

```

1 #ifndef _LL_PRO_H_
2 #define _LL_PRO_H_
3 using namespace std;
4
5 template<typename T>
6 class linkedlist
7 {
8 private:
9     class node
10    {
11        friend class linkedlist;
12    private:
13        T data;
14        node *next;
15        // default constructor of class node
16        node(T d, node*n=NULL): data(d), next(n) {}
17    };
18    node *head, *tail;
19 public:
20    int count;
21    //default constructor of claa linkedlist
22    linkedlist(void): head(NULL), tail(NULL), count(0) {}
23    ~linkedlist(void);
24    linkedlist(const linkedlist<T> &oldlist);
25    void push_front(T);
26    void push_back(T);
27    void push_at(T, int pos);
28    T pop_front(void);
29    T pop_back(void);
30    int getSize(void) {return count;};
31    void reverse(void);
32    void display(void);
33    int empty(void) {return count==0;};
34    void merge(linkedlist<T> &l2);
35    bool check_loop();
36    void create_loop();
37    void break_loop();
38    void remove_dupes();
39 };
40
41 template <typename T>
42 linkedlist<T>::~~linkedlist(void)
43 {
44     // if(!this->empty())
45     // cout << "linkedlist : " << this << " is destroyed\n";
46     while(!this->empty())
47     {
48         this->pop_front();
49     }
50 }
51
52
53 //copy constructor
54 //initializes the new linkedlist , head tail and count to NULL & 0
55 template<typename T>
56 linkedlist<T>::linkedlist(const linkedlist<T>& oldlist): head(NULL), tail(NULL), count(0)
57 {
58     node *current = oldlist.head;
59     while(current!=NULL)
60     {
61         this->push_back(current->data);
62         current = current->next;
63     }
64 }
65
66 template<typename T>
67 void linkedlist<T>::push_back(T d)
68 {

```

```

69  node *temp = new node(d, NULL);
70  if(this->empty())
71  {
72      head = temp;
73  }
74  else
75  {
76      tail->next = temp;
77  }
78  tail = temp;
79  count++;
80 }
81
82 template<typename T>
83 void linkedlist<T>::push_front(T d)
84 {
85     node *temp = new node(d, head);
86     if(this->empty())
87     {
88         head = temp;
89         tail = temp;
90     }
91     else
92     {
93         head = temp;
94     }
95     count++;
96 }
97
98 template<typename T>
99 void linkedlist<T>::push_at(T d, int pos)
100 {
101     pos--;
102     if(this->empty() && pos > this->getSize())
103     {
104         cout << "\nInvalid Index\n";
105         exit(0);
106     }
107     node*temp = this->head;
108     if(pos==0)
109     {
110         this->push_front(d);
111     }
112     else
113     {
114         while(--pos > 0)
115         {
116             temp = temp->next;
117         }
118         node *forward = temp->next;
119         node *newdata = new node(d, forward);
120         temp->next = newdata;
121     }
122 }
123
124 template<typename T>
125 T linkedlist<T>::pop_back(void)
126 {
127     if(tail==NULL)
128     {
129         // cout << "\nUnderflow\n";
130         exit(0);
131     }
132     T data;
133     if(this->getSize()==1)
134     {
135         data = head->data;
136         delete this->head;
137     }

```

```

138     else
139     {
140         node *temp = head;
141         while(temp->next!=tail)
142         {
143             temp = temp->next;
144         }
145         node *oldtail = temp->next;
146         data = oldtail->data;
147         delete oldtail;
148         temp->next = NULL;
149         temp = tail;
150     }
151     count--;
152     return data;
153 }
154
155 template<typename T>
156 T linkedlist<T>::pop_front(void)
157 {
158     if(head==NULL)
159     {
160         // cout << "\nUnderflow\n";
161         exit(0);
162     }
163     T data;
164     if(this->getSize()==1)
165     {
166         data = head->data;
167         delete this->head;
168     }
169     else
170     {
171         node *oldhead = head;
172         data = oldhead->data;
173         head = head->next;
174         delete oldhead;
175     }
176     count--;
177     return data;
178 }
179
180 template<typename T>
181 void linkedlist<T>::reverse(void)
182 {
183     if(!this->empty())
184     {
185         node *current = head, *previous = NULL, *forward = head;
186         tail = head;
187         while(forward!=NULL)
188         {
189             forward = current->next;
190             current->next = previous;
191             previous = current;
192             current = forward;
193         }
194         head = previous;
195     }
196     else
197     {
198         cout << "\nListy Empty or cannot be reversed\n";
199     }
200 }
201
202
203 template<typename T>
204 void linkedlist<T>::display(void)
205 {
206     node* current = head;

