.     void enqueue(int);
17.     int dequeue();
18.     void displayQueue();
19.
20. };
```

queue.cpp File:

```
1.  #include "_Queue.h"
2.  #include "List.h"
3.  #include "conio.h"
4.  #include "Node.h"
5.  #include <iostream>
6.  using namespace std;
7.
8.  _Queue::_Queue(void)
9.  {
10. }
11.
12. void _Queue::enqueue(int value)
13. {
14.     l.endInsert(value);
15. }
16.
17. int _Queue::dequeue()
18. {
```

```
    return ( l.beginDelete() );
}

void _Queue::displayQueue()
{
    l.displayList();
}

bool _Queue::isEmpty()
{
    return ( l.isEmpty() );
}

int _Queue::getFront()
{
    int value;
    value = l.head->next->data;
    return value;
}
```

node.h File:

```
#pragma once
class Node
{
public:
    int data;
    Node *next;
public:
    Node(void);
};
```

node.cpp File:

```
#include "Node.h"


Node::Node(void)
{
}
```

list.h File:

```
#include "Node.h"
#pragma once
class List
{
public:
    Node *head;
public:
    List(void);
    bool isEmpty();

    void beginInsert(int);
    void midInsertWhereData(int, int); //insert to same location where data == X
    void midInsertNodeToNextWhereData(int, int); //insert Node next to location where data == X
    void midInsertWhereAddress(int, Node *); //insert to same location where address == Y
    void midInsertNodeToNextWhereAddress(int, Node *); //insert Node next to location where address == Y
    void midInsertWhereCount(int, int); //count nodes and inset to count==Z
    void midInsertNodeToNextWhereCount(int, int); //count nodes and insert Node next to count == Z
    void endInsert(int);

    void findNode(int);

    int beginDelete();
    void midDelete(int); //delete node with Data Value == Z
```

```
    void endDelete();

    void displayList();
};
```

list.cpp File:

```
#include "List.h"
#include "conio.h"
#include "Node.h"
#include <iostream>
using namespace std;

List::List(void)
{
    head = new Node;
    head->next = '\0';
}

bool List::isEmpty()
{
    if(head->next == '\0')
    {
        return true;
    }
    else
    {
        return false;
    }
}

void List::beginInsert(int value) //inserts node at beginning
{
    Node *ptr;
    ptr = new Node;
    ptr->data = 0;
    ptr->next = '\0';

    ptr->data = value;

    ptr->next = head->next;
    head->next = ptr;
}

void List::midInsertWhereData(int value, int dataLocation) //insert to same location where data == X
{
    if(!isEmpty())
    {
        Node *tempPtr;
        tempPtr = head->next;

        while(tempPtr->data != dataLocation && tempPtr->next != '\0')
        {
            tempPtr = tempPtr->next;
        }
            if(tempPtr->data == dataLocation) //if data is found in node in list
            {
                tempPtr->data = value;
            }
            else
            {
                cout<<"Sorry!!! No Node found with Data Value = "<<dataLocation<<"."<<endl;
            }
    }
    else
    {
        cout<<"Sorry!!! No data inserted-No value found-List is empty"<<endl;
    }
}
```

```
63.
64. void List::midInsertNodeToNextWhereData(int value, int dataLocation) //insert Node next to location where data ==
     X
65. {
66.     if(!isEmpty())
67.     {
68.         Node *tempPtr;
69.         tempPtr = head;
70.
71.         while(tempPtr->data != dataLocation && tempPtr->next != '\0')
72.         {
73.             tempPtr = tempPtr->next;
74.         }
75.             if(tempPtr->data == dataLocation) //if data is found in node in list
76.             {
77.                 Node *ptr;
78.                 ptr = new Node;
79.                 ptr->data = 0;
80.                 ptr->next = '\0';
81.
82.                 ptr->data = value;
83.                 ptr->next = tempPtr->next;
84.                 tempPtr->next = ptr;
85.             }
86.             else
87.             {
88.                 cout<<"Sorry!!! No Node found whith Data Value = "<<dataLocation<<"."<<endl;
89.             }
90.     }
91.     else
92.     {
93.         cout<<"Sorry!!! No data inserted-No value found-List is empty"<<endl;
94.     }
95. }
96.
97. void List::midInsertWhereAddress(int value, Node *addressLocation) //insert to same location where address == Y

