

TASK 01:

```
1  #include <iostream>
2
3  using namespace std;
4
5  template <typename T>
6  class Node{
7  public:
8      T data;
9      Node<T>* next=NULL;
10     Node(int value=0):data(value){};
11 };
12
13
14 template <typename T>
15 class linklist{
16 private:
17     Node<T>* head;
18     Node<T>* tail;
19 public:
20     linklist() : head(NULL), tail(head){}
21     linklist(int value) : head(new Node<T>(value)), tail(head){}
22
23     void add_node_at_tail(T value){
24         Node<T>* new_node=new Node<T>(value);
25         if(head==NULL){
26             head=new_node;
27             tail=head;
28             return;
29         }
30         tail->next=new_node;
31         tail=new_node;
32     }
33
34     void add_node_at_head(T value){
35         Node<T>* new_node=new Node<T>(value);
36         new_node->next=head;
37         head=new_node;
38         return;
39     }
40
41     Node<T>* get_head(void){
42         return head;
43     }
44
45     Node<T>* get_tail(void){
46         return tail;
47     }
48 }
```

```

void set_tail(Node<T>* tail){
    this->tail=tail;
    return;
}
void set_head(Node<T>* head){
    this->head=head;
    return;
}

Node<T>* order_even_odd(void){
    linklist<T> even_list;
    linklist<T> odd_list;
    Node<T>* curr=head;

    while(curr!=NULL){
        if(curr->data%2==0){
            even_list.add_node_at_tail(curr->data);
        } else {
            odd_list.add_node_at_tail(curr->data);
        }
        curr=curr->next;
    }
    Node<T>* even_tail=even_list.get_tail();
    Node<T>* odd_head=odd_list.get_head();
    even_tail->next=odd_head;
    return even_list.get_head();
}

void display(void){
    if(head==NULL){
        cout<<"Linklist is empty"<<endl;
        return;
    }
    Node<T>* curr=head;
    while(curr!=NULL){
        cout<<curr->data<<"->";
        curr=curr->next;
    }
    cout<<"nullptr"<<endl;
    return;
}

```

```

91
92     void display(Node<T>* head){
93         if(head==NULL){
94             cout<<"Linklist is empty"<<endl;
95             return;
96         }
97         Node<T>* curr=head;
98         while(curr!=NULL){
99             cout<<curr->data<<"->";
100             curr=curr->next;
101         }
102         cout<<"nullptr"<<endl;
103         return;
104     }
105
106 };
107
108 int main(){
109
110     linklist<int> list;
111     list.add_node_at_tail(17);
112     list.add_node_at_tail(15);
113     list.add_node_at_tail(8);
114     list.add_node_at_tail(12);
115     list.add_node_at_tail(10);
116     list.add_node_at_tail(5);
117     list.add_node_at_tail(4);
118     list.add_node_at_tail(1);
119     list.add_node_at_tail(7);
120     list.add_node_at_tail(6);
121
122     list.display();
123     Node<int>* ordered_list=list.order_even_odd();
124     list.display(ordered_list);
125     return 0;
126 }

```

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS

```

[Running] cd "c:\Users\phoni\OneDrive\Desktop\DS LAB 03\" && g++ Ta
17->15->8->12->10->5->4->1->7->6->nullptr
8->12->10->4->6->17->15->5->1->7->nullptr

[Done] exited with code=0 in 0.9 seconds

```

TASK 02:

```

#include <iostream>

using namespace std;

template <typename T>
class Node{
public:
    T data;
    Node<T>* next=NULL;
    Node(int value=0):data(value){};
};

template <typename T>
class linklist{
private:
    Node<T>* head;
    Node<T>* tail;
    int size=0;
public:
    linklist() : head(NULL), tail(head){}
    linklist(int value) : head(new Node<T>(value)), tail(head){}

    void add_node_at_tail(T value){
        Node<T>* new_node=new Node<T>(value);
        if(head==NULL){
            head=new_node;
            tail=head;
            return;
        }
        tail->next=new_node;
        tail=new_node;
        size++;
    }

    void add_node_at_head(T value){
        Node<T>* new_node=new Node<T>(value);
        new_node->next=head;
        head=new_node;
        size++;
    }

    Node<T>* get_head(void){
        return head;
    }

    Node<T>* get_tail(void){
        return tail;
    }
}

```

```

52     void set_tail(Node<T>* tail){
53         this->tail=tail;
54         return;
55     }
56     void set_head(Node<T>* head){
57         this->head=head;
58         return;
59     }
60
61     void display(void){
62         if(head==NULL){
63             cout<<"Linklist is empty"<<endl;
64             return;
65         }
66         Node<T>* curr=head;
67         while(curr!=NULL){
68             cout<<curr->data<<"->";
69             curr=curr->next;
70         }
71         cout<<"nullptr"<<endl;
72         return;
73     }
74
75     void display(Node<T>* head){
76         if(head==NULL){
77             cout<<"Linklist is empty"<<endl;
78             return;
79         }
80         Node<T>* curr=head;
81         while(curr!=NULL){
82             cout<<curr->data<<"->";
83             curr=curr->next;
84         }
85         cout<<"nullptr"<<endl;
86         return;
87     }

```

```

89     bool check_palindrome(void){
90         Node<T>* curr=head;
91         int index, checking_size=size;
92
93         while(curr!=NULL && checking_size!=size/2){
94             Node<T>* last_node;
95             last_node=curr;
96             index=0;
97             while(index<checking_size){
98                 last_node=last_node->next;
99                 index++;
100             }
101             if(curr->data!=last_node->data){
102                 return false;
103             }
104             curr=curr->next;
105             checking_size-=2;
106         }
107
108         return true;
109     }
110
111     int get_size(void){
112         return size;
113     }
114
115     void set_size(int size){
116         this->size=size;
117         return;
118     }
119
120     void get_user_input(void){
121         bool stop=false; T value; char choice;
122         system("cls");
123
124         while(!stop){
125             cout<<"Enter Node Value: ";
126             cin>>value;
127             add_node_at_tail(value);
128             cout<<"Continue List: Y/N: ";
129             cin>>choice;
130             if(choice=='n' || choice=='N'){
131                 stop=true;
132             }
133             system("cls");
134         }
135     }
136
137     ~:

```

```

138
139 int main(){
140
141     int type=0; bool flag;
142     cout<<"What Type Of Data Type For Link List: 1) char, 2) int :";
143     cin>>type;
144
145     switch (type) {
146     case 1: {
147         linklist<char> list;
148         list.get_user_input();
149         flag = list.check_palindrome();
150         list.display();
151         break;
152     }
153     case 2: {
154         linklist<int> list;
155         list.get_user_input();
156         flag = list.check_palindrome();
157         list.display();
158         break;
159     }
160     default:
161         cout << "Wrong Choice" << endl;
162         break;
163     }
164
165     if (flag){
166         cout<<"Linklist Is A Palindrom";
167     } else {
168         cout<<"Linklist Is Not A Palindrom" ;
169     }
170
171
172     return 0;
173 }

```

```

● PS C:\Users\phoni\OneDrive\Desktop\DS LAB 03> g++ Task02.cpp
○ PS C:\Users\phoni\OneDrive\Desktop\DS LAB 03> ./a.exe
What Type Of Data Type For Link List: 1) char, 2) int :2

```

```

1->2->3->4->3->2->1->nullptr
● Linklist Is A Palindrom
○ PS C:\Users\phoni\OneDrive\Desktop\DS LAB 03>

```

TASK 03:

```

1  #include <iostream>
2
3  using namespace std;
4
5  template <typename T>
6  class Node{
7      public:
8          T data;
9          Node<T>* next=NULL;
10         Node(int value=0):data(value){};
11 };
12
13
14 template <typename T>
15 class linklist{
16     private:
17         Node<T>* head;
18         Node<T>* tail;
19     public:
20         linklist() : head(NULL), tail(head){}
21         linklist(int value) : head(new Node<T>(value)), tail(head){}
22
23         void add_node_at_tail(T value){
24             Node<T>* new_node=new Node<T>(value);
25             if(head==NULL){
26                 head=new_node;
27                 tail=head;
28                 return;
29             }
30             tail->next=new_node;
31             tail=new_node;
32             tail->next=head;
33         }

```



```

35     void add_node_at_pos(T value,int pos){
36         Node<T>* new_node=new Node<T>(value);
37         if(head==NULL){
38             head=new_node;
39             tail=head;
40             return;
41         }
42         if(pos==0){
43             add_node_at_head(value);
44             return;
45         }
46         int count=0; Node<T>* curr=head;
47         while(count<pos-1 && curr!=tail){
48             if(curr==tail){
49                 cout<<"Error Wrong Position";
50                 return;
51             }
52             curr=curr->next;
53             count++;
54         }
55         if(curr==tail){
56             add_node_at_tail(value);
57             return;
58         }
59
60         Node<T>* prev_node=curr;
61         Node<T>* next_node=curr->next;
62
63         prev_node->next=new_node;
64         new_node->next=next_node;
65     }
66 }
67
68 void add_node_at_head(T value){
69     Node<T>* new_node=new Node<T>(value);
70     new_node->next=head;
71     head=new_node;
72     tail->next=head;
73     return;
74 }

```

```

75
76     void delete_node_at_pos(int pos){
77         if(head==NULL){
78             cout<<"Linklist Does Not Rush"<<endl;
79             return;
80         }
81         if(pos==0){
82             delete_node_at_head();
83             return;
84         }
85         int count=0; Node<T>* curr=head;
86         while(count<pos-1){
87             if(curr==tail){
88                 cout<<"Error Wrong Position";
89                 return;
90             }
91             curr=curr->next;
92             count++;
93         }
94         if(curr->next==tail){
95             delete_node_at_tail();
96             return;
97         }
98         Node<T>* prev_node=curr;
99         Node<T>* tmp_node=curr->next;
100         Node<T>* next_node=curr->next->next;
101         prev_node->next=next_node;
102
103         delete tmp_node;
104     }
105
106
107     void delete_node_at_head(void){
108         if(head==NULL){
109             cout<<"Linklist Does Not Rush"<<endl;
110             return;
111         }
112
113         Node<T>* tmp_node=head;
114         head=head->next;
115         tail->next=head;
116         delete tmp_node;
117     }

```

```

119 void delete_node_at_tail(void){
120     if(head==NULL){
121         cout<<"Linklist Does Not Rush"<<endl;
122         return;
123     }
124
125     Node<T>* tmp_node=tail;
126     Node<T>* curr=head;
127     while(curr->next!=tail){
128         curr=curr->next;
129     }
130     tail=curr;
131     tail->next=head;
132     delete tmp_node;
133 }
134
135 Node<T>* get_head(void){
136     return head;
137 }
138
139 Node<T>* get_tail(void){
140     return tail;
141 }
142
143
144 void set_tail(Node<T>* tail){
145     this->tail=tail;
146     return;
147 }
148 void set_head(Node<T>* head){
149     this->head=head;
150     return;
151 }
152
153 void display(void){
154     if(head==NULL){
155         cout<<"Linklist is empty"<<endl;
156         return;
157     }
158     Node<T>* curr=head;
159     while(curr!=tail){
160         cout<<curr->data<<"->";
161         curr=curr->next;
162     }
163     cout<<curr->data<<"->";
164     cout<<"repeat"<<endl;
165     return;
166 }

