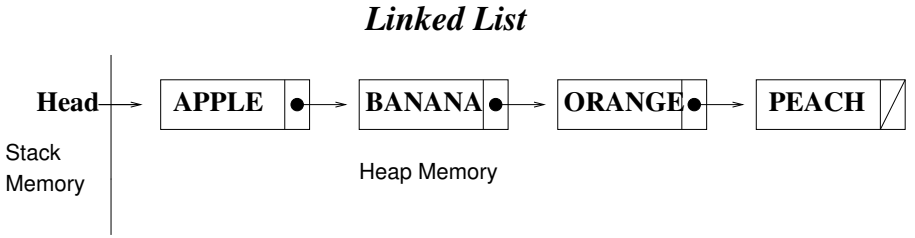
Linked Lists

A *linked list* is a collection of nodes, each containing a data portion and a pointer to another node. The data portion of each node is of the same type. The pointer in a given node points to the location of the next node that logically follows it. The first node of a list is pointed by a separate head pointer.



In C++, every node is allocated in the heap (free memory) using `new` and deallocated with `delete`. An empty linked list consists of no nodes and the value of the head pointer is `NULL`. All operations on linked lists must also work for an empty list.

ListNode Class

The C++ interface corresponding to our stack ADT. It provides to a user useful information about all the public functions of the class `Queue`.

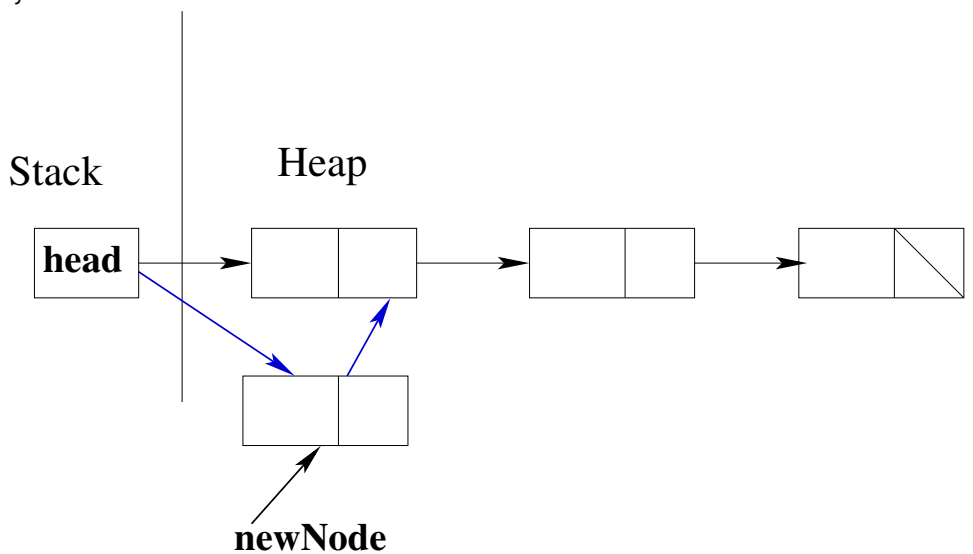
```
class LinkedList; // class declaration
class ListNode {
private: int obj;
   ListNode *next;
   friend class LinkedList;
public:
   ListNode(int e = 0, ListNode *n = NULL) : obj(e), next(n) {}
   int getElem() const { return obj; }
   ListNode * getNext() const { return next; }
};
```

LinkedList Class

```
class LinkedList {
protected:  ListNode *head, *tail;
public:
   LinkedList() : head(NULL), tail(NULL) { }
   ~LinkedList();
   ListNode *getHead() const { return head; }
   bool isEmpty() const { return head == NULL; }
   int first() const throw(EmptyLinkedListException);
   void insertFirst(int newobj);
   int removeFirst() throw(EmptyLinkedListException);
   void insertLast(int newobj);
  };
 ostream& operator<<(ostream& out, const LinkedList &ll);
```

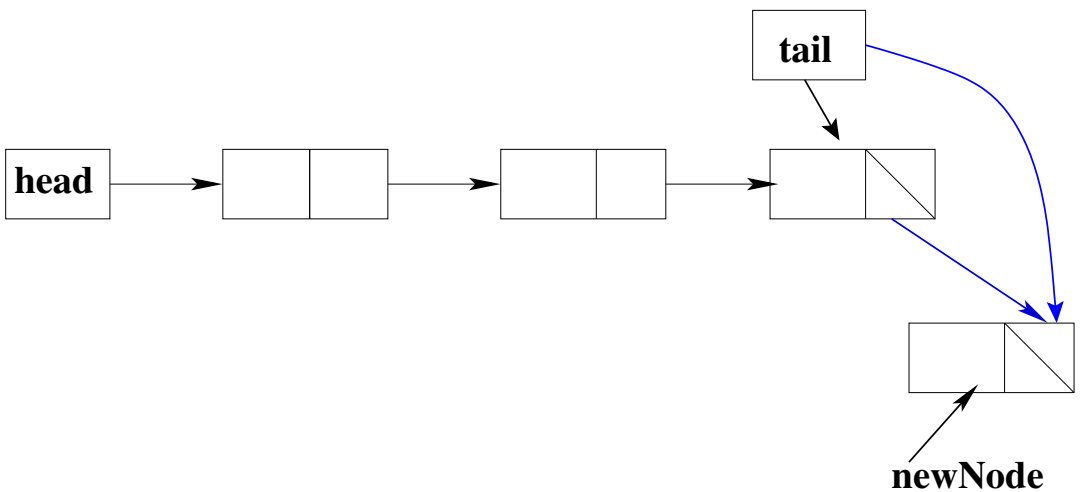
LinkedList Class Implementation

```
void LinkedList::insertFirst(int newobj)
{
   ListNode *newNode = new ListNode(newobj);
   if (head == NULL) tail = newNode;
   newNode->next = head;
   head = newNode;
}
```



