Lesson 4

Data Structures

What is a data structure?

A data structure is a particular way of organizing data in a computer.

A data structure that we have already encountered is the array. An array stores data in a contiguous piece of memory. This means that the data is all stored next to each other in sequence in memory.

We are now going to look at another data structure, a linked list.

Linked List

A big problem with arrays are that they are of a fixed size meaning that you need to know how many items you will be storing in them when you create it.

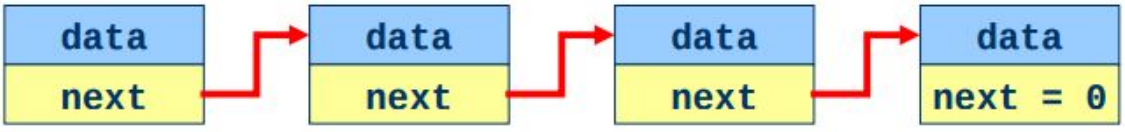
A linked list grows as data is added to it. In a linked list each item is “packaged” into a node.

```
struct Node {
  int data;
  Node* next;
};
```

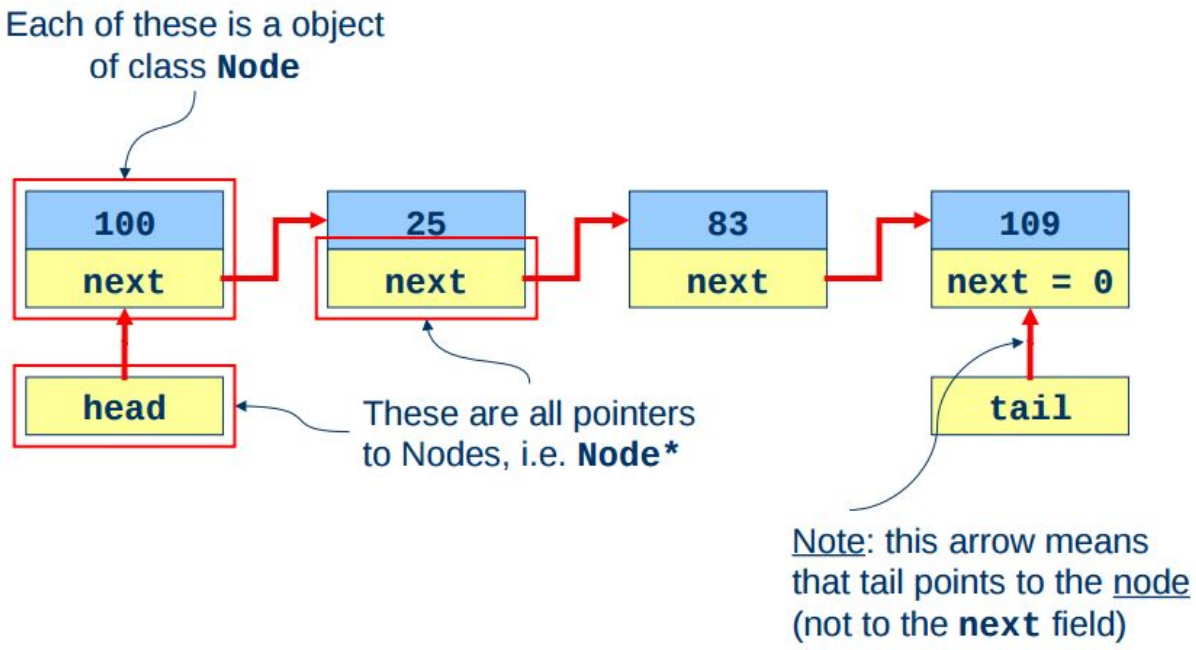
data is the payload, or the information you want to store. This can be any data type and needn’t be limited to one field.

next is a pointer to the next node in the list.
If next = 0 in the diagrams (for this exercise we will assume 0 = NULL) it means that no node follows it in the list. Meaning that we are at the end of the list.

Nodes are then linked into a list or chain.