98. {
99.     if(!isEmpty())
100.            {
101.                Node *tempPtr;
102.                tempPtr = head;
103.                int check=0;
104.
105.                while(tempPtr->next != addressLocation && tempPtr->next != '\0')
106.                {
107.                    tempPtr = tempPtr->next;
108.                }
109.                    if(tempPtr->next == addressLocation) //if address is found in node in list
110.                    {
111.                        addressLocation->data = value;
112.                    }
113.                    else
114.                    {
115.                        cout<<"Sorry!!! No Node found with Address = "<<addressLocation<<"."<<endl;
116.                    }
117.            }
118.            else
119.            {
120.                cout<<"Sorry!!! No data inserted-No address found-List is Empty"<<endl;
121.            }
122.        }
123.
124.        void List::midInsertNodeToNextWhereAddress(int value, Node *addressLocation) //insert Node next to locatio
    n where address == Y
125.        {
126.            if(!isEmpty())
127.            {
128.                Node *tempPtr;
129.                tempPtr = head;
```

```
130.
131.                while(tempPtr != addressLocation && tempPtr->next != '\0')
132.                {
133.                    tempPtr = tempPtr->next;
134.                }
135.                    if(tempPtr == addressLocation ) //if address is found in node in list
136.                    {
137.                        Node *ptr;
138.                        ptr = new Node;
139.                        ptr->data = 0;
140.                        ptr->next = '\0';
141.
142.                        ptr->data = value;
143.                        ptr->next = addressLocation->next;
144.                        addressLocation->next = ptr;
145.                    }
146.                    else
147.                    {
148.                        cout<<"Sorry!!! No Node found whith Address = "<<addressLocation<<"."<<endl;
149.                    }
150.            }
151.            else
152.            {
153.                cout<<"Sorry!!! No data inserted-No address found-List is empty"<<endl;
154.            }
155.        }
156.
157.        void List::midInsertWhereCount(int value, int countLocation) //count Nodes and insert data to count==Z
158.        {
159.            if(!isEmpty())
160.            {
161.                Node *tempPtr;
162.                tempPtr = head;
163.                int count = 0;
164.
165.                while(count != countLocation && tempPtr->next != '\0')
166.                {
167.                    tempPtr = tempPtr->next;
168.                    count++;
169.                }
170.                    if(count == countLocation) //if count position is found in list
171.                    {
172.                        tempPtr->data = value;
173.                    }
174.                    else
175.                    {
176.                        cout<<"Sorry!!! No Node found at Count = "<<countLocation<<"."<<endl;
177.                    }
178.            }
179.            else
180.            {
181.                cout<<"Sorry!!! No data inserted-No count ound-List is empty"<<endl;
182.            }
183.
184.        }
185.
186.        void List::midInsertNodeToNextWhereCount(int value, int countLocation) //count Nodes and insert Node next
    to count==Z)
187.        {
188.            if(!isEmpty())
189.            {
190.                Node *tempPtr;
191.                tempPtr = head;
192.                int count = 0;
193.
194.                while(count != countLocation && tempPtr->next != '\0')
195.                {
196.                    tempPtr = tempPtr->next;
197.                    count++;
198.                }
```

```
199.                    if(count == countLocation) //if count position is found in list
200.                    {
201.                        Node *ptr;
202.                        ptr = new Node;
203.                        ptr->data = 0;
204.                        ptr->next = '\0';
205.
206.                        ptr->data = value;
207.                        ptr->next = tempPtr->next;
208.                        tempPtr->next = ptr;
209.                    }
210.                    else
211.                    {
212.                        cout<<"Sorry!!! No Node found at Count = "<<countLocation<<"."<<endl;
213.                    }
214.            }
215.            else
216.            {
217.                cout<<"Sorry!!! No data inserted-No count found-List is empty"<<endl;
218.            }
219.        }
220.
221.        void List::endInsert(int value)
222.        {
223.            Node *tempPtr;
224.            tempPtr = head;
225.
226.            while(tempPtr->next != '\0')
227.            {
228.                tempPtr = tempPtr->next;
229.            }
230.
231.            Node *ptr;
232.            ptr = new Node;
233.            ptr->data = 0;
234.            ptr->next = '\0';
235.
236.            ptr->data = value;
237.
238.            ptr->next = tempPtr->next;
239.            tempPtr->next = ptr;
240.        }
241.
242.        void List::findNode(int value)
243.        {
244.            if(!isEmpty())
245.            {
246.                Node *tempPtr;
247.                tempPtr = head;
248.                int count = 0;
249.
250.                while(tempPtr->data != value && tempPtr->next != '\0')
251.                {
252.                    tempPtr = tempPtr->next;
253.                    count++;
254.                }
255.                    if(tempPtr->data == value) //if data is found in node in list
256.                    {
257.                        cout<<"NODE FIND RESULT:"<<endl;
258.                        cout<<"================="<<endl;
259.                        cout<<"Node Address: "<<tempPtr<<endl;
260.                        cout<<"Node Data Value: "<<tempPtr->data<<endl;
261.                        cout<<"Node Count Postion: "<<count<<endl;
262.                    }
263.                    else
264.                    {
265.                        cout<<"Sorry!!! No Node found with Data Value = "<<value<<"."<<endl;
266.                    }
267.            }
268.            else
```

