Bahria University,

Karachi Campus

LAB EXPERIMENT NO. _11_
LIST OF TASKS

TASK NO	OBJECTIVE
Task 1	Create a doubly link list and perform following operation. 1. Insert at beginning (). 2. Insert at end (). 3. Insert node at given data (). 4. Delete from start () 5. Delete from end () 6. Delete node of given data ().
Task 2	
Task 3	N/A
Task 4	N/A
Task 5	**N/A**
Task 6	**N/A**
Task 7	N/A
Task 8	N/A

Submitted On:

__12/06/2020__

(Date: DD/MM

Task No. 1 Create a doubly link list and perform following operation.

1. Insert at beginning ().
2. Insert at end ().
3. Insert node at given data ().
4. Delete from start ()
5. Delete from end ()

Delete node of given data ().

Coding:

Q1.cpp

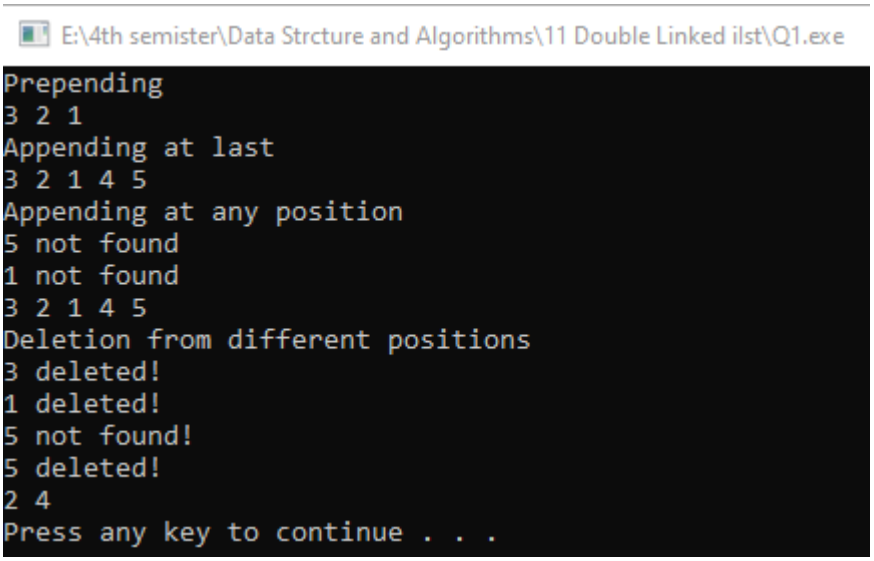
```cpp
#include <iostream>
typedef int ListType;
using namespace std;

class Node
{
public:
    Node();
    ~Node();

    ListType data;
    Node* next;
    Node* previous;
};

Node::Node()
{
    next = previous = NULL;
}

Node::~Node()
{
}

class DList
{
public:
    DList();
    ~DList();

    //Checks if empty
    bool isEmptyList()
    {
        if (head == NULL)
            return true;
        else
```

```
37              return false;
38          }
39          //Inserts at first position
40          void insertAtFirst(int  value)
41          {
42              if (isEmptyList())
43              {
44                  head = createNode(value);
45              }
46              else
47              {
48                  Node* newNode = createNode(value);
49
50                  //put head before newNode
51                  newNode->next = head;
52                  head->previous = newNode;
53
54                  //makes newNode head
55                  head = newNode;
56                  head->previous = NULL;
57              }
58          }
59          //Inserts at Last position
60          void insertAtLast(int value)
61          {
62              if (isEmptyList())
63              {
64                  head = createNode(value);
65              }
66              else
67              {
68                  Node* newNode = createNode(value);
69
70                  //traverses through list
71                  for (Node* tempNode = head; tempNode != NULL; tempNode = tempNode->next)
72                  {
73                      //When finds Last one
74                      if (tempNode->next == NULL)
75                      {
76                          Node* newNode = createNode(value);
77
78                          tempNode->next = newNode;
79                          newNode->previous = tempNode;
80
81                          break;
82                      }
83                  }
84
85              }
86          }
87          //Inserts at Any position
88          void insertAtAny(int oldValue, int newValue)
89          {
90              if (isEmptyList())
91              {
92                  cout << "List is empty" << endl;
93                  exit(0);
94              }
95              else
96              {
97                  for (Node* currentNode = head; currentNode != NULL; currentNode = currentNode->next)
98                  {
99                      //when Finds same value
100                     if (currentNode->data == oldValue)
101                     {
102                         Node* newNode = createNode(newValue);
```

```
103
104                     //Connects newNode to the next and previous one
105                     newNode->next = currentNode->next;
106                     newNode->previous = currentNode;
107
108                     //connects previous one to the newNode
109                     currentNode->next = newNode;
110                     break;