```

```
void display(Node<T>* head){
    if(head==NULL){
        cout<<"Linklist is empty"<<endl;
        return;
    }
    Node<T>* curr=head;
    while(curr!=tail){
        cout<<curr->data<<"->";
        curr=curr->next;
    }
    cout<<curr->data<<"->";
    cout<<"repeat"<<endl;
    return;
}
```

```
};
```

```

int main(){

    linklist<int> list;
    list.add_node_at_tail(0);
    list.add_node_at_tail(1);
    list.add_node_at_tail(2);
    list.add_node_at_tail(3);
    list.add_node_at_tail(4);
    list.add_node_at_tail(5);

    // display circular list
    list.display();

    // insert at end
    list.add_node_at_tail(0);
    list.display();

    // insert at start
    list.add_node_at_head(0);
    list.display();

    // insert at pos
    list.add_node_at_pos(0,6);
    list.display();

    //list.delete_node_at_head();
    // list.display();
    // list.delete_node_at_tail();
    // list.display();

    // delete node at any pos
    list.delete_node_at_pos(5);
    list.delete_node_at_pos(6);
    list.delete_node_at_pos(0);
    // list.delete_node_at_pos(10);

    list.display();

    return 0;
}

```

- PS C:\Users\phoni\OneDrive\Desktop\DS LAB 03> g++ Task03.cpp
- PS C:\Users\phoni\OneDrive\Desktop\DS LAB 03> ./a.exe
 - 0->1->2->3->4->5->repeat
 - 0->1->2->3->4->5->0->repeat
 - 0->0->1->2->3->4->5->0->repeat
 - 0->0->1->2->3->4->0->5->0->repeat
 - 0->1->2->3->0->0->repeat
- PS C:\Users\phoni\OneDrive\Desktop\DS LAB 03>

TASK 04:

```
task04.cpp > main()
1  #include <iostream>
2
3  using namespace std;
4
5  template <typename T>
6  class Node{
7      public:
8          T data;
9          Node<T>* next=NULL;
10         Node<T>* prev=NULL;
11         Node(int value=0):data(value){};
12 };
13
14
15 template <typename T>
16 class linklist{
17     private:
18         Node<T>* head;
19         Node<T>* tail;
20     public:
21         linklist() : head(NULL), tail(head){}
22         linklist(int value) : head(new Node<T>(value)), tail(head){}
23
24         void add_node_at_tail(T value){
25             Node<T>* new_node=new Node<T>(value);
26             if(head==NULL){
27                 head=new_node;
28                 tail=head;
29                 return;
30             }
31             tail->next=new_node;
32             new_node->prev=tail;
33             tail=new_node;
34             tail->next=head;
35             head->prev=tail;
36         }
37 }
```

```

38 void add_node_at_pos(T value,int pos){
39     Node<T>* new_node=new Node<T>(value);
40     if(head==NULL){
41         head=new_node;
42         tail=head;
43         return;
44     }
45     if(pos==0){
46         add_node_at_head(value);
47         return;
48     }
49     int count=0; Node<T>* curr=head;
50     while(count<pos-1 && curr!=tail){
51         if(curr==tail){
52             cout<<"Error Wrong Position";
53             return;
54         }
55         curr=curr->next;
56         count++;
57     }
58     if(curr==tail){
59         add_node_at_tail(value);
60         return;
61     }
62
63     Node<T>* prev_node=curr;
64     Node<T>* next_node=curr->next;
65
66     prev_node->next=new_node;
67     new_node->prev=prev_node->next;
68
69     new_node->next=next_node;
70     next_node->prev=new_node;
71
72 }
73
74 void add_node_at_head(T value){
75     Node<T>* new_node=new Node<T>(value);
76
77     new_node->next=head;
78     head->prev=new_node;
79
80     head=new_node;
81
82     tail->next=head;
83     head->prev=tail;
84     return;
85 }

```

```

87     void delete_node_at_pos(int pos){
88         if(head==NULL){
89             cout<<"Linklist Does Not Rush"<<endl;
90             return;
91         }
92         if(pos==0){
93             delete_node_at_head();
94             return;
95         }
96         int count=0; Node<T>* curr=head;
97         while(count<pos-1){
98             if(curr->next==tail){
99                 cout<<"Error Wrong Position";
100                 return;
101             }
102             curr=curr->next;
103             count++;
104         }
105         if(curr->next==tail){
106             delete_node_at_tail();
107             return;
108         }
109         Node<T>* prev_node=curr;
110         Node<T>* tmp_node=curr->next;
111         Node<T>* next_node=curr->next->next;
112
113         prev_node->next=next_node;
114         next_node->prev=prev_node;
115
116         delete tmp_node;
117     }
118
119     void delete_node_at_head(void){
120         if(head==NULL){
121             cout<<"Linklist Does Not Rush"<<endl<<endl;
122             return;
123         }
124
125         Node<T>* tmp_node=head;
126         head=head->next;
127         tail->next=head;
128         head->prev=tail;
129
130         delete tmp_node;
131     }
132

```



```

132
133     void delete_node_at_tail(void){
134         if(head==NULL){
135             cout<<"Linklist Does Not Rush"<<endl;
136             return;
137         }
138
139         Node<T>* tmp_node=tail;
140         Node<T>* curr=head;
141         while(curr->next!=tail){
142             curr=curr->next;
143         }
144         tail=curr;
145         tail->next=head;
146         head->prev=tail->next;
147
148         delete tmp_node;
149     }
150
151     Node<T>* get_head(void){
152         return head;
153     }
154
155     Node<T>* get_tail(void){
156         return tail;
157     }
158
159
160     void set_tail(Node<T>* tail){
161         this->tail=tail;
162         return;
163     }
164     void set_head(Node<T>* head){
165         this->head=head;
166         return;
167     }
168
169     void display(void){
170         if(head==NULL){
171             cout<<"Linklist is empty"<<endl;
172             return;
173         }
174         Node<T>* curr=head;
175         while(curr!=tail){
176             cout<<curr->data<<"->";
177             curr=curr->next;
178         }
179         cout<<curr->data<<"->";
180         cout<<"repeat"<<endl;
181         return;
182     }