```
269.            {
270.                cout<<"Sorry!!! List is empty."<<endl;
271.            }
272.        }
273.
274.        int List::beginDelete()
275.        {
276.            if(!isEmpty())
277.            {
278.                int temp;
279.                head = head->next;
280.                temp = head->data;
281.                return temp;
282.            }
283.            else
284.            {
285.                cout<<"Sorry!!! List is empty."<<endl;
286.                return (-1);
287.            }
288.        }
289.
290.        void List::midDelete(int dataLocation)
291.        {
292.            if(!isEmpty())
293.            {
294.                Node *predPtr;
295.                Node *tempPtr;
296.                predPtr = head;
297.                tempPtr = head;
298.
299.                while(tempPtr->data != dataLocation && tempPtr->next != '\0')
300.                {
301.                    predPtr = tempPtr;
302.                    tempPtr = tempPtr->next;
303.                }
304.                    if(tempPtr->data == dataLocation) //if data is found in node in list
305.                    {
306.                        predPtr->next = tempPtr->next;
307.                    }
308.                    else
309.                    {
310.                        cout<<"Sorry!!! No Node found with Data Value = "<<dataLocation<<"."<<endl;
311.                    }
312.            }
313.            else
314.            {
315.                cout<<"Sorry!!! List is empty."<<endl;
316.            }
317.        }
318.
319.        void List::endDelete()
320.        {
321.            if(!isEmpty())
322.            {
323.
324.                Node *tempPtr;
325.                Node *predPtr;
326.                tempPtr = head;
327.                predPtr = head;
328.
329.                while(tempPtr->next != '\0')
330.                {
331.                    predPtr = tempPtr;
332.                    tempPtr = tempPtr->next;
333.                }
334.
335.                predPtr->next = '\0';
336.            }
337.            else
338.            {
```

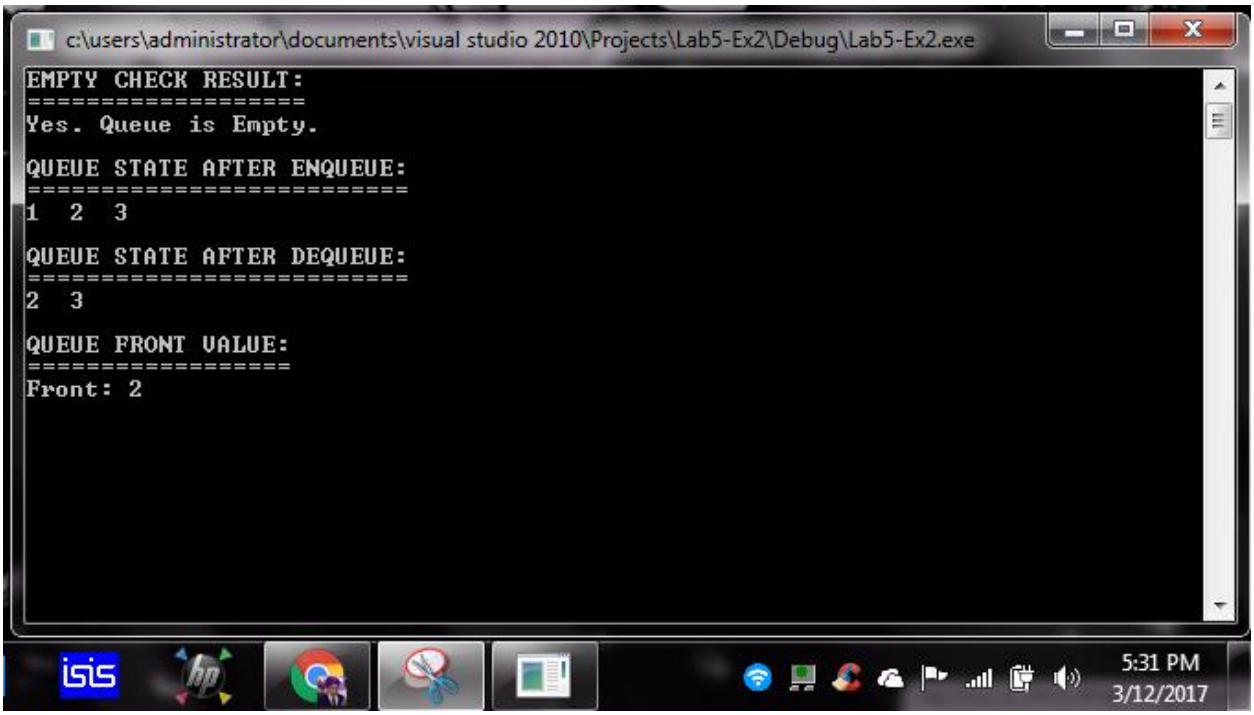
```
339.                cout<<"Sorry!!! List is Empty."<<endl;
340.            }
341.
342.        }
343.
344.        void List::displayList()
345.        {
346.            if(!isEmpty())
347.            {
348.                Node *tempPtr;
349.                tempPtr = head;
350.
351.                while(tempPtr->next != '\0')
352.                {
353.                    tempPtr = tempPtr->next;
354.                    cout<<tempPtr->data<<"  ";
355.                }
356.                cout<<endl;
357.            }
358.            else
359.            {
360.                cout<<"Sorry!!! List is Empty."<<endl;
361.            }
362.        }
```

main.cpp File:

