

Name: Muhammad Anas BaigEnrollment No.: 01-134152-037Section: **BS(CS)-4A****LAB-JOURNAL-8****Exercise 1:**

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

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

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

    // Inserts a new node with value 'value' at position 'pos'
    in the list
    void insert (int pos, 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);

    // Deletes a node from position 'pos' of the list
    void deleteNode(int pos);

    // Deletes a node from the beginning of the list
    void delete_begin();

    // Deletes a node from the end of the list
    void delete_end();

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

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

cList.h File:

```
1. #include "Node.h"
2. #pragma once
3. class cList
4. {
5. public:
6.     Node *head;
7. public:
8.     cList(void);
9.     bool isEmpty();
10.
11.     void beginInsert(int);
12.     void midInsert(int, int); //count nodes and insert Node next to count == Z
13.     void endInsert(int);
14.
15.     void beginDelete();
16.     void midDelete(int); //delete node from position X
17.     void endDelete();
18.
19.     void displaycList();
20. };
```

cList.cpp File:

```
1. #include "cList.h"
2. #include "conio.h"
3. #include "Node.h"
4. #include <iostream>
5. using namespace std;
6.
7. cList::cList(void)
8. {
9.     head = '\0';
10. }
11.
12. bool cList::isEmpty()
```

```
13. {
14.     if(head == '\0')
15.     {
16.         return true;
17.     }
18.     else
19.     {
20.         return false;
21.     }
22. }
23.
24. void cList::beginInsert(int value) //inserts node at beginning
25. {
26.     Node *ptr;
27.     ptr = new Node;
28.     ptr->data = 0;
29.     ptr->next = '\0';
30.
31.     ptr->data = value;
32.     ptr->next = head;
33.
34.     if( !isEmpty() )
35.     {
36.         Node *temp = head;
37.         while( temp->next != head)
38.         {
39.             temp = temp->next;
40.         }
41.         temp->next = ptr;
42.     }
43.     else
44.     {
45.         ptr->next = ptr;
46.     }
47.     head = ptr;
48. }
49.
50. void cList::midInsert(int value, int countLocation) //count Nodes and insert Node next to count==Z
51. {
52.     if(!isEmpty())
53.     {
54.         Node *temp = head;
55.
56.         for( int i=1; i<countLocation; i++ )
57.         {
58.             temp = temp->next;
59.             if( temp == '\0' )
60.             {
61.                 cout<<"Sorry!!! No Node found at Count = "<<countLocation<<". "<<endl;
62.                 return;
63.             }
64.         }
65.         Node *ptr = new Node;
66.         ptr->data = value;
67.         ptr->next = temp->next;
68.         temp->next = ptr;
69.     }
70.     else
71.     {
72.         cout<<"Sorry!!! No data inserted-No count found-cList is empty"<<endl;
73.     }
74. }
75.
76. void cList::endInsert(int value)
77. {
```

```
78.     Node *ptr;
79.     ptr = new Node;
80.     ptr->data = 0;
81.     ptr->next = '\0';
82.
83.     ptr->data = value;
84.
85.     if( !isEmpty() )
86.     {
87.         Node *temp = head;
88.         while( temp->next != head)
89.         {
90.             temp = temp->next;
91.         }
92.         temp->next = ptr;
93.         ptr->next = head;
94.     }
95.     else
96.     {
97.         ptr->next = ptr;
98.         head = ptr;
99.     }
100. }
101.
102. void cList::beginDelete()
103. {
104.     if(!isEmpty())
105.     {
106.         Node *temp = head;
107.         Node *temp1 = head;
108.
109.         if( head->next == head ) //only one node in list
110.         {
111.             head = '\0';
112.             delete temp;
113.             return;
114.         }
115.         else
116.         {
117.             while( temp1->next != head )
118.             {
119.                 temp1 = temp1->next;
120.             }
121.             head = temp->next;
122.             temp1->next = head;
123.             delete temp;
124.         }
125.     }
126.     else
127.     {
128.         cout<<"Sorry!!! cList is empty."<<endl;
129.     }
130. }
131.
132. void cList::midDelete(int dataLocation)
133. {
134.     if(!isEmpty())
135.     {
136.         Node *prev;
137.         Node *temp;
138.         prev = 0;
139.         temp = head;
140.
141.         for( int i=1; i<dataLocation; i++ )
142.         {
```

```
143.         prev = temp;
144.         temp = temp->next;
145.
146.         if( temp == '\0' )
147.         {
148.             cout<<"Sorry!!! No Node found with Data Value = "<<dataLocation<<". "<<endl;
149.             return;
150.         }
151.     }
152.
153.     if( prev == 0 )
154.     {
155.         this->beginDelete();
156.         return;
157.     }
158.
159.     prev->next = temp->next;
160.     delete temp;
161. }
162. else
163. {
164.     cout<<"Sorry!!! cList is empty."<<endl;
165. }
166. }
167.
168. void cList::endDelete()
169. {
170.     if(!isEmpty())
171.     {
172.         Node *temp = head;
173.         Node *t;
174.
175.         if( head->next == head )
176.         {
177.             delete temp;
178.             head = '\0';
179.         }
180.         else
181.         {
182.             t = head->next;
183.             while( t->next != head )
184.             {
185.                 t = t->next;
186.                 temp = temp->next;
187.             }
188.             temp->next = head;
189.             delete t;
190.         }
191.     }
192.     else
193.     {
194.         cout<<"Sorry!!! cList is Empty."<<endl;
195.     }
196. }
197.
198. void cList::displaycList()
199. {
200.     if(!isEmpty())
201.     {
202.         Node *temp = head;
203.
204.         do
205.         {
206.             cout<<temp->data<<" ";
207.             temp = temp->next;
```

```

208.         }
209.         while(temp != head);
210.         cout<<endl;
211.     }
212.     else
213.     {
214.         cout<<"Sorry!!! cList is Empty."<<endl;
215.     }
216. }