```

```

183     void display_reverse(void){
184         if(tail==NULL){
185             cout<<"Linklist is empty"<<endl;
186             return;
187         }
188         Node<T>* curr=tail;
189         while(curr!=head){
190             cout<<curr->data<<"-";
191             curr=curr->prev;
192         }
193         cout<<curr->data<<"-";
194         cout<<"repeat"<<endl;
195         return;
196     }
197
198     void display_reverse(Node<T>* tail){
199         if(tail==NULL){
200             cout<<"Linklist is empty"<<endl;
201             return;
202         }
203         Node<T>* curr=tail;
204         while(curr!=head){
205             cout<<curr->data<<"-";
206             curr=curr->prev;
207         }
208         cout<<curr->data<<"-";
209         cout<<"repeat"<<endl;
210         return;
211     }
212
213     void display(Node<T>* head){
214         if(head==NULL){
215             cout<<"Linklist is empty"<<endl;
216             return;
217         }
218         Node<T>* curr=head;
219         while(curr!=tail){
220             cout<<curr->data<<"-";
221             curr=curr->next;
222         }
223         cout<<curr->data<<"-";
224         cout<<"repeat"<<endl;
225         return;
226     }
227
228
229
230 };
231

```

```

int main(){

    linklist<int> list;
    list.add_node_at_tail(0);
    list.add_node_at_tail(1);
    list.add_node_at_tail(2);
    list.add_node_at_tail(3);
    list.add_node_at_tail(4);
    list.add_node_at_tail(5);

    // display circular list
    list.display();
    list.display_reverse();
    cout<<endl<<endl;

    // // insert at end
    list.add_node_at_tail(-1);
    list.display();
    list.display_reverse();
    cout<<endl<<endl;

    // insert at start
    list.add_node_at_head(-1);
    list.display();
    list.display_reverse();
    cout<<endl<<endl;

    // insert at pos
    list.add_node_at_pos(0,6);
    list.display();

    // DELETE ADD HEAD
    list.delete_node_at_head();
    list.display();
    list.display_reverse();
    cout<<endl<<endl;

    // delete at tail
    list.delete_node_at_tail();
    list.display();
    list.display_reverse();
    cout<<endl<<endl;

    // delete node at any pos
    list.delete_node_at_pos(2);
    list.display();
    list.display_reverse();

    return 0;
}

```

```
[Running] cd "c:\Users\phoni\OneDrive\Desktop\DS LAB 03\" && g++ Task04.cpp -o Task04 -lbgil -lstdc++ -lwinpthread -loleaut32 -lole32 && "c:\Users\phoni\OneDrive\Desktop\DS LAB 03\Task04
0->1->2->3->4->5->repeat
5->4->3->2->1->0->repeat

0->1->2->3->4->5->1->repeat
-1->5->4->3->2->1->0->repeat

1->0->1->2->3->4->5->1->repeat
-1->5->4->3->2->1->0->1->repeat

1->0->1->0->2->3->4->5->1->repeat
-1->5->4->3->2->0->1->0->1->repeat

0->1->0->2->3->4->5->1->repeat
-1->5->4->3->2->0->1->0->repeat

0->1->0->3->4->5->1->repeat
-1->5->4->3->0->1->0->repeat

[Done] exited with code=0 in 0.864 seconds
```

TASK 05:

```
1  #include <iostream>
2
3  using namespace std;
4
5  template <typename T>
6  class Doubly_Node{
7  public:
8      T data;
9      Doubly_Node<T>* next=NULL;
10     Doubly_Node<T>* prev=NULL;
11     Doubly_Node(int value=0):data(value){};
12 };
13
14 template <typename T>
15 class Singly_Node{
16 public:
17     T data;
18     Singly_Node<T>* next=NULL;
19     Singly_Node(int value=0):data(value){};
20 };
21
22
23 template <typename T>
24 class Doubly_linklist{
25 private:
26     Doubly_Node<T>* head;
27     Doubly_Node<T>* tail;
28 public:
29     Doubly_linklist() : head(NULL), tail(head){}
30     Doubly_linklist(int value) : head(new Doubly_Node<T>(value)), tail(head){}
31
32     void add_Doubly_node_at_tail(T value){
33         Doubly_Node<T>* new_Doubly_node=new Doubly_Node<T>(value);
34         if(head==NULL){
35             head=new_Doubly_node;
36             tail=head;
37             return;
38         }
39         tail->next=new_Doubly_node;
40         new_Doubly_node->prev=tail;
41         tail=new_Doubly_node;
42         tail->next=head;
43         head->prev=tail;
44     }
45 }
```

```

46 void add_Doubly_node_at_pos(T value,int pos){
47     Doubly_Node<T>* new_Doubly_node=new Doubly_Node<T>(value);
48     if(head==NULL){
49         head=new_Doubly_node;
50         tail=head;
51         return;
52     }
53     if(pos==0){
54         add_Doubly_node_at_head(value);
55         return;
56     }
57     int count=0; Doubly_Node<T>* curr=head;
58     while(count<pos-1 && curr!=tail){
59         if(curr==tail){
60             cout<<"Error Wrong Position";
61             return;
62         }
63         curr=curr->next;
64         count++;
65     }
66     if(curr==tail){
67         add_Doubly_node_at_tail(value);
68         return;
69     }
70
71     Doubly_Node<T>* prev_Doubly_node=curr;
72     Doubly_Node<T>* next_Doubly_node=curr->next;
73
74     prev_Doubly_node->next=new_Doubly_node;
75     new_Doubly_node->prev=prev_Doubly_node->next;
76
77     new_Doubly_node->next=next_Doubly_node;
78     next_Doubly_node->prev=new_Doubly_node;
79
80 }
81
82 void add_Doubly_node_at_head(T value){
83     Doubly_Node<T>* new_Doubly_node=new Doubly_Node<T>(value);
84
85     new_Doubly_node->next=head;
86     head->prev=new_Doubly_node;
87
88     head=new_Doubly_node;
89
90     tail->next=head;
91     head->prev=tail;
92     return;
93 }