```
1.  #include "_Queue.h"
2.  #include "List.h"
3.  #include "conio.h"
4.  #include "Node.h"
5.  #include "conio.h"
6.  #include <iostream>
7.  using namespace std;
8.
9.  void main()
10. {
11.     _Queue q;
12.
13.     cout<<"EMPTY CHECK RESULT:"<<endl;
14.     cout<<"==================="<<endl;
15.     if(q.isEmpty())
16.     {
17.         cout<<"Yes. Queue is Empty."<<endl;
18.     }
19.     else
20.     {
21.         cout<<"No. Queue is not Empty."<<endl;
22.     }
23.     cout<<endl;
24.
25.     cout<<"QUEUE STATE AFTER ENQUEUE:"<<endl;
26.     cout<<"=========================="<<endl;
27.     q.enqueue(1);
28.     q.enqueue(2);
29.     q.enqueue(3);
30.     q.displayQueue();
31.     cout<<endl;
32.
33.     cout<<"QUEUE STATE AFTER DEQUEUE:"<<endl;
34.     cout<<"=========================="<<endl;
35.     q.dequeue();
36.     q.displayQueue();
37.     cout<<endl;
38.
39.     cout<<"QUEUE FRONT VALUE:"<<endl;
40.     cout<<"=================="<<endl;
41.     cout<<"Front: "<<q.getFront();
```

```
42.     cout<<endl;
43.
44.     getch();
45. }
```

Output:



Exercise 3:

Write the following C++ functions to realize the indicated functionality on a singly linked list of integers.

- A function which accepts a pointer to the first node and returns the maximum value in the list.
- A function that counts the total number of nodes in the list
- A function to search a given value in the list and return the node number where the queried value is found
- A function to display the elements of the list using recursion.
- A function swap(Node *p1, Node *p2) that swaps the data in the nodes p1 and p2

Solution:

node.h File:

```
1.  #pragma once
2.  class Node
3.  {
4.  public:
5.      int data;
6.      Node *next;
7.  public:
8.      Node(void);
9.  };
```

node.cpp File:

```
1.  #include "Node.h"
2.
```

```
3.
4.  Node::Node(void)
5.  {
6.  }
```

list.h File:

```
1.  #include "Node.h"
2.  #pragma once
3.  class List
4.  {
5.  public:
6.      Node *head;
7.  public:
8.      List(void);
9.      bool isEmpty();
10.
11.     void beginInsert(int);
12.     void midInsertWhereData(int, int); //insert to same location where data == X
13.     void midInsertNodeToNextWhereData(int, int); //insert Node next to location where data == X
14.     void midInsertWhereAddress(int, Node *); //insert to same location where address == Y
15.     void midInsertNodeToNextWhereAddress(int, Node *); //insert Node next to location where address == Y
16.     void midInsertWhereCount(int, int); //count nodes and inset to count==Z
17.     void midInsertNodeToNextWhereCount(int, int); //count nodes and insert Node next to count == Z
18.     void endInsert(int);
19.
20.     void findNode(int);
21.
22.     void beginDelete();
23.     void midDelete(int); //delete node with Data Value == Z
24.     void endDelete();
25.
26.     void displayList();
27. };
```

list.cpp File:

```
1.  #include "List.h"
2.  #include "conio.h"
3.  #include "Node.h"
4.  #include <iostream>
5.  using namespace std;
6.
7.  List::List(void)
8.  {
9.      head = new Node;
10.     head->next = '\0';
11. }
12.
13. bool List::isEmpty()
14. {
15.     if(head->next == '\0')
16.     {
17.         return true;
18.     }
19.     else
20.     {
21.         return false;
22.     }
23. }
24.
25. void List::beginInsert(int value) //inserts node at beginning
26. {
27.     Node *ptr;
28.     ptr = new Node;
29.     ptr->data = 0;
30.     ptr->next = '\0';
```

```
31.
32.     ptr->data = value;
33.
34.     ptr->next = head->next;
35.     head->next = ptr;
36. }
37.
38. void List::midInsertWhereData(int value, int dataLocation) //insert to same location where data == X
39. {
40.     if(!isEmpty())
41.     {
42.         Node *tempPtr;
43.         tempPtr = head->next;
44.
45.         while(tempPtr->data != dataLocation && tempPtr->next != '\0')
46.         {
47.             tempPtr = tempPtr->next;
48.         }
49.             if(tempPtr->data == dataLocation) //if data is found in node in list
50.             {
51.                 tempPtr->data = value;
52.             }
53.             else
54.             {
55.                 cout<<"Sorry!!! No Node found with Data Value = "<<dataLocation<<"."<<endl;
56.             }
57.     }
58.     else
59.     {
60.         cout<<"Sorry!!! No data inserted-No value found-List is empty"<<endl;
61.     }
62. }
63.
64. void List::midInsertNodeToNextWhereData(int value, int dataLocation) //insert Node next to location where data ==
    X
65. {
66.     if(!isEmpty())
67.     {
68.         Node *tempPtr;
69.         tempPtr = head;
70.
71.         while(tempPtr->data != dataLocation && tempPtr->next != '\0')
72.         {
73.             tempPtr = tempPtr->next;
74.         }
75.             if(tempPtr->data == dataLocation) //if data is found in node in list
76.             {
77.                 Node *ptr;
78.                 ptr = new Node;
79.                 ptr->data = 0;
80.                 ptr->next = '\0';
81.
82.                 ptr->data = value;
83.                 ptr->next = tempPtr->next;
84.                 tempPtr->next = ptr;
85.             }
86.             else
87.             {
88.                 cout<<"Sorry!!! No Node found whith Data Value = "<<dataLocation<<"."<<endl;
89.             }
90.     }
91.     else
92.     {
93.         cout<<"Sorry!!! No data inserted-No value found-List is empty"<<endl;
94.     }
95. }
96.
97. void List::midInsertWhereAddress(int value, Node *addressLocation) //insert to same location where address == Y

98. {
```

```
    if(!isEmpty())
            {
                Node *tempPtr;
                tempPtr = head;
                int check=0;

                while(tempPtr->next != addressLocation && tempPtr->next != '\0')
                {
                    tempPtr = tempPtr->next;
                }
                    if(tempPtr->next == addressLocation) //if address is found in node in list
                    {
                        addressLocation->data = value;
                    }
                    else
                    {
                        cout<<"Sorry!!! No Node found with Address = "<<addressLocation<<"."<<endl;
                    }
            }
            else
            {
                cout<<"Sorry!!! No data inserted-No address found-List is Empty"<<endl;
            }
        }

        void List::midInsertNodeToNextWhereAddress(int value, Node *addressLocation) //insert Node next to locatio
    n where address == Y
        {
            if(!isEmpty())
            {
                Node *tempPtr;
                tempPtr = head;

                while(tempPtr != addressLocation && tempPtr->next != '\0')
                {
                    tempPtr = tempPtr->next;
                }
                    if(tempPtr == addressLocation ) //if address is found in node in list
                    {
                        Node *ptr;
                        ptr = new Node;
                        ptr->data = 0;
                        ptr->next = '\0';