```

Main.cpp File:

```

1. #include "cList.h"
2. #include "conio.h"
3. #include "Node.h"
4. #include <iostream>
5. using namespace std;
6.
7. void main()
8. {
9.     cList 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.displaycList();
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.displaycList();
35.     cout<<endl;
36.
37.     cout<<"DISPLAY AFTER-COUNTING NODES-
NODE INSERTION:"<<endl; //count nodes and insert Node next to count == Z
38.     cout<<"====="<<endl;
39.     l.midInsert(90, 3);
40.     l.displaycList();
41.     cout<<endl;
42.
43.     cout<<"DISPLAY AFTER-BEGIN NODE-DELETION:"<<endl;
44.     cout<<"====="<<endl;
45.     l.beginDelete();
46.     l.displaycList();
47.     cout<<endl;
48.
49.     cout<<"DISPLAY AFTER-MID NODE-DELETION:"<<endl; //delete node with Position == Z
50.     cout<<"====="<<endl;
51.     l.midDelete(3);

```

```

52.     l.displaycList();
53.     cout<<endl;
54.
55.     cout<<"DISPLAY AFTER-END NODE-DELETION:"<<endl;
56.     cout<<"===== "<<endl;
57.     l.endDelete();
58.     l.displaycList();
59.     cout<<endl;
60.
61.     getch();
62. }

```

Output:

```

C:\Users\Muhammad Anas Baig\Documents\Visual Studio 2010\Projects\Lab8-Ex1\Debug\Lab8-Ex...
LINKED LIST EMPTY CEHCK RESULT:
=====
Yes. List is Empty.
DISPLAY AFTER BEGIN NODE INSERTION:
=====
2 1
DISPLAY AFTER END NODE INSERTION:
=====
2 1 3 4
DISPLAY AFTER-COUNTING NODES-NODE INSERTION:
=====
2 1 3 90 4
DISPLAY AFTER-BEGIN NODE-DELETION:
=====
1 3 90 4
DISPLAY AFTER-MID NODE-DELETION:
=====
1 3 4
DISPLAY AFTER-END NODE-DELETION:
=====
1 3

```

Exercise 2 – Part 1:

Write C++ Function to Display the contents of alternate nodes of circular doubly linked list.