```

```

95 void delete_Doubly_node_at_pos(int pos){
96     if(head==NULL){
97         cout<<"Doubly_Linklist Does Not Rush"<<endl;
98         return;
99     }
100     if(pos==0){
101         delete_Doubly_node_at_head();
102         return;
103     }
104     int count=0; Doubly_Node<T>* curr=head;
105     while(count<pos-1){
106         if(curr->next==tail){
107             cout<<"Error Wrong Position";
108             return;
109         }
110         curr=curr->next;
111         count++;
112     }
113     if(curr->next==tail){
114         delete_Doubly_node_at_tail();
115         return;
116     }
117     Doubly_Node<T>* prev_Doubly_node=curr;
118     Doubly_Node<T>* tmp_Doubly_node=curr->next;
119     Doubly_Node<T>* next_Doubly_node=curr->next->next;
120
121     prev_Doubly_node->next=next_Doubly_node;
122     next_Doubly_node->prev=prev_Doubly_node;
123
124     delete tmp_Doubly_node;
125 }
126
127 void delete_Doubly_node_at_head(void){
128     if(head==NULL){
129         cout<<"Doubly_Linklist Does Not Rush"<<endl<<endl;
130         return;
131     }
132
133     Doubly_Node<T>* tmp_Doubly_node=head;
134     head=head->next;
135     tail->next=head;
136     head->prev=tail;
137
138     delete tmp_Doubly_node;
139 }
140

```

```

141 void delete_Doubly_node_at_tail(void){
142     if(head==NULL){
143         cout<<"Doubly_Linklist Does Not Rush"<<endl;
144         return;
145     }
146
147     Doubly_Node<T>* tmp_Doubly_node=tail;
148     Doubly_Node<T>* curr=head;
149     while(curr->next!=tail){
150         curr=curr->next;
151     }
152     tail=curr;
153     tail->next=head;
154     head->prev=tail->next;
155
156     delete tmp_Doubly_node;
157 }
158
159 Doubly_Node<T>* get_head(void){
160     return head;
161 }
162
163 Doubly_Node<T>* get_tail(void){
164     return tail;
165 }
166
167
168 void set_tail(Doubly_Node<T>* tail){
169     this->tail=tail;
170     return;
171 }
172 void set_head(Doubly_Node<T>* head){
173     this->head=head;
174     return;
175 }
176
177 void display(void){
178     if(head==NULL){
179         cout<<"Doubly_Linklist is empty"<<endl;
180         return;
181     }
182     Doubly_Node<T>* curr=head;
183     while(curr!=tail){
184         cout<<curr->data<<"->";
185         curr=curr->next;
186     }
187     cout<<curr->data<<"->";
188     cout<<"repeat"<<endl;
189     return;
190 }

```

```

191 void display_reverse(void){
192     cout<<"Doubly_Linklist is empty"<<endl;
193     return;
194 }
195 Doubly_Node<T>* curr=tail;
196 while(curr!=head){
197     cout<<curr->data<<"-";
198     curr=curr->prev;
199 }
200 cout<<curr->data<<"-";
201 cout<<"repeat"<<endl;
202 return;
203 }
204
205 void display_reverse(Doubly_Node<T>* tail){
206     if(tail==NULL){
207         cout<<"Doubly_Linklist is empty"<<endl;
208         return;
209     }
210     Doubly_Node<T>* curr=tail;
211     while(curr!=head){
212         cout<<curr->data<<"-";
213         curr=curr->prev;
214     }
215     cout<<curr->data<<"-";
216     cout<<"repeat"<<endl;
217     return;
218 }
219
220 void display(Doubly_Node<T>* head){
221     if(head==NULL){
222         cout<<"Doubly_Linklist is empty"<<endl;
223         return;
224     }
225     Doubly_Node<T>* curr=head;
226     while(curr!=tail){
227         cout<<curr->data<<"-";
228         curr=curr->next;
229     }
230     cout<<curr->data<<"-";
231     cout<<"repeat"<<endl;
232     return;
233 }
234
235
236 };
237

```



```

239
240 template <typename T>
241 class Singly_linklist{
242     private:
243         Singly_Node<T>* head;
244         Singly_Node<T>* tail;
245     public:
246         Singly_linklist() : head(NULL), tail(head){}
247         Singly_linklist(int value) : head(new Singly_Node<T>(value)), tail(head){}
248
249         void add_Singly_node_at_tail(T value){
250             Singly_Node<T>* new_Singly_node=new Singly_Node<T>(value);
251             if(head==NULL){
252                 head=new_Singly_node;
253                 tail=head;
254                 return;
255             }
256             tail->next=new_Singly_node;
257             tail=new_Singly_node;
258         }
259
260         void add_Singly_node_at_head(T value){
261             Singly_Node<T>* new_Singly_node=new Singly_Node<T>(value);
262             new_Singly_node->next=head;
263             head=new_Singly_node;
264             return;
265         }
266
267         Singly_Node<T>* get_head(void){
268             return head;
269         }
270
271         Singly_Node<T>* get_tail(void){
272             return tail;
273         }
274
275
276         void set_tail(Singly_Node<T>* tail){
277             this->tail=tail;
278             return;
279         }
280         void set_head(Singly_Node<T>* head){
281             this->head=head;
282             return;
283         }

```

```

285     Singly_Node<T>* order_even_odd(void){
286         Singly_linklist<T> even_list;
287         Singly_linklist<T> odd_list;
288         Singly_Node<T>* curr=head;
289
290         while(curr!=NULL){
291             if(curr->data%2==0){
292                 even_list.add_Singly_node_at_tail(curr->data);
293             } else {
294                 odd_list.add_Singly_node_at_tail(curr->data);
295             }
296             curr=curr->next;
297         }
298         Singly_Node<T>* even_tail=even_list.get_tail();
299         Singly_Node<T>* odd_head=odd_list.get_head();
300         even_tail->next=odd_head;
301         return even_list.get_head();
302     }
303
304     void display(void){
305         if(head==NULL){
306             cout<<"Singly_Linklist is empty"<<endl;
307             return;
308         }
309         Singly_Node<T>* curr=head;
310         while(curr!=NULL){
311             cout<<curr->data<<"-";
312             curr=curr->next;
313         }
314         cout<<"nullptr"<<endl;
315         return;
316     }
317
318     void display(Singly_Node<T>* head){
319         if(head==NULL){
320             cout<<"Singly_Linklist is empty"<<endl;
321             return;
322         }
323         Singly_Node<T>* curr=head;
324         while(curr!=NULL){
325             cout<<curr->data<<"-";
326             curr=curr->next;
327         }
328         cout<<"nullptr"<<endl;
329         return;
330     }