                        ptr->data = value;
                        ptr->next = addressLocation->next;
                        addressLocation->next = ptr;
                    }
                    else
                    {
                        cout<<"Sorry!!! No Node found whith Address = "<<addressLocation<<"."<<endl;
                    }
            }
            else
            {
                cout<<"Sorry!!! No data inserted-No address found-List is empty"<<endl;
            }
        }

        void List::midInsertWhereCount(int value, int countLocation) //count Nodes and insert data to count==Z
        {
            if(!isEmpty())
            {
                Node *tempPtr;
                tempPtr = head;
                int count = 0;

                while(count != countLocation && tempPtr->next != '\0')
                {
                    tempPtr = tempPtr->next;
```

```
                    count++;
                }
                    if(count == countLocation) //if count position is found in list
                    {
                        tempPtr->data = value;
                    }
                    else
                    {
                        cout<<"Sorry!!! No Node found at Count = "<<countLocation<<"."<<endl;
                    }
            }
            else
            {
                cout<<"Sorry!!! No data inserted-No count ound-List is empty"<<endl;
            }

        }

        void List::midInsertNodeToNextWhereCount(int value, int countLocation) //count Nodes and insert Node next to count==Z)
        {
            if(!isEmpty())
            {
                Node *tempPtr;
                tempPtr = head;
                int count = 0;

                while(count != countLocation && tempPtr->next != '\0')
                {
                    tempPtr = tempPtr->next;
                    count++;
                }
                    if(count == countLocation) //if count position is found in list
                    {
                        Node *ptr;
                        ptr = new Node;
                        ptr->data = 0;
                        ptr->next = '\0';

                        ptr->data = value;
                        ptr->next = tempPtr->next;
                        tempPtr->next = ptr;
                    }
                    else
                    {
                        cout<<"Sorry!!! No Node found at Count = "<<countLocation<<"."<<endl;
                    }
            }
            else
            {
                cout<<"Sorry!!! No data inserted-No count found-List is empty"<<endl;
            }
        }

        void List::endInsert(int value)
        {
            Node *tempPtr;
            tempPtr = head;

            while(tempPtr->next != '\0')
            {
                tempPtr = tempPtr->next;
            }

            Node *ptr;
            ptr = new Node;
            ptr->data = 0;
            ptr->next = '\0';

            ptr->data = value;
```

```
237.
238.            ptr->next = tempPtr->next;
239.            tempPtr->next = ptr;
240.        }
241.
242.        void List::findNode(int value)
243.        {
244.            if(!isEmpty())
245.            {
246.                Node *tempPtr;
247.                tempPtr = head;
248.                int count = 0;
249.
250.                while(tempPtr->data != value && tempPtr->next != '\0')
251.                {
252.                    tempPtr = tempPtr->next;
253.                    count++;
254.                }
255.                    if(tempPtr->data == value) //if data is found in node in list
256.                    {
257.                        cout<<"NODE FIND RESULT:"<<endl;
258.                        cout<<"================="<<endl;
259.                        cout<<"Node Address: "<<tempPtr<<endl;
260.                        cout<<"Node Data Value: "<<tempPtr->data<<endl;
261.                        cout<<"Node Count Postion: "<<count<<endl;
262.                    }
263.                    else
264.                    {
265.                        cout<<"Sorry!!! No Node found with Data Value = "<<value<<"."<<endl;
266.                    }
267.            }
268.            else
269.            {
270.                cout<<"Sorry!!! List is empty."<<endl;
271.            }
272.        }
273.
274.        void List::beginDelete()
275.        {
276.            if(!isEmpty())
277.            {
278.                head = head->next;
279.            }
280.            else
281.            {
282.                cout<<"Sorry!!! List is empty."<<endl;
283.            }
284.        }
285.
286.        void List::midDelete(int dataLocation)
287.        {
288.            if(!isEmpty())
289.            {
290.                Node *predPtr;
291.                Node *tempPtr;
292.                predPtr = head;
293.                tempPtr = head;
294.
295.                while(tempPtr->data != dataLocation && tempPtr->next != '\0')
296.                {
297.                    predPtr = tempPtr;
298.                    tempPtr = tempPtr->next;
299.                }
300.                    if(tempPtr->data == dataLocation) //if data is found in node in list
301.                    {
302.                        predPtr->next = tempPtr->next;
303.                    }
304.                    else
305.                    {
306.                        cout<<"Sorry!!! No Node found with Data Value = "<<dataLocation<<"."<<endl;
```