Solution:

Node.h File:

```

1. #pragma once
2. class Node
3. {
4. public:
5.     Node *prev;
6.     int data;
7.     Node *next;
8. public:

```

```
9.     Node(void);
10. };
```

Node.cpp File:

```
#include "Node.h"
```

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

circularDoublyList.h File:

```
1. #include "Node.h"
2.
3. #pragma once
4. class circularDoublyList
5. {
6. public:
7.     Node *head;
8. public:
9.     circularDoublyList(void);
10.    bool isEmpty();
11.    void insertBegin(int);
12.    void display();
13.    void displayAlternate();
14. };
```

circularDoublyList.cpp File:

```
1. #include "circularDoublyList.h"
2. #include "Node.h"
3. #include <iostream>
4. using namespace std;
5.
6. circularDoublyList::circularDoublyList(void)
7. {
8.     head = '\0';
9. }
10.
11. bool circularDoublyList::isEmpty()
12. {
13.     if( head == '\0' )
14.     {
15.         return true;
16.     }
17.     else
18.     {
19.         return false;
20.     }
21. }
22.
23. void circularDoublyList::insertBegin(int newVal)
24. {
25.     Node *ptr = new Node;
26.     ptr->prev = '\0';
27.     ptr->data = 0;
28.     ptr->next = '\0';
29.
30.     if( !isEmpty() )
31.     {
32.         ptr->data = newVal;
33.         ptr->next = head;
```



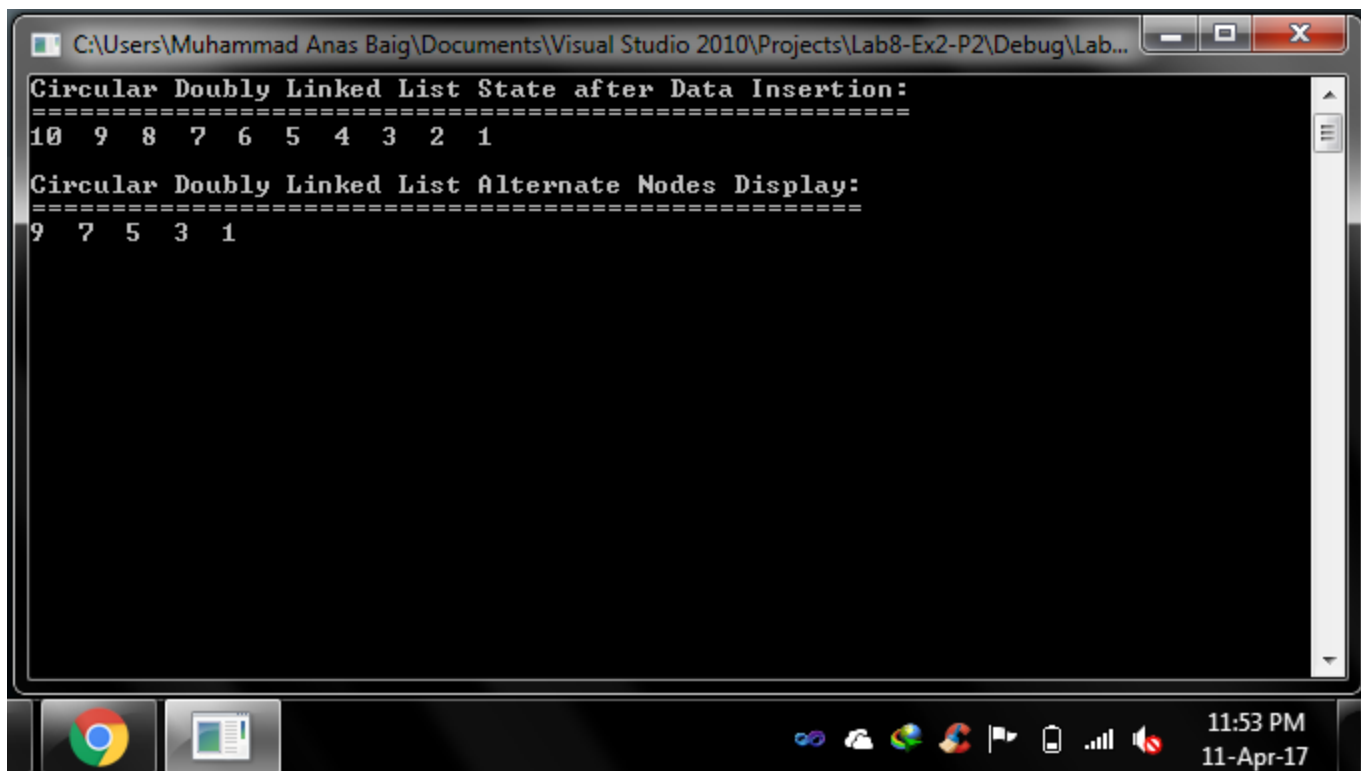
```
34.         head->prev = ptr;
35.         head = ptr;
36.     }
37.     else
38.     {
39.         ptr->data = newVal;
40.         head = ptr;
41.     }
42. }
43.
44. void circularDoublyList::display()
45. {
46.     if( !isEmpty() )
47.     {
48.         Node *temp = head;
49.
50.         while( temp != '\0' )
51.         {
52.             cout<<temp->data<<" ";
53.             temp = temp->next;
54.         }
55.         cout<<endl;
56.     }
57.     else
58.     {
59.         cout<<"SORRY!!! circularDoublyList is Empty."<<endl;
60.     }
61. }
62.
63. void circularDoublyList::displayAlternate()
64. {
65.     if( !isEmpty() )
66.     {
67.         Node *temp = head;
68.         int count = 0;
69.
70.         while( temp != '\0' )
71.         {
72.             count++;
73.             if(count%2 == 0)
74.             {
75.                 cout<<temp->data<<" ";
76.             }
77.             temp = temp->next;
78.         }
79.         cout<<endl;
80.     }
81.     else
82.     {
83.         cout<<"SORRY!!! circularDoublyList is Empty."<<endl;
84.     }
85. }
```

