

```

1  #include<iostream>
2  #include<string>
3  // #include "ll.h"
4  using namespace std;
5
6  template <typename T>
7  class linkedlist {
8  public:
9      class node
10     {
11         friend class linkedlist<T>;
12
13         public:
14             T data;
15             node* next;
16             node(T d, node* n= NULL): data(d), next(n) {}
17             //node(node*n=NULL): next(n){}
18     };
19
20     node* head;
21     node* tail;
22
23     public:
24         int count;
25         // Default Constructor
26         linkedlist(void): head(NULL), tail(NULL), count(0) {}
27         // Copy Constructor
28         linkedlist(const linkedlist<T>& src);
29         // Destructor
30         ~linkedlist(void);
31         int size(void) { return count;}
32         bool empty(void) {return count==0 ;}
33         void push_back(T);
34         void push_front(T);
35         void pop_front(void);
36         void pop_back(void);
37         void display(void);
38         linkedlist<T> merge( linkedlist<T> &l, linkedlist<T> &f);
39 };
40
41 // Insert elements from behind
42 template <typename T>
43 void linkedlist<T>::push_back(T d)
44 {
45     node* temp = new node(d, NULL);
46     if(this->empty())
47     {
48         head = temp;
49     }
50     else
51     {
52         tail->next = temp;
53     }
54
55     tail = temp;
56     count++;
57 }
58
59 // Insert elements from front
60 template<typename T>
61 void linkedlist<T>::push_front(T d)

```

```

62 {
63     node* temp = new node(d, head); // new node linked to head.
64     if(this->empty())
65     {
66         head = temp;
67         tail = temp;
68     }
69     else
70     {
71         head = temp;
72     }
73     count++;
74 }
75
76 // Delete elements from front
77 template<typename T>
78 void linkedlist<T>::pop_front(void)
79 {
80     if(head==NULL)
81         cout << "Underflow\n";
82     //If there is only one node, then set head and tail to NULL
83     if(this->size()==1)
84     {
85         head=NULL;
86         tail=NULL;
87         count--;
88     }
89     else
90     {
91         node* oldhead = head;
92         delete oldhead;
93         head = head->next;
94         count--;
95     }
96 }
97 }
98
99 // Delete elements from behind
100 template<typename T>
101 void linkedlist<T>::pop_back(void)
102 {
103     if(tail==NULL)
104         cout << "Underflow\n";
105
106     //If there is only one node, then set head and tail to NULL
107     if(this->size()==1)
108     {
109         head=NULL;
110         tail=NULL;
111         count--;
112     }
113     else
114     {
115         node* itr = head;
116         //find the node prior to tail node
117         while(itr->next!=tail)
118         {
119             itr = itr->next;
120         }
121         node* oldtail = itr->next;
122         delete oldtail;
123         itr->next = NULL;
124         tail = itr;

```

```

125         count--;
126     }
127
128 }
129
130 // Display Function
131 template <typename T>
132 void linkedlist<T>::display(void)
133 {
134     node* current = head;
135     if(current!=NULL)
136     {
137         cout << this << ": ";
138         cout << "[ ";
139     }
140     if(current!=NULL)
141     {
142         while(current->next!=NULL)
143         {
144             cout << current->data << " ]->[ ";
145             current = current->next;
146         }
147         cout << current->data;
148     }
149     else
150     {
151         cout << "Underflow!\n";
152         return;
153     }
154
155     cout << " ]->NULL\n";
156 }
157
158 // Copy Constructor
159 template <typename T>
160 linkedlist<T>::linkedlist(const linkedlist<T>& oldlist): head(NULL), tail(NULL),
count(0)
161 {
162     node* current = oldlist.head;
163     while(current!=NULL)
164     {
165         this->push_back(current->data);
166         current = current->next;
167     }
168 }
169
170 // Destructor
171 template <typename T>
172 linkedlist<T>::~~linkedlist(void)
173 {
174
175     while(!this->empty())
176     {
177         this->pop_front();
178     }
179 }
180
181 template <typename T>
182 linkedlist<T> linkedlist<T>::merge( linkedlist<T> &l, linkedlist<T> &f)
183 {
184     node *pl = l.head;
185     node *pf = f.head;
186     node *temp = NULL;

```

```

187     if(!pl) return f;
188     if(!pf) return l;
189     //cout << pl->data << " " << pf->data << "\n";
190     temp = (pl->data > pf->data ? pf: pl);
191     if(temp==pl) pl=pl->next;
192     if(temp==pf) pf=pf->next;
193
194     //cout << pf->data << "\n";
195     // if(pf==NULL) cout << "IN\n";
196     // else cout << "out\n";
197     // cout << temp->data << "\n";
198     while(pl || pf)
199     {
200
201         if(pl!=NULL && pf!=NULL && pf->data > pl->data)
202         {
203             temp->next = pl;
204             temp = pl;
205             pl = pl->next;
206         }
207         if(pl!=NULL && pf!=NULL && pf->data < pl->data)
208         {
209             temp->next = pf;
210             temp = pf;
211             pf = pf->next;
212         }
213         if(!pf)
214         {
215             while(pl)
216             {
217                 temp->next = pl;
218                 temp = pl;
219                 pl = pl->next;
220             }
221         }
222         if(!pl)
223         {
224             while(pf)
225             {
226                 temp->next = pf;
227                 temp = pf;
228                 pf = pf->next;
229             }
230         }
231     }
232     return ((l.head->data) > (f.head->data) ? f : l);
233 }
234
235
236
237 int main()
238 {
239     linkedlist<float> l; // linked-list 1
240     linkedlist<float> f; // linked-list 2
241
242     //push_back in linkedlist 1
243     l.push_back(2);
244     l.push_back(5);
245     l.push_back(7);
246     l.push_back(9);
247     l.push_back(13);
248
249     //push_back in linkedlist 2

```

```
250     f.push_back(1);
251     f.push_back(6);
252     f.push_back(8);
253     f.push_back(10);
254     f.push_back(11);
255
256     (l.merge(l,f)).display();
257     return 0;
258 }
259
```