

## Part 1 Circular Linked List (50 marks)

The Circular Linked List is a useful data structure for many applications. A circular linked list is a linked list that the last node has its next pointer to the first node in the list, instead of NULL.

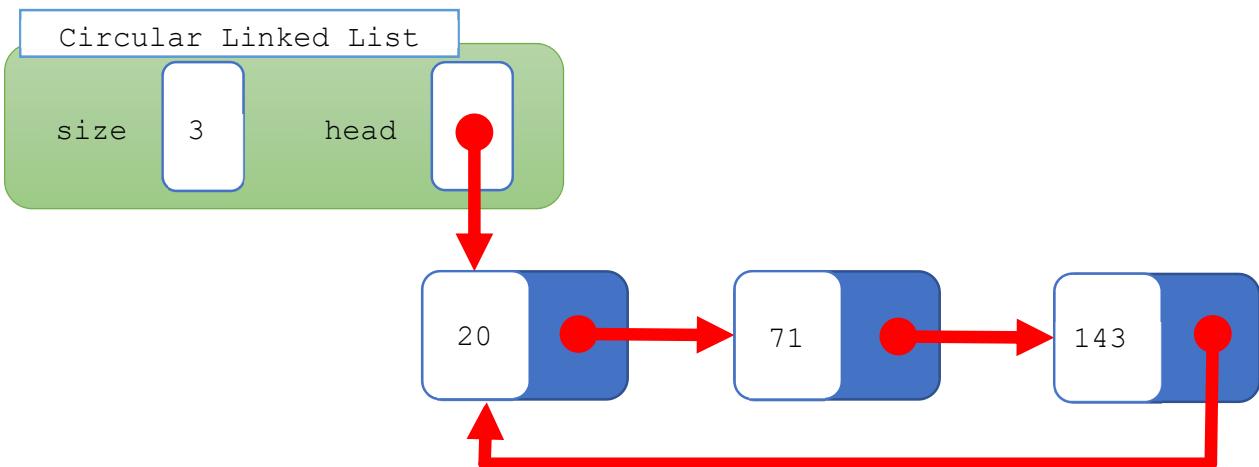


Figure 1: An example of a Circular Linked List

And the head can be “advanced”, namely, move to the next node. For example, we can advance the head of the above like this:

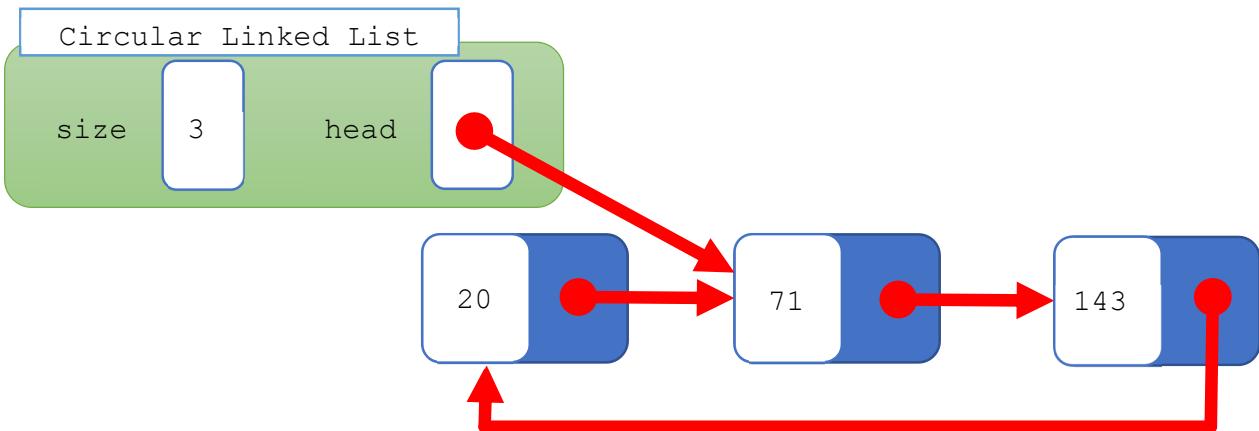


Figure 2: After 1 “Advance Head” from the list in Figure 1

And we can keep advancing forever.

One of the important applications is scheduling tasks in a multitasking operating system. All the running applications are kept in a circular linked list and the OS gives a fixed time slot to all for running. The Operating System keeps on iterating over the linked list until all the applications are completed. A new task will be added to the linked list when the user started one. And a task will be removed from the list when the task is done.

### Project/File Settings

You will receive the following files in your project file:

- `circularIntLinkedList.{h, cpp}` and `main.cpp`

## Your Tasks

Before you started, please bear in mind that **you will only get full marks if all of your functions have a time complexity of O(1)** except `print()` and `exist()`. (Hint: However, the *trick* to achieve O(1) without adding new attribute is not following the implementation in Figures 1 and 2 exactly.) The followings are the requirements for the functions.