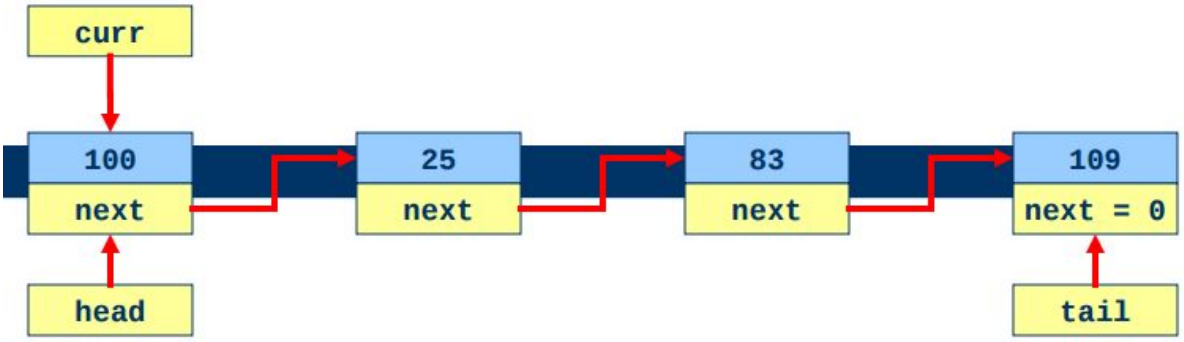


To keep track of our linked list we create a pointer to the first node, or a head pointer.

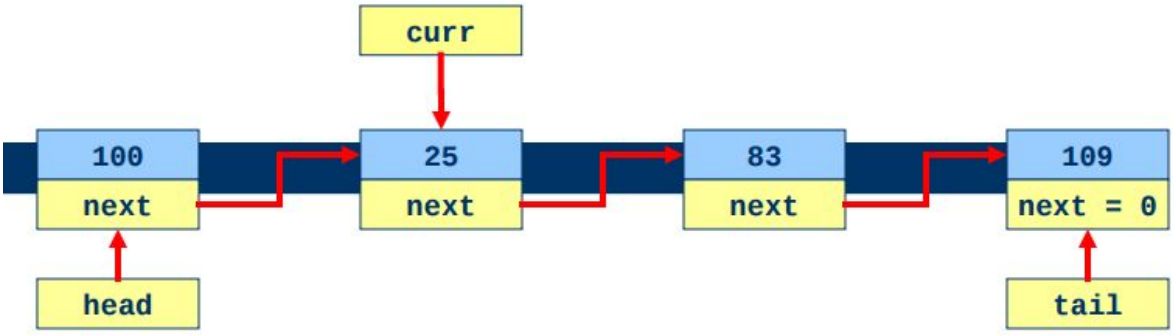


Iterating through a linked list

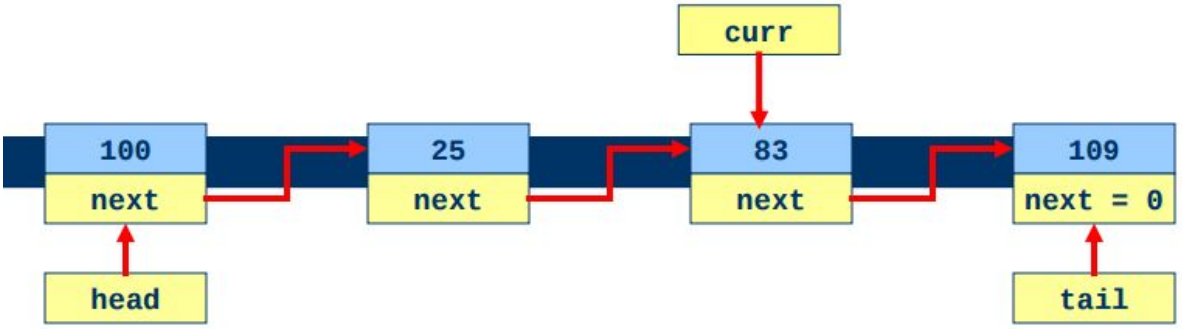
As with an array we need some method of accessing the elements of the linked list. To do this we need to create a curr pointer. We set this pointer to point to the same place as the head pointer.

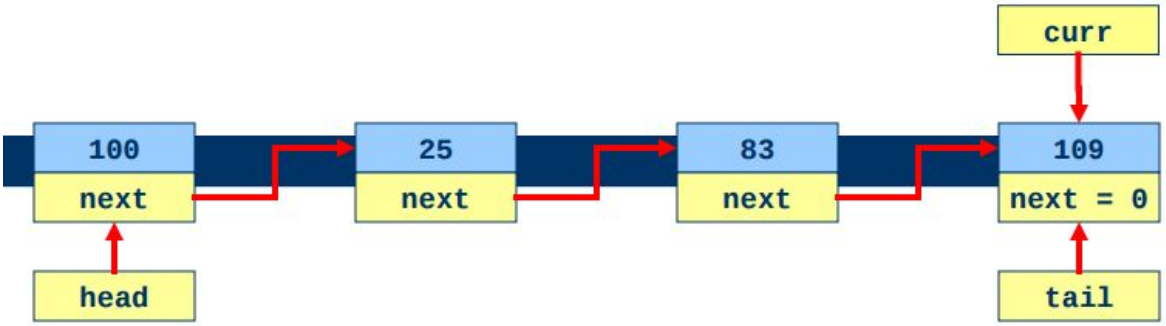


If this node's next pointer does not equal NULL we can set the curr pointer to node next pointer. So that curr is now pointing at the second node.

