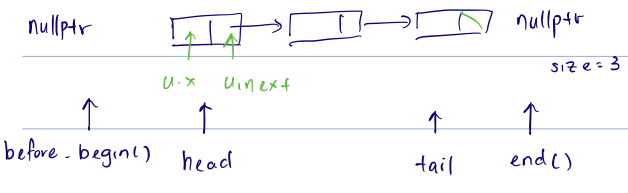
- pointer based data structure
  - disadvantage: no more get(i) and set(i,x). in constant time
  - advantage: more dynamic (we can insert and delete node u in constant time).
  - keeps track of head, tail, size.

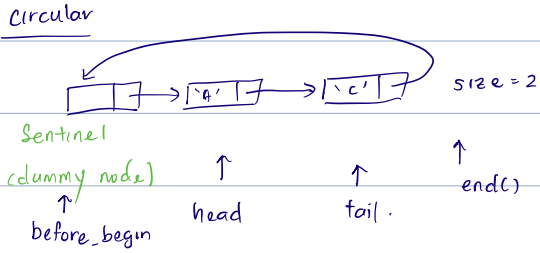
basic operations.

- add(x) } queue operation O(1)
- remove() }
- pop() } stack operation O(1)
- push(y) }

null terminated list.



circular



before_begin() = nullptr	= Sentinel.
begin() = head	= head.
end() = nullptr	= Sentinel.
individual node.	
	
next = nullptr	= this.
Private member (size=0)	
head = nullptr	= Sentinel.next
tail = nullptr	= Sentinel
size: 0	= 0
insert_after(pos, data)	

(create new node)

```
Node* newNode = new Node(data).
```

```
Node * newNode = new Node(data);
```

(keep temporary reference to node at given pos)

```
Node * currentNode = position._nodeptr.;
```

```
Node * currentNode = position._nodeptr;
```

(if list empty, set head, tail to new node)

```
(if size() == 0) head = tail = newNode;
```

(assign new node's next to curr->next)

```
newNode -> next = currentNode -> next
```

(assign current node's next to newnode)

```
current -> next = newNode.
```

(special case: if insert at beginning)
(if currentNode == before_begin()) {
(assign newNode's next to head)
newNode → next = head;
(reassign head to point to new node)
head = newNode; }

(if inserting at back of list)
(if currentNode == tail) {
(old tail's next assign to new node
tail → next = newNode;
(tail reassigned to new node.
tail = newNode; }.

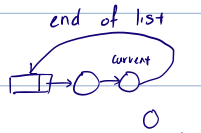
(insert somewhere in the list.)
else {
(reassign newNode's next to currentNode's next)
newNode → next = currentNode → next
(reassign currentNode's next to new node)
currentNode → next = newNode;

(update tail if necessary)
if (currentNode → next == end()) tail = newNode;

*Notice that the operations are similar no matter what position we are in list
beginning / middle of list.

newNode → next = current Node → next

currentNode → Next = newNode.



newNode → next = current → next
current → next = newnode.
tail = newnode.
we just need to update tail if necessary.

erase_after (pos)

Node* toBeRemoved = nullptr.

if (isEmpty()) {throw error}
if (pos = tail), nothing to do
if (pos == tail) return;
if removing first item in list
if (pos == before_begin()) {
mark head for deletion
toBeRemoved = head;
reassign head
head = head → next;
if removing somewhere in list
else {
mark position → next for deletion
toBeRemoved = Position → next;
reassign position → next to following node
Position → next = Position → next → next;
}

size -= 1
delete toBeRemoved.

if (isEmpty()) {throw error}

if (pos == tail) return;
mark next node for deletion
Node* toBeRemoved = Position → next;
current node to point to following node
Position → next = position → next → next;
Check if tail needs to be reassigned
if (toBeRemoved == tail) {
tail = Position}

size -= 1;
delete toBeRemoved;