```

```

332
333     Singly_linklist<T> convert_to_singly_list(Doubly_linklist<T> list){
334         Doubly_Node<T>* doubly_curr=list.get_head();
335         Doubly_Node<T>* doubly_tail=list.get_tail();
336
337         Singly_linklist<T> new_list;
338
339         if(doubly_curr==NULL){
340             cout<<"Doubly list is empty"<<endl;
341             return new_list;
342         }
343
344         while(doubly_curr!=doubly_tail){
345             new_list.add_Singly_node_at_tail(doubly_curr->data);
346             doubly_curr=doubly_curr->next;
347         }
348         new_list.add_Singly_node_at_tail(doubly_curr->data);
349
350
351         return new_list;
352     }
353
354     Singly_linklist<T> operator =(Singly_linklist<T> list){
355         this->head=list.head;
356         return *this;
357     }
358
359
360 };

```

```

5  int main(){
6
7      Doubly_linklist<int> doubly_list;
8      doubly_list.add_Doubly_node_at_tail(0);
9      doubly_list.add_Doubly_node_at_tail(1);
10     doubly_list.add_Doubly_node_at_tail(2);
11     doubly_list.add_Doubly_node_at_tail(3);
12     doubly_list.add_Doubly_node_at_tail(4);
13     doubly_list.add_Doubly_node_at_tail(5);
14
15     // display circular list
16     doubly_list.display();
17     doubly_list.display_reverse();
18     cout<<endl<<endl;
19
20     Singly_linklist<int> singly_list;
21     singly_list=singly_list.convert_to_singly_list(doubly_list);
22     singly_list.display();
23
24     return 0;
25 }

```

```

[Running] cd "c:\Users\phoni\OneDrive\Desktop\DS LAB 03\" && g++ Task05.cpp -o Task05 -lbg1 -lstdc++ -lwinid -loleaut32 -lole32 && "c:\Users\phoni\OneDrive\Desktop\DS LAB 03\"Task05
0->1->2->3->4->5->repeat
5->4->3->2->1->0->repeat

0->1->2->3->4->5->nullptr

```

TASK 06:

```
#include <iostream>

using namespace std;

template <typename T>
class Node{
public:
    T data;
    Node<T>* next=NULL;
    Node<T>* prev=NULL;
    Node(int value=0):data(value){};
};

template <typename T>
class linklist{
private:
    Node<T>* head;
    Node<T>* tail;
public:
    linklist() : head(NULL), tail(head){}
    linklist(int value) : head(new Node<T>(value)), tail(head){}

    void add_node_at_tail(T value){
        Node<T>* new_node=new Node<T>(value);
        if(head==NULL){
            head=new_node;
            tail=head;
            return;
        }
        tail->next=new_node;
        new_node->prev=tail;
        tail=new_node;
        tail->next=head;
        head->prev=tail;
    }

    void add_node_at_pos(T value,int pos){
        Node<T>* new_node=new Node<T>(value);
        if(head==NULL){
            head=new_node;
            tail=head;
            return;
        }
    }
}
```

```

        if(pos==0){
            add_node_at_head(value);
            return;
        }
        int count=0; Node<T>* curr=head;
        while(count<pos-1 && curr!=tail){
            if(curr==tail){
                cout<<"Error Wrong Position";
                return;
            }
            curr=curr->next;
            count++;
        }
        if(curr==tail){
            add_node_at_tail(value);
            return;
        }

        Node<T>* prev_node=curr;
        Node<T>* next_node=curr->next;

        prev_node->next=new_node;
        new_node->prev=prev_node->next;

        new_node->next=next_node;
        next_node->prev=new_node;

    }

    void add_node_at_head(T value){
        Node<T>* new_node=new Node<T>(value);

        new_node->next=head;
        head->prev=new_node;

        head=new_node;

        tail->next=head;
        head->prev=tail;
        return;
    }

    Node<T>* get_head(void){
        return head;
    }

    Node<T>* get_tail(void){
        return tail;
    }

```

```

}

void set_tail(Node<T>* tail){
    this->tail=tail;
    return;
}
void set_head(Node<T>* head){
    this->head=head;
    return;
}

void display(void){
    if(head==NULL){
        cout<<"Linklist is empty"<<endl;
        return;
    }
    Node<T>* curr=head;
    while(curr!=tail){
        cout<<curr->data<<"->";
        curr=curr->next;
    }
    cout<<curr->data<<"->";
    cout<<"repeat"<<endl;
    return;
}

void display_reverse(void){
    if(tail==NULL){
        cout<<"Linklist is empty"<<endl;
        return;
    }
    Node<T>* curr=tail;
    while(curr!=head){
        cout<<curr->data<<"->";
        curr=curr->prev;
    }
    cout<<curr->data<<"->";
    cout<<"repeat"<<endl;
    return;
}

void display_reverse(Node<T>* tail){
    if(tail==NULL){
        cout<<"Linklist is empty"<<endl;
        return;
    }
    Node<T>* curr=tail;
    while(curr!=head){

```

```

        cout<<curr->data<<"-";
        curr=curr->prev;
    }
    cout<<curr->data<<"-";
    cout<<"repeat"<<endl;
    return;
}

void display(Node<T>* head){
    if(head==NULL){
        cout<<"Linklist is empty"<<endl;
        return;
    }
    Node<T>* curr=head;
    while(curr!=tail){
        cout<<curr->data<<"-";
        curr=curr->next;
    }
    cout<<curr->data<<"-";
    cout<<"repeat"<<endl;
    return;
}

void swap_nodes(int pos01,int pos02){
    if(head==tail){
        cout<<"List Too Short"<<endl;
        return;
    }

    int cnt01=0;
    int cnt02=0;
    Node<T>* node01=head;
    Node<T>* node02=head;

    while(cnt01<pos01 || cnt02<pos02){
        if(cnt01<pos01){
            node01=node01->next;
            cnt01++;
        }
        if(cnt02<pos02){
            node02=node02->next;
            cnt02++;
        }
    }

    if (node01==head || node01==tail || node02==head || node02==tail) {
        if (node01==head && node02!=tail) {