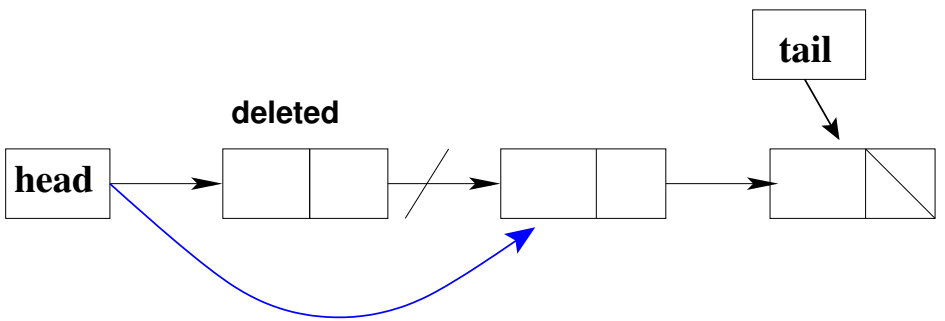
LinkedList Class (cont.)

```
void LinkedList::insertLast(int newobj) {
   ListNode *newNode = new ListNode(newobj);
   if (head == NULL) head = tail = newNode;
   else {
      tail->next = newNode;
      tail = newNode;
   }
}
```



LinkedList Class (cont.)

```
int LinkedList::removeFirst() throw(EmptyLinkedListException) {
   if (head == NULL)
      throw EmptyLinkedListException("Empty Linked List");
   ListNode *node = head;
   head = node->next;
   if (head == NULL) tail = NULL;
   int obj = node->obj;
   delete node;
   return obj;
}
```



Linked List Destructor

It removes the whole list, and sets `head` and `tail` to `NULL`.

```
LinkedList::~LinkedList() {
   ListNode *node;
   while (head != NULL) {
      node = head;
      head = head->next;
      delete node;
   }
   tail = NULL;
}
```

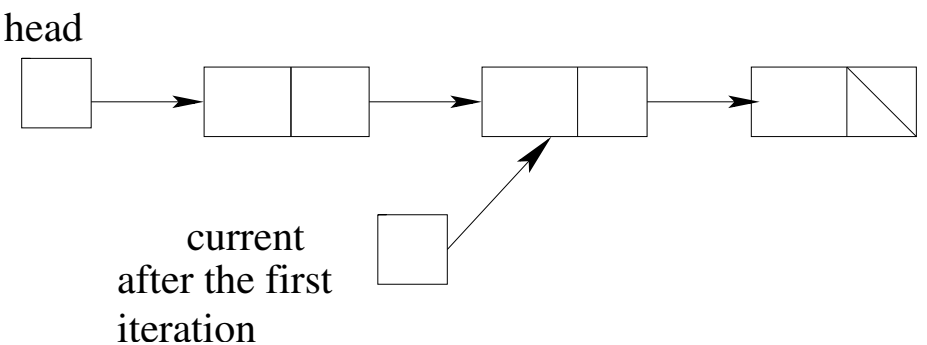
It returns the object in the first node.

```
int LinkedList::first() const throw(EmptyLinkedListException) {
   if (head == NULL)
      throw EmptyLinkedListException("Empty Linked List");
   return head->obj;
}
```

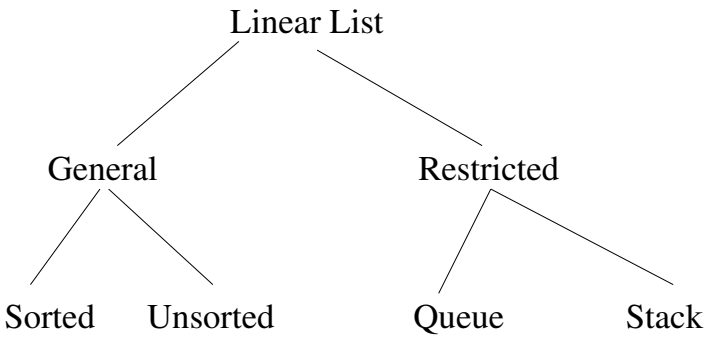
The lenght of Linked List

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

```
int LinkedListLength(LinkedList& ll) {
   ListNode *current = ll.getHead();
   int count = 0;
   while(current != NULL) {
      count++;
      current = current->getNext(); //iterate
   }
   return count;
}
```



Linked List Types



List ADT

Insertion -- add a new element

Deletion -- search and remove an elem

Retrieval -- get access to an element

Traversal -- all data are retrieved