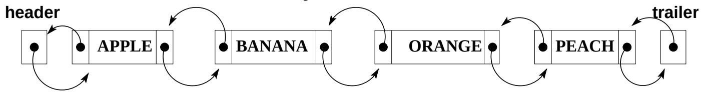
Doubly Linked Lists

A ***doubly linked list*** contains nodes storing two pointers: a ***next*** link which points to the next node in the list (similar to the singly linked list), and a ***previous*** link, which points to the previous node in the list. In this way, a variety of operations (including insertion and removal at both ends of the list) will run in $O(1)$ time. In our implementation, we will add two sentinel nodes (called ***header*** and ***trailer*** nodes) at the beginning and end of the list. They do not store any element.

Doubly Linked List



A queue-like data structure that supports insertion and deletion at both the front and the rear of the queue is called a ***double-ended queue***, or ***deque***.

DoublyListNode Class

In C++, every node is allocated in the heap (free memory) using `new` and deallocated with `delete`. An empty doubly linked list consists of two sentinel nodes pointing to each other. All operations on doubly linked lists must also work for an empty list.

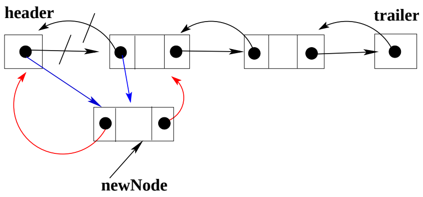
```
class DoublyLinkedList; // class declaration
class DListNode {
private: int obj;
   DListNode *prev, *next;
   friend class DoublyLinkedList;
public:
   DListNode(int e=0, DListNode *p = NULL, DListNode *n = NULL
      : obj(e), prev(p), next(n) {}
   int getElem() const { return obj; }
   DListNode * getNext() const { return next; }
   DListNode * getPrev() const { return prev; }
};
```

DoublyLinkedList Class (cont.)

```
class DoublyLinkedList {
protected:  DListNode header, trailer;
public:
   DoublyLinkedList() : header(0), trailer(0)
       { header.next = &trailer; trailer.prev = &header; }
   ~DoublyLinkedList();
   DListNode *getFirst() const { return header.next; }
   DListNode *getAfterLast() { return &trailer; }
   bool isEmpty() const { return header.next == &trailer; }
   int first() const throw(EmptyDLinkedListException);
   int last() const trow(EmptyDLinkedListException);
   void insertFirst(int newobj);
   int removeFirst() throw(EmptyDLinkedListException);
   void insertLast(int newobj);
   int removeLast() throw(EmptyDLinkedListException);
};
```

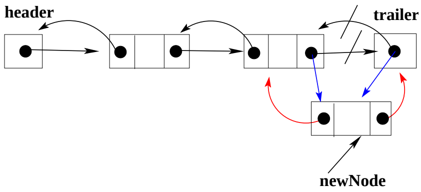
The insertFirst() Function

```
void DoublyLinkedList::insertFirst(int newobj)
{  DListNode *newNode = new DListNode(newobj, &header,
                                     header.next);
   header.next->prev = newNode;
   header.next = newNode;
}
```



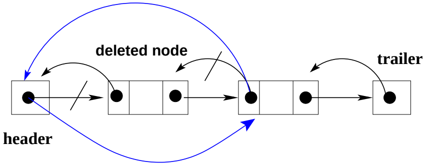
The insertLast() Function

```
void DoublyLinkedList::insertLast(int newobj)
{  DListNode *newNode = new DListNode(newobj, trailer.prev,
                                                     &trailer);
   trailer.prev->next = newNode;
   trailer.prev = newNode;
}
```



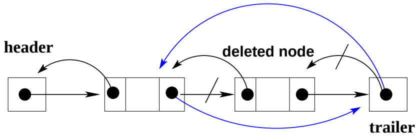
The removeFirst() Function

```
int DoublyLinkedList::removeFirst()
        throw(EmptyDLinkedListException)
{  if (isEmpty())
      throw EmptyDLinkedListException("Empty Doubly Linked List
   DListNode *node = header.next;
   node->next->prev = &header;
   header.next = node->next;
   int obj = node->obj;
   delete node;
   return obj;
}
```



The removeLast() Function

```
int DoublyLinkedList::removeLast()
        throw(EmptyDLinkedListException)
{  if (isEmpty())
      throw EmptyDLinkedListException("Empty Doubly Linked List
   DListNode ⋆node = trailer.prev;
   node->prev->next = &trailer;
   trailer.prev = node->prev;
   int obj = node->obj;
   delete node;
   return obj;
}
```



first() & last() Functions

It returns the object in the first node.

```
int DoublyLinkedList::first() const
      throw(EmptyLinkedListException)
{  if (isEmpty())
       throw EmptyDLinkedListException("Empty Doubly Linked Lis
   return header.next->obj;
}
```

It returns an object in the last node.

```
int DoublyLinkedList::last() const
      throw(EmptyLinkedListException)
{  if (isEmpty())
      throw EmptyDLinkedListException("Empty Doubly Linked List
   return trailer.prev->obj;
}
```

Doubly LinkedList Destructor

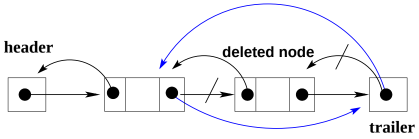
It removes the whole list, and sets `header` and `trailer` to initial values.

```
DoublyLinkedList::~DoublyLinkedList()
{
   DListNode *prev_node, *node = header.next;
   while (node != &trailer) {
      prev_node = node;
      node = node->next;
      delete prev_node;
   }
   header.next = &trailer;
   trailer.prev = &header;
}
```

The Doubly Linked List Count

The length of a doubly linked list is equal to the number of nodes contained in the list.

```
int DoublyLinkedListLength(DoublyLinkedList& dll) {
   DListNode *current = dll.getFirst();
   int count = 0;
   while(current != dll.getAfterLast()) {
      count++;
      current = current->getNext(); //iterate
   }
   return count;
}
```


header
deleted node
trailer