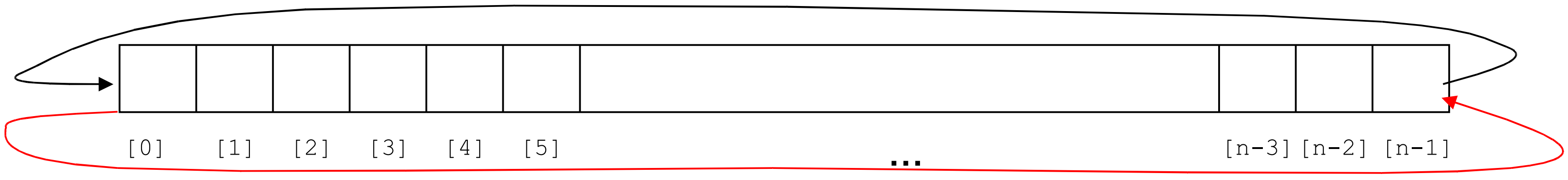
The circular array

- Most of our problems with the use of an array (or a vector, which uses the array internally) to add and delete from a list are on two grounds
  1. There is a restriction that the array might be full
  2. When we add or delete in the array, there is a performance overhead as the rest of the data is moved up or down by 1 index position.

- There are many data-structures and algorithms to attempt a fix of the second problem.
  - One is the "circular array"
  - The circular array enables data to be added or subtracted efficiently at *both ends* of the array.
  - It will still not help if data is added or removed in the middle of the array.

The circular array: demonstration

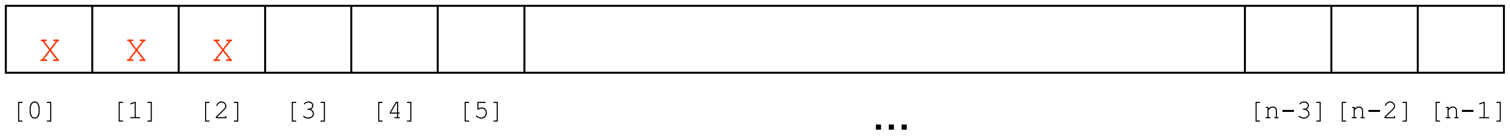
- In a circular array, the array indices ‘wrap around’ so that in an array of size n, the next index position *after* [n – 1] is the index 0



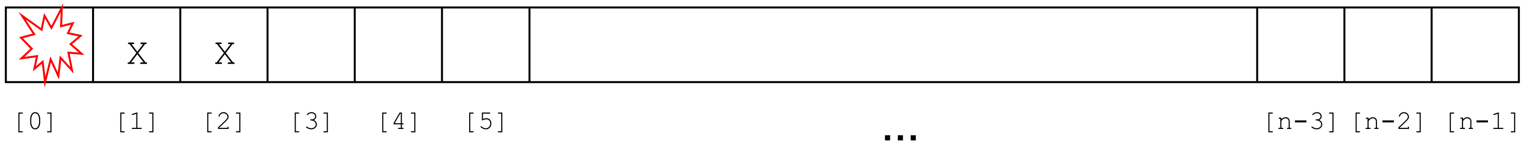
- And the index position *before* [0] is the index [n-1]

- As items are added or removed at the ‘back’ of the array and added or removed at the ‘front’ of the array, it is necessary to maintain information about the first index and last index of the valid array items.

