

Bahria University,

Karachi Campus



LAB EXPERIMENT NO. _10_ LIST OF TASKS

TASK NO	OBJECTIVE
Task 1	Write all programs to create a linked list.
Task 2	Write a program to implement dynamic Stacks using linked list.
Task 3	Write a program to implement dynamic queue by using linked list.
Task 4	N/A
Task 5	N/A
Task 6	N/A
Task 7	N/A
Task 8	N/A

Submitted On:

__04/06/2020__
(Date: DD/MM)

Task No. 1: Write all programs to create a linked list.

Coding:

```
Q1.cpp
1  #include <iostream>
2  using namespace std;
3
4  class Node
5  {
6  public:
7      Node();
8      ~Node();
9
10     int data;
11     Node* next;
12
13 };
14
15 Node::Node()
16 {
17     next = NULL;
18 }
19
20 Node::~~Node()
21 {
22 }
23
24 class List
25 {
26 public:
27     List();
28     ~List();
29     //Checks if list is empty
30     bool isEmpty()
31     {
32         if (head == NULL)
33             return true;
34         else
35             return false;
36     }
37
38     //Insert at First index
39     void insertFirst(int value)
40     {
41         if (isEmpty())
42         {
43             head = createNode(value);
44         }
45         else
46         {
47             Node* newNode = createNode(value);
48             newNode->next = head;
49             head = newNode;
50         }
51     }
52 }
```

```

51 //Insert at Last index
52 void insertLast(int value)
53 {
54     if (isEmpty())
55     {
56         head = createNode(value);
57     }
58     else
59     {
60         Node* tempNode = head;
61
62         while (tempNode != NULL)
63         {
64             if (tempNode->next->next == NULL)
65             {
66                 Node* newNode = createNode(value);
67
68                 tempNode->next->next = newNode;
69                 newNode->next = NULL;
70                 break;
71             }
72
73             tempNode = tempNode->next;
74         }
75     }
76 }
77 //Insert at any index
78 void insertAny(int oldValue, int newValue)
79 {
80     if (isEmpty())
81     {
82         head = createNode(newValue);
83     }
84     else
85     {
86         for (Node* tempNode = head; tempNode != NULL; tempNode = tempNode->next)
87         {
88             if (tempNode->data == oldValue)
89             {
90                 Node* newNode = createNode(newValue);
91
92                 newNode->next = tempNode->next;
93                 tempNode->next = newNode;
94             }
95         }
96     }
97 }
98
99 //Display
100 void display()
101 {
102     if (!isEmpty())
103     {
104         Node* tempNode = head;
105
106         while (tempNode != NULL)
107         {
108             cout << tempNode->data << " ";
109             tempNode = tempNode->next;
110         }
111         cout << endl;
112     }
113     else
114     {
115         cout << "List is Empty..." << endl;
116     }
117 }

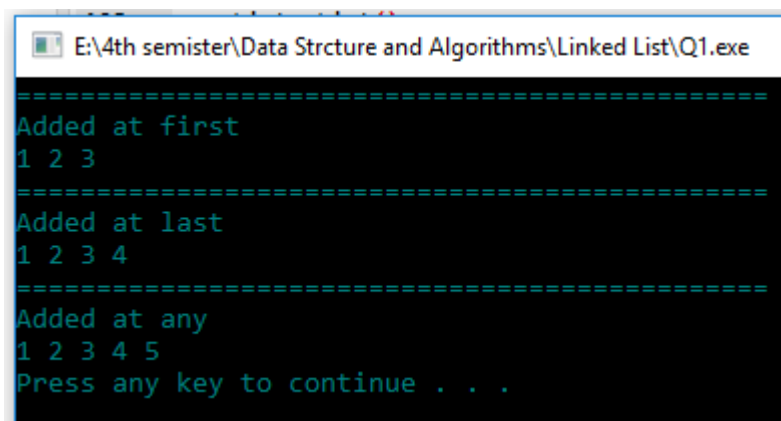
```

```

118     private:
119     Node* head;
120     //Creates a new node
121     Node* createNode(int value)
122     {
123         Node* n = new Node;
124         n->data = value;
125         return n;
126     }
127 };
128 List::List()
129 {
130     head = NULL;
131 }
132
133 List::~~List()
134 {
135     //deletes every node upon completion
136     while (head != NULL)
137     {
138         Node* temp = head;
139         head = head->next;
140
141         delete temp;
142     }
143 }
144 int main()
145 {
146     List l;
147     cout<<"===== "<<endl;
148     cout << "Added at first" << endl;
149     l.insertFirst(3);
150     l.insertFirst(2);
151     l.insertFirst(1);
152     l.display();
153
154     cout<<"===== "<<endl;
155     cout << "Added at last" << endl;
156     l.insertLast(4);
157     l.display();
158
159     cout<<"===== "<<endl;
160     cout << "Added at any" << endl;
161     l.insertAny(4, 5);
162     l.display();
163     system("pause");
164     return 0;
165 }