```

```

        swap_head(pos02);
    } else if (node02==head && node01!=tail) {
        swap_head(pos01);
    } else if (node01==tail && node02!=head) {
        swap_tail(pos02);
    } else if (node02==tail && node01!=head) {
        swap_tail(pos01);
    } else if ((node01==head && node02==tail) || (node01==tail &&
node02==head)) {
        swap_head_tail();
    }
    return;
}

if(node01->next==node02 && node02->prev==node01){
    swap_adjacent(node01,node02);
}

if(node02->next==node01 && node01->prev==node02){
    swap_adjacent(node02,node01);
}

Node<T>* prev01=node01->prev;
Node<T>* next01=node01->next;
Node<T>* prev02=node02->prev;
Node<T>* next02=node02->next;

prev01->next=node02;
next01->prev=node02;
node02->next=next01;
node02->prev=prev01;

prev02->next=node01;
next02->prev=node01;
node01->next=next02;
node01->prev=prev02;

}

void swap_head(int pos) {
    if (head==NULL) {
        cout<<"List is empty"<<endl;
        return;
    }

    if (pos==0) {
        cout<<"Position is the head, no swap needed."<<endl;
        return;
    }

```



```

    }

    Node<T>* node01=head;
    Node<T>* node02=head;
    int cnt=0;

    while (cnt<pos) {
        node02=node02->next;
        cnt++;
    }

    if (node02==tail) {
        swap_head_tail();
        return;
    }

    Node<T>* prev01=node01->prev;
    Node<T>* prev02=node02->prev;
    Node<T>* next01=node01->next;
    Node<T>* next02=node02->next;

    prev01->next=node02;
    next01->prev=node02;
    node02->next=next01;
    node02->prev=prev01;

    prev02->next=node01;
    next02->prev=node01;
    node01->next=next02;
    node01->prev=prev02;

    head=node02;
    tail->next=head;
    head->prev=tail;
}

void swap_tail(int pos) {
    if (head==NULL) {
        cout<<"List is empty"<<endl;
        return;
    }

    if (pos==0) {
        swap_head_tail();
        return;
    }
}

```

```

Node<T>* node01=head;
Node<T>* node02=tail;
int cnt=0;

while (cnt<pos) {
    node01=node01->next;
    cnt++;
}

if (node01==head) {
    swap_head_tail();
    return;
}

if (node01==tail) {
    cout<<"Node is already the head."<<endl;
    return;
}

Node<T>* prev01=node01->prev;
Node<T>* next01=node01->next;
Node<T>* prev02=tail->prev;
Node<T>* next02=tail->next;

tail->next=next01;
tail->prev=prev01;
prev01->next=tail;
next01->prev=tail;
tail=node01;

prev02->next=node01;
next02->prev=node01;
node01->next=next02;
node01->prev=prev02;

}

void swap_head_tail() {
    if (head==NULL || head==tail) {
        cout<<"List is too short to swap head and tail."<<endl;
        return;
    }

    Node<T>* temp_head=head;
    Node<T>* temp_tail=tail;

```

```

Node<T>* prev_head=head->prev;
Node<T>* next_head=head->next;
Node<T>* prev_tail=tail->prev;
Node<T>* next_tail=tail->next;

head=temp_tail;
tail=temp_head;

head->next=next_head;
head->prev=prev_head;
next_head->prev=head;
head->prev=tail;

tail->next=next_tail;
tail->prev=prev_tail;
prev_tail->next=tail;
tail->next=head;
}

void swap_adjacent(Node<T>* node1, Node<T>* node2) {
    if(node1==NULL || node2==NULL) {
        cout<<"Invalid nodes."<<endl;
        return;
    }

    if((node1==tail && node2==head) || (node2==tail && node1==head)) {
        swap_head_tail();
        return;
    }

    Node<T>* prev1=node1->prev;
    Node<T>* next2=node2->next;

    prev1->next=node2;
    next2->prev=node1;

    node2->prev=prev1;
    node2->next=node1;

    node1->prev=node2;
    node1->next=next2;

    if (node1==head) {
        head=node2;
    } else if (node2==tail) {
        tail=node1;
    }
}

```

```

    }

};

int main(){

    linklist<int> list;
    list.add_node_at_tail(1);
    list.add_node_at_tail(7);
    list.add_node_at_tail(4);
    list.add_node_at_tail(2);
    list.add_node_at_tail(6);
    list.add_node_at_tail(4);
    list.add_node_at_tail(5);
    list.add_node_at_tail(3);
    list.add_node_at_tail(9);
    list.add_node_at_tail(8);

    int pos01=0,pos02=0;
    cout<<"Input position 01: ";
    cin>>pos01;
    cout<<"Input position 02: ";
    cin>>pos02;

    cout<<endl<<"Before Swaping: "<<endl;
    list.display();
    list.display_reverse();
    cout<<endl;
    list.swap_nodes(pos01,pos02);
    cout<<endl<<"After Swaping: "<<endl;
    list.display();
    list.display_reverse();
    cout<<endl<<endl;

    return 0;
}

```

```

PS C:\Users\phoni\OneDrive\Desktop\DS LAB 03> ./a.exe
Input position 01: 3
Input position 02: 7

Before Swaping:
1->7->4->2->6->4->5->3->9->8->repeat
8->9->3->5->4->6->2->4->7->1->repeat

After Swaping:
1->7->4->3->6->4->5->2->9->8->repeat
8->9->2->5->4->6->3->4->7->1->repeat

PS C:\Users\phoni\OneDrive\Desktop\DS LAB 03>