- Add 3 items to the array: front is at `[0]`, back is at `[2]`

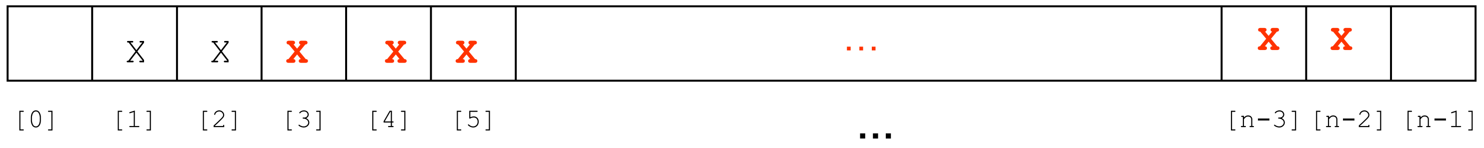


- Remove the front item: front is at `[1]`, back is at `[2]`

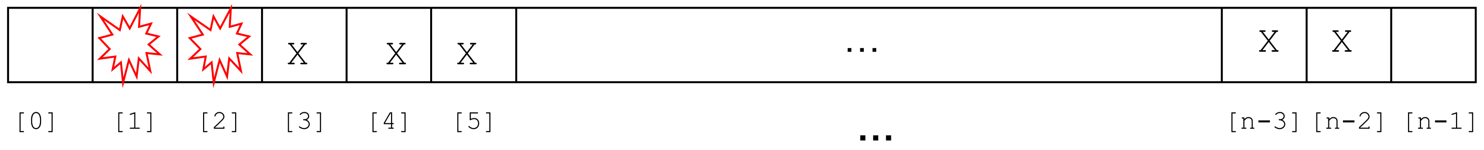


The circular array: demonstration cont

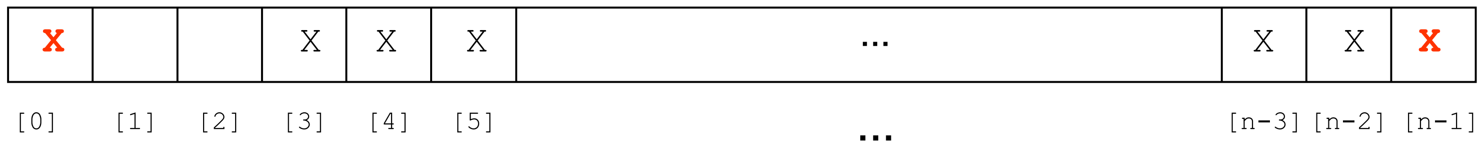
- Add a lot more items at the end: front is still `[1]`, lets say back is at `[n-2]`



- Remove 2 more items from the front: front is at `[3]`, back is still at `[n-2]`

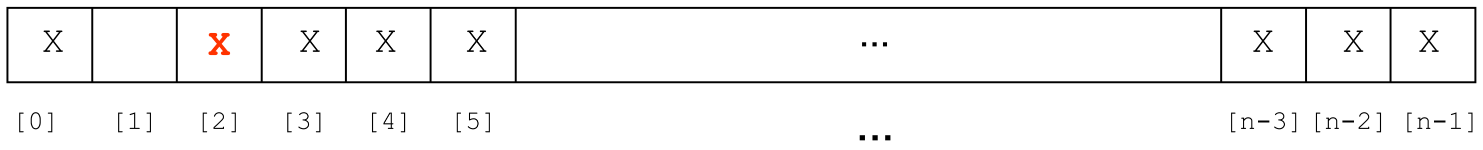


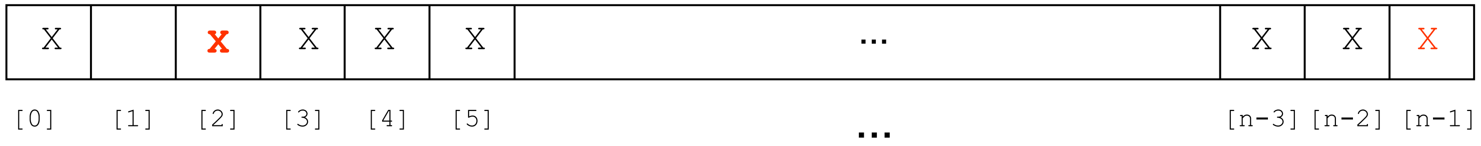
- Now add 2 more items to the end : front is at `[3]`, back is at `[0]`



- Notice that we have now 'wrapped around' the array

- Add 1 item at the front : front is at `[2]`, back is at `[0]`





- We must check before we add more items that we are not overwriting any already there: for example, we could now only add 1 more items to this list before overwriting
  - It is easy to check if the list is full: if the back is just one behind the front.

- if front and back are the same index it has exactly 1 element.

- What about calculating the number of items in the array?

```
if (back >= front)
  size = back – front + 1;
else
     size = arysize – front + back + 1
```

exercise

- Write a `deque` class to hold a list of integers which is implemented internally with a circular array
  - The size of the array can be passed in the constructor, or you can decide on a default value.
  - The class will maintain data members which hold the index position of the head and tail of the list
  - the class should have member functions
    - **`bool isEmpty();`**
    - **`bool isFull();`**
    - **`bool insertFront(int)`**
    - **`bool removeFront(int&)`**
    - **`bool insertBack(int)`**
    - **`bool removeBack(int&)`**
- Write a program which uses your class, adds and removes sufficient items to ‘wrap around’ the array, and finally prints all items in the array by removing them one at a time from the front