```

Output:



```

E:\4th semester\Data Structure and Algorithms\Linked List\Q1.exe
=====
Added at first
1 2 3
=====
Added at last
1 2 3 4
=====
Added at any
1 2 3 4 5
Press any key to continue . . .

```

Task No. 2: Write a program to implement dynamic Stacks using linked list.

Coding:

Q2.cpp

```
1  #include <iostream>
2  using namespace std;
3
4  class Node
5  {
6  public:
7      Node();
8      ~Node();
9
10     int data;
11     Node* next;
12
13 };
14
15 Node::Node()
16 {
17     next = NULL;
18 }
19
20 Node::~Node()
21 {
22 }
23
24 class List
25 {
26 public:
27     List();
28     ~List();
29     //Checks if list is empty
30     bool isEmpty()
31     {
32         if (head == NULL)
33             return true;
34         else
35             return false;
36     }
37     //Insert at First index
38     void insertFirst(int value)
39     {
40         if (isEmpty())
41         {
42             head = createNode(value);
43         }
44         else
45         {
46             Node* newNode = createNode(value);
47             newNode->next = head;
48             head = newNode;
49         }
50     }
51     //Insert at Last index
52     void insertLast(int value)
53     {
54         if (isEmpty())
55         {
56             head = createNode(value);
57         }
58         else
59         {
60             Node* tempNode = head;
```

```

60     Node* tempNode = head;
61
62     while (tempNode != NULL)
63     {
64         if (tempNode->next->next == NULL)
65         {
66             Node* newNode = createNode(value);
67
68             tempNode->next->next = newNode;
69             newNode->next = NULL;
70             break;
71         }
72
73         tempNode = tempNode->next;
74     }
75 }
76
77 //Insert at any index
78 void insertAny(int oldValue, int newValue)
79 {
80     if (isEmpty())
81     {
82         head = createNode(newValue);
83     }
84     else
85     {
86         for (Node* tempNode = head; tempNode != NULL; tempNode = tempNode->next)
87         {
88             if (tempNode->data == oldValue)
89             {
90                 Node* newNode = createNode(newValue);
91
92                 newNode->next = tempNode->next;
93                 tempNode->next = newNode;
94             }
95         }
96     }
97 }
98
99 //Display
100 void display()
101 {
102     if (!isEmpty())
103     {
104         Node* tempNode = head;
105
106         while (tempNode != NULL)
107         {
108             cout << tempNode->data << " ";
109             tempNode = tempNode->next;
110         }
111         cout << endl;
112     }
113     else
114     {
115         cout << "List is Empty..." << endl;
116     }
117 }
118
119 //Stack implementation using Linked List
120 //Checks if stack isEmpty
121 bool stackIsEmpty()
122 {
123     if (head == NULL)
124         return true;
125     else
126         return false;