```

```

207  if(current!=NULL)
208  {
209      cout << this << ": ";
210      cout << "[ ";
211  }
212  if(current!=NULL)
213  {
214      while(current->next!=NULL)
215      {
216          cout << current->data << " ]->[ ";
217          current = current->next;
218      }
219      cout << current->data;
220  }
221  else
222  {
223      cout << "Underflow!\n";
224      return;
225  }
226  cout << " ]->NULL\n";
227
228 }
229
230 template<typename T>
231 void linkedlist<T>::merge(linkedlist<T> &l2)
232 {
233     int s1 = this->getSize(), s2 = l2.getSize();
234     node *t1 = this->head;
235     node *t2 = l2.head;
236     int pos = 1;
237     while(1)
238     {
239         if(t1==NULL)
240         {
241             while(t2!=NULL)
242             {
243                 this->push_back(t2->data);
244                 t2 = t2->next;
245             }
246             break;
247         }
248         if(t2==NULL)
249             break;
250         if(t1->data >= t2->data)
251         {
252             push_at(t2->data,pos);
253             t2 = t2->next;
254         }
255         else
256         {
257             t1 = t1->next;
258         }
259         pos++;
260     }
261     this->count = s2 + s1;
262     l2.~linkedlist<T>();
263 }
264
265 template<typename T>
266 bool linkedlist<T>::check_loop()
267 {
268     node* fast = head, *slow = head;
269     while(fast !=NULL && slow!=NULL)
270     {
271         if(fast->next)
272             fast = fast->next;
273         else
274             return false;
275         fast = fast->next;

```

```

276     slow = slow->next;
277
278     if(fast == slow && fast && slow)
279     {
280         //dedug
281         cout << "Met @ " << fast->data << "\n";
282         return true;
283     }
284 }
285 return false;
286 }
287
288 template<typename T>
289 void linkedlist<T>::create_loop()
290 {
291     if(head==NULL)
292     {
293         cout << "linkedlist too small\n";
294         return;
295     }
296     node *temp = head;
297     while(temp!=tail)
298     {
299         temp = temp->next;
300     }
301     //change loop point here
302     if(head->next->next)
303         temp->next = head->next->next;
304     else if(head->next)
305         temp->next = head->next;
306     else
307         temp->next = head;
308 }
309 }
310
311 template<typename T>
312 void linkedlist<T>::break_loop()
313 {
314     if(this->check_loop())
315     {
316         node* fast = head, *slow = head, *temp = head;
317         while(fast !=NULL && slow!=NULL)
318         {
319             fast = fast->next->next;
320             slow = slow->next;
321
322             if(fast == slow)
323             {
324                 while(slow!=temp)
325                 {
326                     slow = slow->next;
327                     temp = temp->next;
328                 }
329                 cout << "Loop Point @ " << temp->data << "\n";
330                 while(slow->next!=temp)
331                 {
332                     slow = slow->next;
333                 }
334                 slow->next = NULL;
335                 tail = slow;
336                 return;
337             }
338         }
339     }
340     else
341     {
342         cout << "No loop detected\n";
343         return;
344     }

```

```
345 }
346
347 template<typename T>
348 void linkedlist<T>::remove_dupes()
349 {
350     if(this->head==NULL)
351     {
352         cout << "Nothing to remove\n";
353         return;
354     }
355
356     node* temp = this->head;
357     node* previous = head;
358     T value = temp->data;
359
360     while(temp->next!=NULL )
361     {
362         temp = temp->next;
363         if(temp->data != value)
364         {
365             value = temp->data;
366             previous->next = temp;
367             previous = temp;
368         }
369     }
370
371     if(temp->data!=value)
372     {
373         previous->next = temp;
374         temp->next = NULL;
375     }
376     else
377         previous->next = NULL;
378 }
379
380 #endif
```