Main.cpp File:

```
1. #include "circularDoublyList.h"
2. #include "Node.h"
3. #include "conio.h"
4. #include <iostream>
5. using namespace std;
6.
7. void main()
8. {
9.     circularDoublyList c;
```

```
10.
11.     c.insertBegin(1);
12.     c.insertBegin(2);
13.     c.insertBegin(3);
14.     c.insertBegin(4);
15.     c.insertBegin(5);
16.     c.insertBegin(6);
17.     c.insertBegin(7);
18.     c.insertBegin(8);
19.     c.insertBegin(9);
20.     c.insertBegin(10);
21.
22.     cout<<"Circular Doubly Linked List State after Data Insertion:"<<endl;
23.     cout<<"===== "<<endl;
24.     c.display();
25.     cout<<endl;
26.
27.     cout<<"Circular Doubly Linked List Alternate Nodes Display:"<<endl;
28.     cout<<"===== "<<endl;
29.     c.displayAlternate();
30.
31.     getch();
32. }
```

Output:



```
C:\Users\Muhammad Anas Baig\Documents\Visual Studio 2010\Projects\Lab8-Ex2-P2\Debug\Lab...
Circular Doubly Linked List State after Data Insertion:
=====
10 9 8 7 6 5 4 3 2 1
Circular Doubly Linked List Alternate Nodes Display:
=====
9 7 5 3 1
```

Exercise 2 – Part 2:

Write C++ Function to Reverse a singly linked list with dummy header node using stack (rearrange links not just data).

Solution:

Node.h File:

```
1. #pragma once
2. class _Node
3. {
4. public:
5.     int data;
6.     _Node *prev;
7.     _Node *add;
8.     _Node *next;
9. public:
10.    _Node(void);
11. };;
```

Node.cpp File:

```
1. #include "_Node.h"
2.
3. _Node::_Node(void)
4. {
5. }
```

Stack.h File:

```
1. #include "_Node.h"
2.
3. #pragma once
4. class doublyListStack
5. {
6. public:
7.     _Node *top;
8. public:
9.     doublyListStack(void);
10.    bool isEmpty();
11.    void push( _Node * );
12.    _Node * pop();
13. };;
```