```

```

127 //Pushes value at top of stack
128 void push(int value)
129 {
130     if (stackIsEmpty())
131     {
132         head = createNode(value);
133     }
134     else
135     {
136         Node* newNode = createNode(value);
137
138         newNode->next = head;
139         head = newNode;
140     }
141 }
142 //Pops a value from top of stack
143 void pop()
144 {
145     if (stackIsEmpty())
146     {
147         cout << "Stack is Empty..." << endl;
148     }
149     else
150     {
151         Node* tempNode = head;
152
153         head = head->next;
154
155         delete tempNode;
156     }
157 }
158 //Returns the topmost value
159 int returnTop()
160 {
161     return head->data;
162 }
163
164 private:
165 Node* head;
166 //Creates a new node
167 Node* createNode(int value)
168 {
169     Node* n = new Node;
170     n->data = value;
171     return n;
172 };
173 List::List()
174 {
175     head = NULL;
176 }
177
178 List::~List()
179 {
180     //deletes every node upon completion
181     while (head != NULL)
182     {
183         Node* temp = head;
184         head = head->next;
185
186         delete temp;
187     }
188 }

```

```

188 }
189 int main()
190 {
191     List l;
192     cout<<"===== "<<endl;
193     cout << "Added at first" << endl;
194     l.insertFirst(3);
195     l.insertFirst(2);
196     l.insertFirst(1);
197     l.display();
198     .....
199     cout<<"===== "<<endl;
200     cout << "Added at last" << endl;
201     l.insertLast(4);
202     l.display();
203     .....
204     cout<<"===== "<<endl;
205     cout << "Added at any" << endl;
206     l.insertAny(4, 5);
207     l.display();
208     .....
209     cout<<"===== "<<endl;
210     cout << "After pushing" << endl;
211     l.push(1);
212     l.push(2);
213     l.push(3);
214     l.push(4);
215     cout << "Value at top: " << l.returnTop() << endl;
216     l.display();
217     .....
218     cout<<"===== "<<endl;
219     cout << endl << "After popping" << endl;
220     l.pop();
221     l.display();
222     cout << "Value at top: " << l.returnTop() << endl;
223     system("pause");
224     return 0;
225 }

```

Output:

```

E:\4th semester\Data Strcture and Algorithms\Linked List\Q2.exe
=====
Added at first
1 2 3
=====
Added at last
1 2 3 4
=====
Added at any
1 2 3 4 5
=====
After pushing
Value at top: 4
4 3 2 1 1 2 3 4 5
=====
After popping
3 2 1 1 2 3 4 5
Value at top: 3
Press any key to continue . . .

```


Task No. 3: Write a program to implement dynamic queue by using linked list.

Coding:

q3.cpp

```
1  #include <iostream>
2  using namespace std;
3
4  class Node
5  {
6  public:
7      Node();
8      ~Node();
9
10     int data;
11     Node* next;
12
13 };
14
15 Node::Node()
16 {
17     next = NULL;
18 }
19
20 Node::~~Node()
21 {
22 }
23
24 class List
25 {
26 public:
27     List();
28     ~List();
29     //Checks if list is empty
30     bool isEmpty()
31     {
32         if (head == NULL)
33             return true;
34         else
35             return false;
36     }
37     //Insert at First index
38     void insertFirst(int value)
39     {
40         if (isEmpty())
41         {
42             head = createNode(value);
43         }
44         else
45         {
46             Node* newNode = createNode(value);
47             newNode->next = head;
48             head = newNode;
49         }
50     }
51     //Insert at Last index
52     void insertLast(int value)
53     {
54         if (isEmpty())
55         {
56             head = createNode(value);
57         }
58         else
```

```

59 {
60     Node* tempNode = head;
61
62     while (tempNode != NULL)
63     {
64         if (tempNode->next->next == NULL)
65         {
66             Node* newNode = createNode(value);
67
68             tempNode->next->next = newNode;
69             newNode->next = NULL;
70             break;
71         }
72
73         tempNode = tempNode->next;
74     }
75 }
76
77 //Insert at any index
78 void insertAny(int oldValue, int newValue)
79 {
80     if (isEmpty())
81     {
82         head = createNode(newValue);
83     }
84     else
85     {
86         for (Node* tempNode = head; tempNode != NULL; tempNode = tempNode->next)
87         {
88             if (tempNode->data == oldValue)
89             {
90                 Node* newNode = createNode(newValue);
91
92                 newNode->next = tempNode->next;
93                 tempNode->next = newNode;
94             }
95         }
96     }
97 }
98
99 //Display
100 void display()
101 {
102     if (!isEmpty())
103     {
104         Node* tempNode = head;
105
106         while (tempNode != NULL)
107         {
108             cout << tempNode->data << " ";
109             tempNode = tempNode->next;
110         }
111         cout << endl;
112     }
113     else
114     {
115         cout << "List is Empty..." << endl;
116     }
117 }
118
119 //Single Ended Queue Implementation using Linked List
120 //Checks if queue isEmpty
121 bool queueisEmpty()
122 {
123     if (head == NULL)
124         return true;