```

TASK 07:

```

#include <iostream>

using namespace std;

template <typename T>
class Node{
public:
    T data;
    Node<T>* next=nullptr;
    Node<T>* prev=nullptr;
    Node(int value=0):data(value){};
};

template <typename T>
class linklist{
private:
    Node<T>* head;
    Node<T>* tail;
public:
    linklist() : head(nullptr), tail(head){}
    linklist(int value) : head(new Node<T>(value)), tail(head){}

    void add_node_at_pos(T value,int pos){
        Node<T>* new_node=new Node<T>(value);
        if(head==nullptr){
            head=new_node;
            tail=head;
            return;
        }
        if(pos==0){
            add_node_at_head(value);

```

```

        return;
    }
    int count=0; Node<T>* curr=head;
    while(count<pos-1 && curr!=tail){
        if(curr==tail){
            cout<<"Error Wrong Position";
            return;
        }
        curr=curr->next;
        count++;
    }
    if(curr==tail){
        add_node_at_tail(value);
        return;
    }

    Node<T>* prev_node=curr;
    Node<T>* next_node=curr->next;

    prev_node->next=new_node;
    new_node->prev=prev_node->next;

    new_node->next=next_node;
    next_node->prev=new_node;

}

void add_node_at_head(T value){
    Node<T>* new_node=new Node<T>(value);

    new_node->next=head;
    head->prev=new_node;

    head=new_node;

    tail->next=head;
    head->prev=nullptr;
    return;
}

Node<T>* get_head(void){
    return head;
}

Node<T>* get_tail(void){
    return tail;
}

```

```

void set_tail(Node<T>* tail){
    this->tail=tail;
    return;
}

void set_head(Node<T>* head){
    this->head=head;
    return;
}

void display(void){
    if(head==nullptr){
        cout<<"Linklist is empty"<<endl;
        return;
    }
    Node<T>* curr=head;
    while(curr){
        cout<<curr->data<<"-";
        curr=curr->next;
    }
    cout<<"nullptr"<<endl;
    return;
}

void display_reverse(void){
    if(tail==nullptr){
        cout<<"Linklist is empty"<<endl;
        return;
    }
    Node<T>* curr=tail;
    while(curr){
        cout<<curr->data<<"-";
        curr=curr->prev;
    }
    cout<<"nullptr"<<endl;
    return;
}

void display_reverse(Node<T>* tail){
    if(tail==nullptr){
        cout<<"Linklist is empty"<<endl;
        return;
    }
    Node<T>* curr=tail;
    while(curr){
        cout<<curr->data<<"-";
        curr=curr->prev;
    }
    cout<<"nullptr"<<endl;
}

```

```

        return;
    }

    void display(Node<T>* head){
        if(head==nullptr){
            cout<<"Linklist is empty"<<endl;
            return;
        }
        Node<T>* curr=head;
        while(curr){
            cout<<curr->data<<"-";
            curr=curr->next;
        }
        cout<<"nullptr"<<endl;
        return;
    }

    linklist<T> operator =(linklist<T> list){
        this->head=list.head;
        this->tail=list.tail;
        return *this;
    }

    void reverse(void){
        Node<T>* prevptr=nullptr;
        Node<T>* curr=head;
        while(curr->next){
            Node<T>* nextptr=curr->next;
            curr->prev=nextptr;
            curr->next=prevptr;
            prevptr=curr;
            curr=nextptr;
        }

        tail=head;

        head=prevptr->prev;
        head->next=prevptr;

        head->prev=nullptr;
        tail->next=nullptr;
    }

    void add_node_at_tail(T value){
        Node<T>* new_node=new Node<T>(value);
        if(head==nullptr){

```



```

        head=new_node;
        tail=head;
        return;
    }
    tail->next=new_node;
    new_node->prev=tail;
    tail=new_node;
    tail->next=nullptr;
}

void delete_node_at_head(void){
    if(head==nullptr){
        cout<<"Linklist Does Not Rush"<<endl<<endl;
        return;
    }

    Node<T>* tmp_node=head;
    head=head->next;
    tail->next=head;
    head->prev=nullptr;

    delete tmp_node;
}

void delete_node_at_tail(void){
    if(head==nullptr){
        cout<<"Linklist Does Not Rush"<<endl;
        return;
    }

    Node<T>* tmp_node=tail;
    tail=tail->prev;
    tail->next=nullptr;

    delete tmp_node;
}

void delete_node(Node<T>* &node){
    if(head==nullptr){
        cout<<"Linklist Does Not Rush"<<endl;
        return;
    }
    if(node==head){
        delete_node_at_head();
        return;
    }
    if(node==tail){

```

```

        delete_node_at_tail();
        return;
    }

    Node<T>* tmp=node;
    Node<T>* prev=node->prev;
    Node<T>* next=node->next;

    prev->next=next;
    next->prev=prev;

    node=node->next;
    delete tmp;
}

void aternate_list(void){
    if(head==nullptr){
        cout<<"Linklist Does Not Rush"<<endl;
        return;
    }

    Node<T>* curr=head;
    int index=0;
    while(curr!=tail){
        Node<T>* next=curr->next;
        if(index++%2==1){
            add_node_at_tail(curr->data);
            delete_node(curr);
        }
        curr=next;
    }
}

void get_user_input(void){
    bool stop=false; T value; char choice;
    system("cls");

    while(!stop){
        cout<<"Enter Node Value: ";
        cin>>value;
        add_node_at_tail(value);
        cout<<"Continue List: Y/N: ";
        cin>>choice;
        if(choice=='n' || choice=='N'){
            stop=true;
        }
        system("cls");
    }
}

```

```

        }
    }

};

int main(){
    linklist<int> list;

    list.get_user_input();

    list.aternate_list();
    list.display();

    return 0;
}

```

```

10->9->3->9->4->5->1->4->nullptr
> PS C:\Users\phoni\OneDrive\Desktop\DS LAB 03>

```