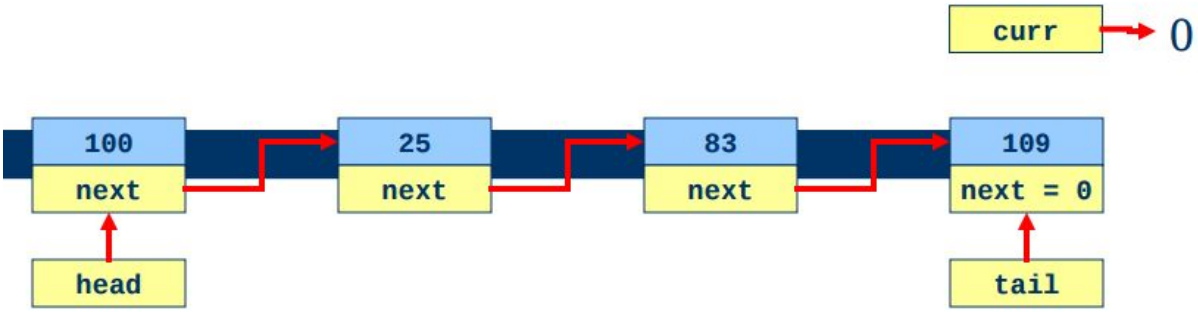


We can continue this process to get to the third and fourth node.





Finally we set curr pointer to the last node's next value which is NULL. At this point, when curr equals NULL we know that we have reached the end of the list and can stop iterating.



Here is the code to iterate through the linked list.

```
void printList() {
  Node *current = head;
  while (current != NULL) {
    cout << current->data << endl;
    current = current->next;
  }
```

Adding a node to a linked list

Next we need to know how to add a node to a linked. The example we will go through adds a node to the end of the linked list but it is useful to be able to add a node anywhere in the linked list.

For adding a node to a linked list there are two distinct situations that we need to deal with. The first is that the list is currently empty and the second is that there are already nodes in the list.

First we need to create a new node and give it its values. We then check if the head pointer is NULL. This will tell us if the list is empty. If the list is empty we simply set head to point to our new node and return from the function.

The second scenario is that the list isn't empty. If this is the case we need to iterate through the linked list in a similar manner as we did before except that we will check if current->next is equal to NULL instead of current. When current->next equals NULL we know that we are at the last node in the list. We then set current->next equal to the newNode we have just created.