```
                    }
            }
            else
            {
                cout<<"Sorry!!! List is empty."<<endl;
            }
        }

        void List::endDelete()
        {
            if(!isEmpty())
            {

                Node *tempPtr;
                Node *predPtr;
                tempPtr = head;
                predPtr = head;

                while(tempPtr->next != '\0')
                {
                    predPtr = tempPtr;
                    tempPtr = tempPtr->next;
                }

                predPtr->next = '\0';
            }
            else
            {
                cout<<"Sorry!!! List is Empty."<<endl;
            }

        }

        void List::displayList()
        {
            if(!isEmpty())
            {
                Node *tempPtr;
                tempPtr = head;

                while(tempPtr->next != '\0')
                {
                    tempPtr = tempPtr->next;
                    cout<<tempPtr->data<<"  ";
                }
                cout<<endl;
            }
            else
            {
                cout<<"Sorry!!! List is Empty."<<endl;
            }
        }
```

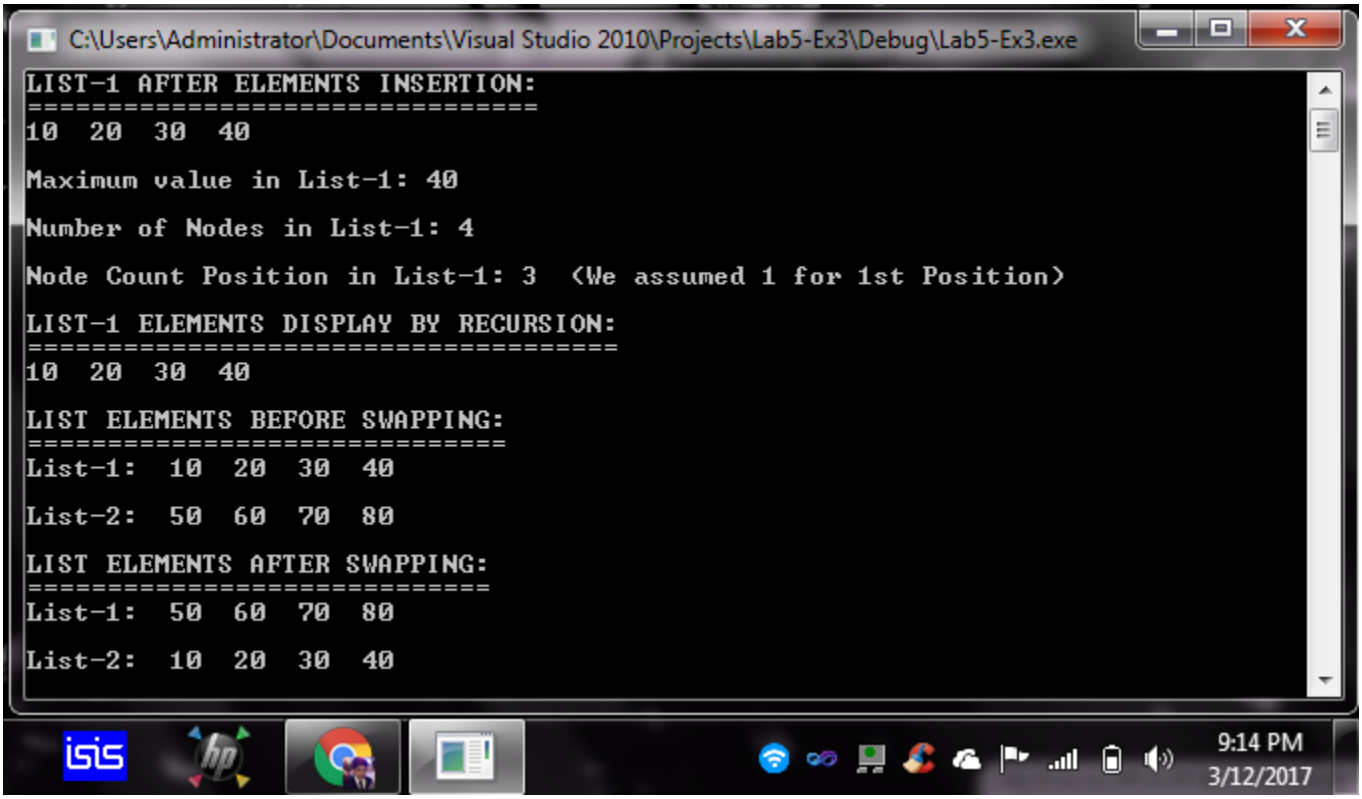
main.cpp File:

```
#include "List.h"
#include "conio.h"
#include "Node.h"
#include "conio.h"
#include <iostream>
using namespace std;

int maximumListValue(Node *n)
{
        int max = n->data;

        while(n->next != '\0')
        {
            n = n->next;
```

```
            if(n->data > max)
            {
                max = n->data;
            }

        }

        return max;
}

int countNodes(Node *n)
{
    if(n != '\0')
    {
        int count = 1; //we took count=1 because function always take first node pointer this mean it must have first node
        while(n->next != '\0')
        {
            n = n->next;
            count++;
        }
        return count;
    }
}

int findNodeCount(Node *n, int dataLocation)
{
        int count = 1; //we took count=1 because function always take first node pointer this mean it must have first node
        while(n->next != '\0' && n->data != dataLocation)
            {
                n = n->next;
                count++;
            }
                if(n->data == dataLocation) //if data is found in node in list
                {
                    return count;
                }
                else
                {
                    cout<<"Sorry!!! No Node found with Data Value = "<<dataLocation<<"."<<endl;
                    return (-1);
                }
}

int recursiveDisplay(Node *n)
{
    if(n == '\0')
    {
        return (1);
    }
    else
    {
        cout<<n->data<<"  ";
        recursiveDisplay(n->next);
    }
}

void swapNodes(Node *n1, Node *n2)
{
    int temp1, temp2;
    while( n1 != '\0' && n2 != '\0')
    {
        temp1 = n1->data;
        temp2 = n2->data;
        n1->data = temp2;
        n2->data = temp1;
        n1 = n1->next;
        n2 = n2->next;
    }
```

```
}

void main()
{
    List l, m;

    l.endInsert(10);
    l.endInsert(20);
    l.endInsert(30);
    l.endInsert(40);

    m.endInsert(50);
    m.endInsert(60);
    m.endInsert(70);
    m.endInsert(80);

    cout<<"LIST-1 AFTER ELEMENTS INSERTION:"<<endl;
            cout<<"================================"<<endl;
            l.displayList();

            cout<<endl;
            cout<<"Maximum value in List-1: "<<maximumListValue(l.head->next)<<endl;

            cout<<endl;
            cout<<"Number of Nodes in List-1: "<<countNodes(l.head->next)<<endl;

            cout<<endl;
            cout<<"Node Count Position in List-1: "<<findNodeCount(l.head-
   >next, 30)<<"  (We assumed 1 for 1st Position)"<<endl;

            cout<<endl;
            cout<<"LIST-1 ELEMENTS DISPLAY BY RECURSION:"<<endl;
            cout<<"======================================"<<endl;
            recursiveDisplay(l.head->next);
            cout<<endl;

            cout<<endl;
            cout<<"LIST ELEMENTS BEFORE SWAPPING:"<<endl;
            cout<<"=============================="<<endl;
            cout<<"List-1:  ";
            l.displayList();
            cout<<endl;
            cout<<"List-2:  ";
            m.displayList();
            cout<<endl;

            swapNodes(l.head->next, m.head->next);

            cout<<"LIST ELEMENTS AFTER SWAPPING:"<<endl;
            cout<<"============================="<<endl;
            cout<<"List-1:  ";
            l.displayList();
            cout<<endl;
            cout<<"List-2:  ";
            m.displayList();
            cout<<endl;

            getch();
       }
```

Output:


C:\Users\Administrator\Documents\Visual Studio 2010\Projects\Lab5-Ex3\Debug\Lab5-Ex3.exe
LIST-1 AFTER ELEMENTS INSERTION:
==========
10 20 30 40
Maximum value in List-1: 40
Number of Nodes in List-1: 4
Node Count Position in List-1: 3 (We assumed 1 for 1st Position)
LIST-1 ELEMENTS DISPLAY BY RECURSION:
==========
10 20 30 40
LIST ELEMENTS BEFORE SWAPPING:
==========
List-1: 10 20 30 40
List-2: 50 60 70 80
LIST ELEMENTS AFTER SWAPPING:
==========
List-1: 50 60 70 80
List-2: 10 20 30 40
9:14 PM
3/12/2017