111                 }
112                 else
113                 {
114                     cout << oldValue <<" not found" << endl;
115                     break;
116                 }
117             }
118         }
119     }
120     //Deletes from any position
121     void deleteAtAny(int value)
122     {
123         if (isEmptyList())
124         {
125             cout << "List is empty..." << endl;
126         }
127         else
128         {
129             for (Node* currentNode = head; currentNode != NULL; currentNode = currentNode->next)
130             {
131                 //If first node is to be deleted
132                 if (head->data == value)
133                 {
134                     Node* tempNode = head;
135
136                     //updates head to next and delete previos value
137                     head = head->next;
137                     head = head->next;
138                     head->previous = NULL;
139
140                     delete tempNode;
141                     break;
142                 }//if()
143                 else if (currentNode->next->data == value)
144                 {
145                     if (currentNode->next->next == NULL)
146                     {
147                         Node* tempNode = currentNode->next;
148                         currentNode->next = NULL;
149
150                         delete tempNode;
151                         break;
152                     }
153
154                 }
155                 else if (currentNode->data == value)
156                 {
157                     Node* pre;
158                     Node* post;
159
160                     pre = currentNode->previous;
161                     post = currentNode->next;
162
163                     pre->next = post;
164                     post->previous = pre;
165
166                     delete currentNode;
167                     break;
168                 }
169                 else
170                 {
```

```
172                 }
173             }
174             cout << value << " deleted!" << endl;
175         }
176     }
177     //Traverse and display
178     void traverse()
179     {
180         for (Node* tempNode = head; tempNode != NULL; tempNode = tempNode->next)
181         {
182             cout << tempNode->data << " ";
183         }
184         cout << endl;
185     }
186
187 private:
188     Node* head;
189
190     //Creates a new Node
191     Node* createNode(int value)
192     {
193         Node* n = new Node;
194         n->data = value;
195
196         return n;
197     }
198 };
199
200 DList::DList()
201 {
202     head = NULL;
203 }
204
205 DList::~DList()
206 {
207 }
208
209 int main()
210 {
211     DList l;
212
213     cout << "Prepending" << endl;
214     l.insertAtFirst(1);
215     l.insertAtFirst(2);
216     l.insertAtFirst(3);
217     l.traverse();
218
219     cout << "Appending at last" << endl;
220     l.insertAtLast(4);
221     l.insertAtLast(5);
222     l.traverse();
223
224     cout << "Appending at any position" << endl;
225     l.insertAtAny(5, 6);
226     l.insertAtAny(1, 0);
227     l.traverse();
228
229     cout << "Deletion from different positions" << endl;
230     l.deleteAtAny(3);
231     l.deleteAtAny(1);
232     l.deleteAtAny(5);
233     l.traverse();
234
235
236     system("pause");
237     return 0;
238 }
```

Output:


E:\4th semister\Data Strcture and Algorithms\11 Double Linked ilst\Q1.exe
Prepending
3 2 1
Appending at last
3 2 1 4 5
Appending at any position
5 not found
1 not found
3 2 1 4 5
Deletion from different positions
3 deleted!
1 deleted!
5 not found!
5 deleted!
2 4
Press any key to continue . . .