Stack.cpp File:

```
1. #include "doublyListStack.h"
2. #include "_Node.h"
3. #include <iostream>
4. using namespace std;
5.
6. doublyListStack::doublyListStack(void)
7. {
8.     top = '\0';
9. }
10.
11. bool doublyListStack::isEmpty()
12. {
13.     if( top == '\0' )
```

```

14.     {
15.         return true;
16.     }
17.     else
18.     {
19.         return false;
20.     }
21. }
22.
23. void doublyListStack::push(_Node * newVal)
24. {
25.     _Node *ptr = new _Node;
26.     ptr->prev = '\0';
27.     ptr->add = '\0';
28.     ptr->next = '\0';
29.
30.     if( !isEmpty() )
31.     {
32.         ptr->add = newVal;
33.         ptr->next = top;
34.         top->prev = ptr;
35.         top = ptr;
36.     }
37.     else
38.     {
39.         ptr->add = newVal;
40.         top = ptr;
41.     }
42. }
43.
44. _Node * doublyListStack::pop()
45. {
46.     if( !isEmpty() )
47.     {
48.         _Node *temp = top;
49.         _Node *tempadd = top->add;
50.
51.         if( top->next == '\0' )
52.         {
53.             top = '\0';
54.         }
55.         else
56.         {
57.             top = top->next;
58.             top->prev = '\0';
59.         }
60.         return (tempadd);
61.     }
62.     else
63.     {
64.         cout<<"SORRY!!! List is Empty."<<endl;
65.         return ('\0');
66.     }
67. }

```

List.h File:

```

1. #include "_Node.h"
2. #include "_Node.h"
3.
4. #pragma once
5. class list
6. {
7. public:

```

```
8.     _Node *head;
9. public:
10.    list(void);
11.    void add_Node(int);
12.    void display();
13.    void reverse();
14. };
```

List.cpp File:

```
1. #include "list.h"
2. #include "_Node.h"
3. #include "doublyListStack.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. void list::add_Node(int data)
14. {
15.     _Node *ptr = new _Node;
16.     ptr->next = '\0';
17.     ptr->data = 0;
18.
19.     ptr->data = data;
20.
21.     ptr->next = head->next;
22.     head->next = ptr;
23. }
24.
25. void list::display()
26. {
27.     if(head->next != '\0')
28.     {
29.         _Node *temp = head;
30.
31.         cout<<"List:"<<endl;
32.         cout<<"===="<<endl;
33.         while(temp->next != '\0')
34.         {
35.             temp = temp->next;
36.             cout<<temp->data<<" ";
37.         }
38.         cout<<endl;
39.     }
40.     else
41.     {
42.         cout<<"List is Empty."<<endl;
43.     }
44. }
45.
46. void list::reverse()
47. {
48.     _Node *temp = head->next;
49.     doublyListStack d;
50.
51.     while( temp != '\0' )
52.     {
53.         d.push( temp);
54.         temp = temp->next;
```

```
55.     }
56.
57.     head->next = d.pop();
58.     temp = head->next;
59.
60.     while( !d.isEmpty() )
61.     {
62.         temp->next = d.pop();
63.         temp = temp->next;
64.     }
65.     temp->next = '\\0';
66. }
```

Main.cpp File:

```
1. #include "list.h"
2. #include "doublyListStack.h"
3. #include "_Node.h"
4. #include "conio.h"
5. #include <iostream>
6. using namespace std;
7.
8. void main()
9. {
10.     list l;
11.     l.add_Node(1);
12.     l.add_Node(2);
13.     l.add_Node(3);
14.     l.add_Node(4);
15.     l.add_Node(5);
16.
17.     cout<<"BEFORE REVERSE:"<<endl;
18.     l.display();
19.     cout<<endl;
20.     cout<<endl;
21.
22.     l.reverse();
23.
24.     cout<<"AFTER REVERSE:"<<endl;
25.     l.display();
26.     cout<<endl;
27.
28.
29.     getch();
30. }
```

Output:

```
C:\Users\Muhammad Anas Baig\Documents\Visual Studio 2010\Projects\Lab8-Ex2-P1\Debug\mid...
BEFORE REVERSE:
List:
=====
5  4  3  2  1

AFTER REVERSE:
List:
=====
1  2  3  4  5
```