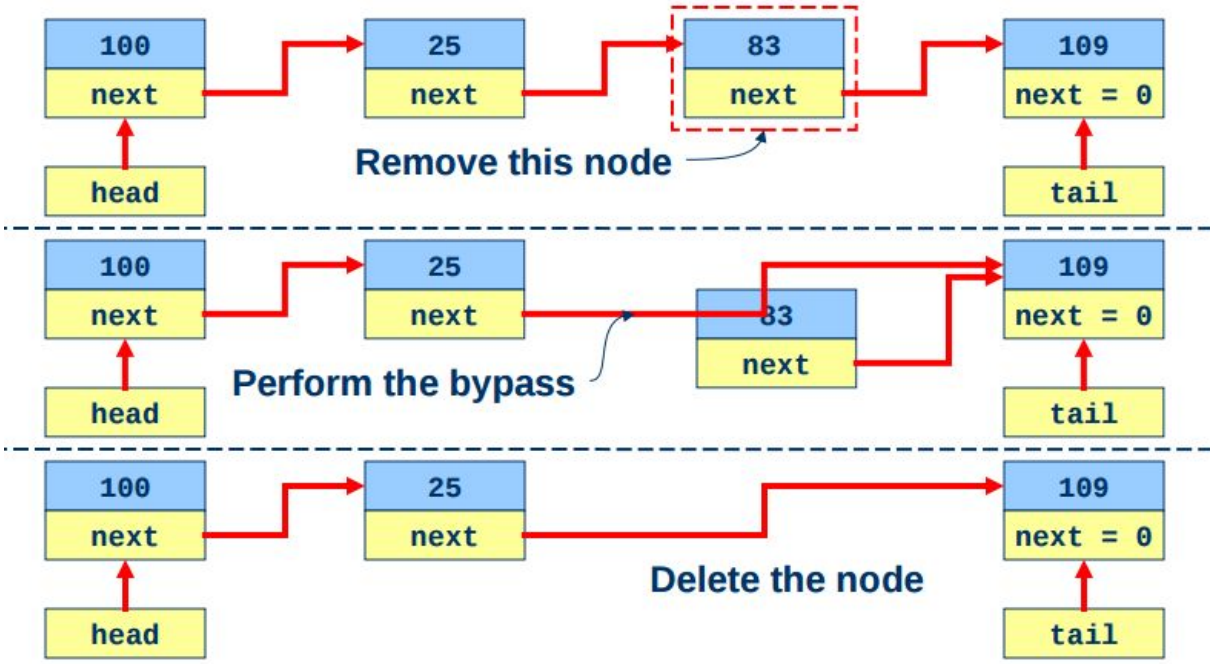
```
void addNode(int data) {
  Node *newNode = new Node;
  newNode->data = data;
  newNode->next = NULL;

  Node *current = head;
  if (head == NULL) {
    head = newNode;
    return;
  }
  while (current->next != NULL) {
    current = current->next;
  }
  current->next = newNode;
  return;
}
```

Removing a node from a linked list

To remove a node we need to

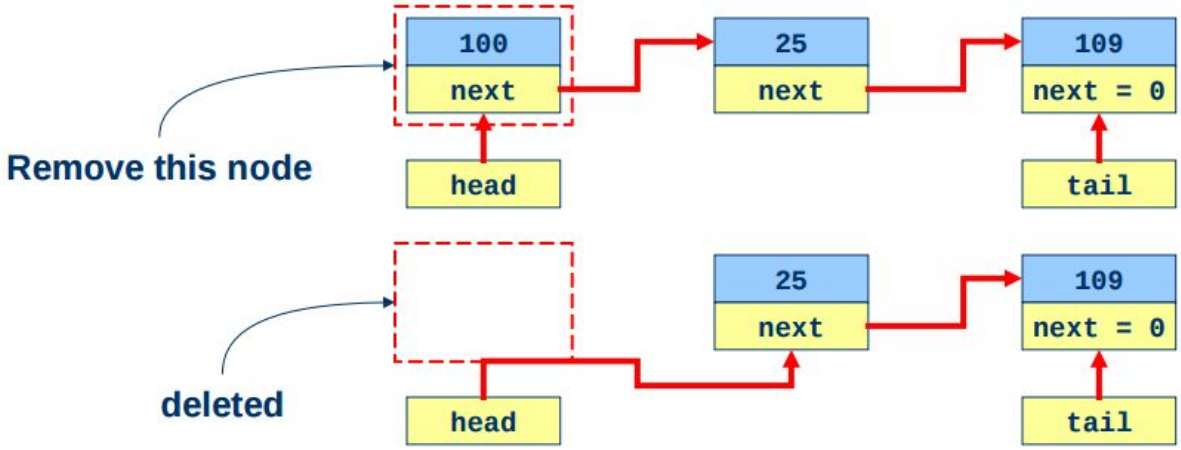
- Find the node. The node we are removing is the current node
- Make the previous node's next point to current->next
- Delete the node



Removing the head node

The simplest situation is that we just want to remove the first node in the list.
In this situation we

- Need to set the head node to point to the second node in the list. We do this by letting the head node equal the next value of the current node (the node we are deleting) **Set the head node equal to current->next**
- Delete the head node



This code will delete the first node in the linked list.

N.B. When you are removing a node always first make sure that the list isn't empty

```
void removeHead() {
  Node *current = head;
  while (current != NULL) {
    head = current->next;
    delete current;
    return;
  }
  cout << "This item is not in the list" << endl;
  return;
}
```

To remove a node from somewhere else in the list you need to first find the correct node. The most common ways of specifying a node to delete are by the data it stores are by its position in the list.

The easiest way to correctly remove a node is to have a second pointer to the previous node.

```
Node *current = head;
Node *prev = NULL;
```

While searching for the correct node you need to keep track of the previous node. The code below will move the current and previous node through the list.

```
prev = current;
current = current->next;
```

N.B. The order of these instructions is very important