Your tasks in this part is to implement the following seven functions. You should submit all of the following 7 functions and 7 only to coursemology :

- `void CircularList::insertHead(int n)`
- `void CircularList::advanceHead()`
- `void CircularList::removeHead()`
- `void CircularList::print()`
- `bool CircularList::exist(int n)`
- `int CircularList::headItem()`
- `CircularList::~CircularList()`

You should not modify `circularIntLinkedList.h`. Namely, you cannot add/remove/change the definitions, attributes or member functions of the given classes. However, you can modify `main.cpp` for your own testing. And you should not use any extra global variables. Make it short, you should ONLY submit the above seven member functions only with their bodies implemented.

### `insertHead(int n)`

Insert the integer `n` into the head/front of the list. Here is the sample output for the test function given on the right.

```
After adding 123
123
After adding 11
11 123
After adding 9
9 11 123
After adding 1
1 9 11 123
After adding 20
20 1 9 11 123
```

### `print()`

The function should print the list in one line and end with a new line. There is no leading space before the line but there is a single space after each integer printed, including the last integer. E.g. the second printout above is "11 123 ". And finally, print a new line at the end, e.g. `endl`.

### `removeHead();`

It will remove the first item currently at the head. If the list is empty, the function will just do nothing.

```
After removing the head, the current list is: 1 9 11 123
After removing the head, the current list is: 9 11 123
After removing the head, the current list is: 11 123
After removing the head, the current list is: 123
After removing the head, the current list is:
```

### `exist(int n):`

The function will return true if the integer `n` is in the list, and return false otherwise.

### `headItem();`

This function will return the integer at the head of the list currently. You can assume that we will call this function only if the list is not empty.

## **advanceHead()**

The function will move make the item next to the current head to be the head of the list. The previous head will be moved to the end of the list. Please see Figures 1 and 2 for an example. The function should work even if the list is empty.

```
The original list:20 1 9 11 123
After 1 "advance"
The current list is: 1 9 11 123 20
After 2 "advance"
The current list is: 9 11 123 20 1
After 3 "advance"
The current list is: 11 123 20 1 9
After 4 "advance"
The current list is: 123 20 1 9 11
After 5 "advance"
The current list is: 20 1 9 11 123
```

## **~CircularList()**

The destructor of the class should remove all the items of the list before the list is destroyed.

## **Submission**

Please paste the above seven functions into “Question 1: Circular Linked List (Part 1)” in coursemology.

## **Sample Output:**

If you uncomment all the functions in main() , you should have following outputs

```
insertHeadtest()
After adding 123
123
After adding 11
11 123
After adding 9
9 11 123
After adding 1
1 9 11 123
After adding 20
20 1 9 11 123
```

```
insertHeadtest()

existTest()
The list is: 20 1 9 11 123
Does 9 exist in the list? Yes

Does 11 exist in the list? Yes

Does 99 exist in the list? No
```

```
advanceHeadtest()
The original list:20 1 9 11 123
After 1 "advance"
The current list is: 1 9 11 123 20
After 2 "advance"
The current list is: 9 11 123 20 1
After 3 "advance"
The current list is: 11 123 20 1 9
After 4 "advance"
The current list is: 123 20 1 9 11
After 5 "advance"
The current list is: 20 1 9 11 123
```

```
removeHeadtest()
After removing the head, the current list is: 1 9 11 123
Does 9 exist in the list? Yes

After removing the head, the current list is: 9 11 123
Does 9 exist in the list? Yes

After removing the head, the current list is: 11 123
Does 9 exist in the list? No

After removing the head, the current list is:
Does 9 exist in the list? No

removeAndAdvanceHeadTest()
The current list is: 20 1 9 11 123
Does 9 exist in the list? Yes

Now we remove the head and advance it once
The current list is: 9 11 123 1
Does 9 exist in the list? Yes

Now we remove the head and advance it once
The current list is: 123 1 11
Does 9 exist in the list? No

Now we remove the head and advance it once
The current list is: 11 1
Does 9 exist in the list? No

Timing Test
Time taken for adding 1 items : 0s
Time taken for advancing three quarters of the list: 0s (Head item = 0)

Time taken for adding 10 items : 0s
Time taken for advancing three quarters of the list: 0s (Head item = 2)

Time taken for adding 100 items : 0s
Time taken for advancing three quarters of the list: 0s (Head item = 24)

Time taken for adding 1000 items : 0s
Time taken for advancing three quarters of the list: 0s (Head item = 249)

Time taken for adding 10000 items : 0.002s
Time taken for advancing three quarters of the list: 0s (Head item = 2499)

Time taken for adding 100000 items : 0.019s
Time taken for advancing three quarters of the list: 0.002s (Head item = 24999)

Time taken for adding 1000000 items : 0.19s
Time taken for advancing three quarters of the list: 0.02s (Head item = 249999)
```

Note that the timing for the final part may be different. However, **everything should be able to finish within 1s.**