p | arrlist.h | arrlist.cpp | list2.cpp | list2.h

```
1  #include "list.h"
2  template<class T>
3  class UnsortedList: protected SinglyLinkedList<T>{
4  public:
5      UnsortedList(){}
6      ~UnsortedList(){}
7
8      //make the following members of ancestor classes
9      //publicly accessible which are now protected members
10     //of UnsortedList class (because of protected inheritance)
11     using SinglyLinkedList<T>::isEmpty;
12     using SinglyLinkedList<T>::searchItem;
13     using SinglyLinkedList<T>::reset;
14     using SinglyLinkedList<T>::nextItem;
15     using SinglyLinkedList<T>::hasNext;
16
17     //functions specific to UnsortedList class
18     virtual void insertItem(T value);
19     virtual void deleteItem(T value);
20 };
21
22 /* Linked List based implementation of SortedList */
23 template<class T>
24 class SortedList: public UnsortedList<T>{
25 public:
26     SortedList(){}
27     ~SortedList(){}
28     //functions specific to SortedList class
29     virtual void insertItem(T value);
30 };
```

arrlist.h | arrlist.cpp | **list2.cpp** | list2.h

```cpp
#include "list2.h"
template<class T>
void UnsortedList<T>::insertItem(T value)//O(1)
{
    this->insertAtStart(value);
}

template<class T>
void UnsortedList<T>::deleteItem(T value)//O(N)
{
    if(isEmpty())throw ListEmpty();

    //if head->data == value
    if(this->head->data==value){
        this->deleteStart();
        return;
    }

    node *pre=NULL, *cur=(this->head);
    while(cur->data != value){
        pre = cur;
        cur = cur->next;
    }

    if(cur != NULL){     //if value is found in t
        pre->next = cur->next;
        delete cur;
        (this->length)--;
    }
}
```

```
31
32  template<class T>
33  void SortedList<T>::insertItem(T value)//O(N)
34  {
35      if(this->isEmpty() || value < (this->head)->data)
36      {
37          this->insertAtStart(value);
38      }
39      else
40      {
41          node *temp=new node;
42          node *pre=NULL, *cur=(this->head);
43          while(cur != NULL && (cur->data) < value)
44          {
45              pre = cur;
46              cur = cur->next;
47          }
48
49          temp->data = value;
50          temp->next = cur;
51          pre->next = temp;
52          (this->length)++;
53      }
54  }
```