```
#ifndef _LL_PRO_H_
#define _LL_PRO_H_
using namespace std;

template<typename T>
class linkedlist
{
private:
  class node
  {
    friend class linkedlist;
  private:
    T data;
    node *next;
    // default constructor of class node
    node(T d, node*n=NULL): data(d), next(n) {}
  };
  node *head, *tail;
public:
  int count;
  //default constructor of claa linkedlist
  linkedlist(void): head(NULL), tail(NULL), count(0) {}
  ~linkedlist(void);
  linkedlist(const linkedlist<T> &oldlist);
  void push_front(T);
  void push_back(T);
  void push_at(T, int pos);
  T pop_front(void);
  T pop_back(void);
  int getSize(void) {return count;};
  void reverse(void);
  void display(void);
  int empty(void) {return count==0;};
  void merge(linkedlist<T> &l2);
  bool check_loop();
  void create_loop();
  void break_loop();
  void remove_dupes();
};

template <typename T>
linkedlist<T>::~linkedlist(void)
{
  // if(!this->empty())
  //   cout << "linkedlist : " << this <<  " is destroyed\n";
  while(!this->empty())
  {
    this->pop_front();
  }
}


//copy constructor
//initializes the new linkedlist , head tail and count to NULL & 0
template<typename T>
linkedlist<T>::linkedlist(const  linkedlist<T>& oldlist): head(NULL), tail(NULL), count(0)
{
  node *current = oldlist.head;
  while(current!=NULL)
  {
    this->push_back(current->data);
    current = current->next;
  }
}

template<typename T>
void linkedlist<T>::push_back(T d)
{
```

```
  node *temp = new node(d,NULL);
  if(this->empty())
  {
    head = temp;
  }
  else
  {
    tail->next = temp;
  }
  tail = temp;
  count++;
}

template<typename T>
void linkedlist<T>::push_front(T d)
{
  node *temp = new node(d, head);
  if(this->empty())
  {
    head = temp;
    tail = temp;
  }
  else
  {
    head = temp;
  }
  count++;
}

template<typename T>
void linkedlist<T>::push_at(T d, int pos)
{
  pos--;
  if(this->empty() && pos > this->getSize())
    {
      cout << "\nInvalid Index\n";
      exit(0);
    }
  node*temp = this->head;
  if(pos==0)
  {
    this->push_front(d);
  }
  else
  {
    while(--pos > 0)
    {
      temp = temp->next;
    }
    node *forward = temp->next;
    node *newdata = new node(d, forward);
    temp->next = newdata;
  }
}

template<typename T>
T linkedlist<T>::pop_back(void)
{
  if(tail==NULL)
  {
    // cout << "\nUnderflow\n";
    exit(0);
  }
  T data;
  if(this->getSize()==1)
  {
    data = head->data;
    delete this->head;
  }
```

```
  else
  {
    node *temp = head;
    while(temp->next!=tail)
    {
      temp = temp->next;
    }
    node *oldtail = temp->next;
    data = oldtail->data;
    delete oldtail;
    temp->next = NULL;
    temp = tail;
  }
  count--;
  return data;
}

template<typename T>
T linkedlist<T>::pop_front(void)
{
  if(head==NULL)
  {
    // cout << "\nUnderflow\n";
    exit(0);
  }
  T data;
  if(this->getSize()==1)
  {
    data = head->data;
    delete this->head;
  }
  else
  {
    node *oldhead = head;
    data = oldhead->data;
    head = head->next;
    delete oldhead;
  }
  count--;
  return data;
}

template<typename T>
void linkedlist<T>::reverse(void)
{
  if(!this->empty())
  {
    node *current = head, *previous = NULL, *forward = head;
    tail = head;
    while(forward!=NULL)
    {
      forward = current->next;
      current->next = previous;
      previous = current;
      current = forward;
    }
    head = previous;
  }
  else
  {
    cout << "\nListy Empty or cannot be reversed\n";
  }
}


template<typename T>
void linkedlist<T>::display(void)
{
  node* current = head;
```

```
  if(current!=NULL)
  {
    cout << this << ": ";
    cout << "[ ";
  }
  if(current!=NULL)
  {
    while(current->next!=NULL)
    {
      cout << current->data << " ]->[ ";
      current = current->next;
    }
    cout << current->data;
  }
  else
  {
    cout << "Underflow!\n";
    return;
  }
  cout << " ]->NULL\n";

}

template<typename T>
void linkedlist<T>::merge(linkedlist<T> &l2)
{
    int s1 = this->getSize(), s2 = l2.getSize();
    node *t1 = this->head;
    node *t2 = l2.head;
    int pos = 1;
    while(1)
    {
        if(t1==NULL)
        {
          while(t2!=NULL)
          {
            this->push_back(t2->data);
            t2 = t2->next;
          }
          break;
        }
        if(t2==NULL)
          break;
        if(t1->data >= t2->data)
        {
            push_at(t2->data,pos);
            t2 = t2->next;
        }
        else
        {
          t1 = t1->next;
        }
        pos++;
    }
    this->count = s2 + s1;
    l2.~linkedlist<T>();
}

template<typename T>
bool linkedlist<T>::check_loop()
{
  node* fast = head, *slow = head;
  while(fast !=NULL && slow!=NULL)
  {
    if(fast->next)
      fast = fast->next;
    else
      return false;
    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
```