```

```

124         return true;
125     else
126         return false;
127     }
128     //Enqueue a node to the queue
129     void Enqueue(int value)
130     {
131         if (queueisEmpty())
132         {
133             head = createNode(value);
134         }
135         else
136         {
137             Node* newNode = createNode(value);
138             for (Node* currentNode = head; currentNode != NULL; currentNode = currentNode->next)
139             {
140                 if (currentNode->next == NULL)
141                 {
142                     currentNode->next = newNode;
143                     break;
144                 }
145             }
146         }
147     }
148     //Dequeue a node
149     void Dequeue()
150     {
151         if (queueisEmpty())
152         {
153             cout << "Queue is Empty" << endl;
154         }
155         else
156         {
157             Node* tempNode = head;
158             head = head->next;
159             head = head->next;
160             delete tempNode;
161         }
162     }
163 }
164 private:
165 Node* head;
166 //Creates a new node
167 Node* createNode(int value)
168 {
169     Node* n = new Node;
170     n->data = value;
171     return n;
172 }
173 };
174 List::List()
175 {
176     head = NULL;
177 }
178
179 List::~~List()
180 {
181     //deletes every node upon completion
182     while (head != NULL)
183     {
184         Node* temp = head;
185         head = head->next;
186         delete temp;
187     }

```

```

187         delete temp;
188     }
189 }
190 int main()
191 {
192     List l;
193     cout<<"===== "<<endl;
194     cout << "Added at first" << endl;
195     l.insertFirst(3);
196     l.insertFirst(2);
197     l.insertFirst(1);
198     l.display();
199
200     cout<<"===== "<<endl;
201     cout << "Added at last" << endl;
202     l.insertLast(4);
203     l.display();
204
205     cout<<"===== "<<endl;
206     cout << "Added at any" << endl;
207     l.insertAny(4, 5);
208     l.display();
209
210     cout<<"===== "<<endl;
211     cout << "After Enqueue" << endl;
212     l.Enqueue(1);
213     l.Enqueue(1);
214     l.Enqueue(3);
215     l.Enqueue(1);
216     l.Enqueue(2);
217     l.Enqueue(4);
218     l.Enqueue(1);
219     l.display();
220
221     cout<<"===== "<<endl;
222     cout << "After Normal Dequeue" << endl;
223     l.Dequeue();
224     l.display();
225     system("pause");
226     return 0;
227 }

```

Output:

E:\4th semester\Data Structure and Algorithms\Linked List\q3.exe

```

=====
Added at first
1 2 3
=====
Added at last
1 2 3 4
=====
Added at any
1 2 3 4 5
=====
After Enqueue
1 2 3 4 5 1 1 3 1 2 4 1
=====
After Normal Dequeue
2 3 4 5 1 1 3 1 2 4 1
Press any key to continue . . .

```