Deleting Nodes from a Linked List

Kevin Browne

Deleting nodes

- We'll first look at deleting nodes from the head and tail of the list

- But as with inserting nodes, we could delete nodes in different ways too
  - e.g. delete the first node with a value that matches a given value
  - e.g. delete all nodes with a value that matches a given value

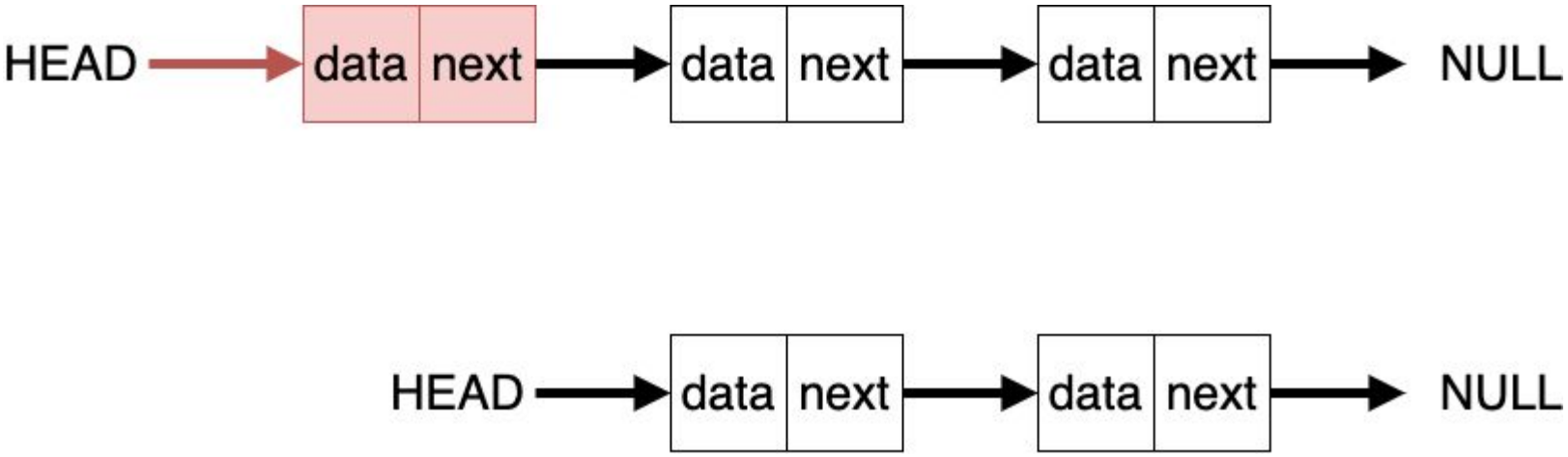
Deleting the head node

- If the head element is NULL/empty, we can just return NULL/empty

- If the head element is not NULL/empty, we need to return a pointer to the next node (even if it is NULL)
  - We need to remember to free the existing head node given it is deleted!
  - This may require us to store a pointer to the new head node in a variable before we can free the existing head

- We'll need to return a pointer to the new head too!

Deleting a node at the head of a non-empty list...



Deleting a node at the head...

```
Node* delete_at_head(Node *head)
{
  if (head == NULL) return NULL;
  else
  {
    Node *to_return = head->next;
    free(head);
    return to_return;
  }
}
```

Testing the function...

```
list1_head = delete_at_head(list1_head);
printf("\nPrint out list after deleting at head...\n");
print_list(list1_head);
```

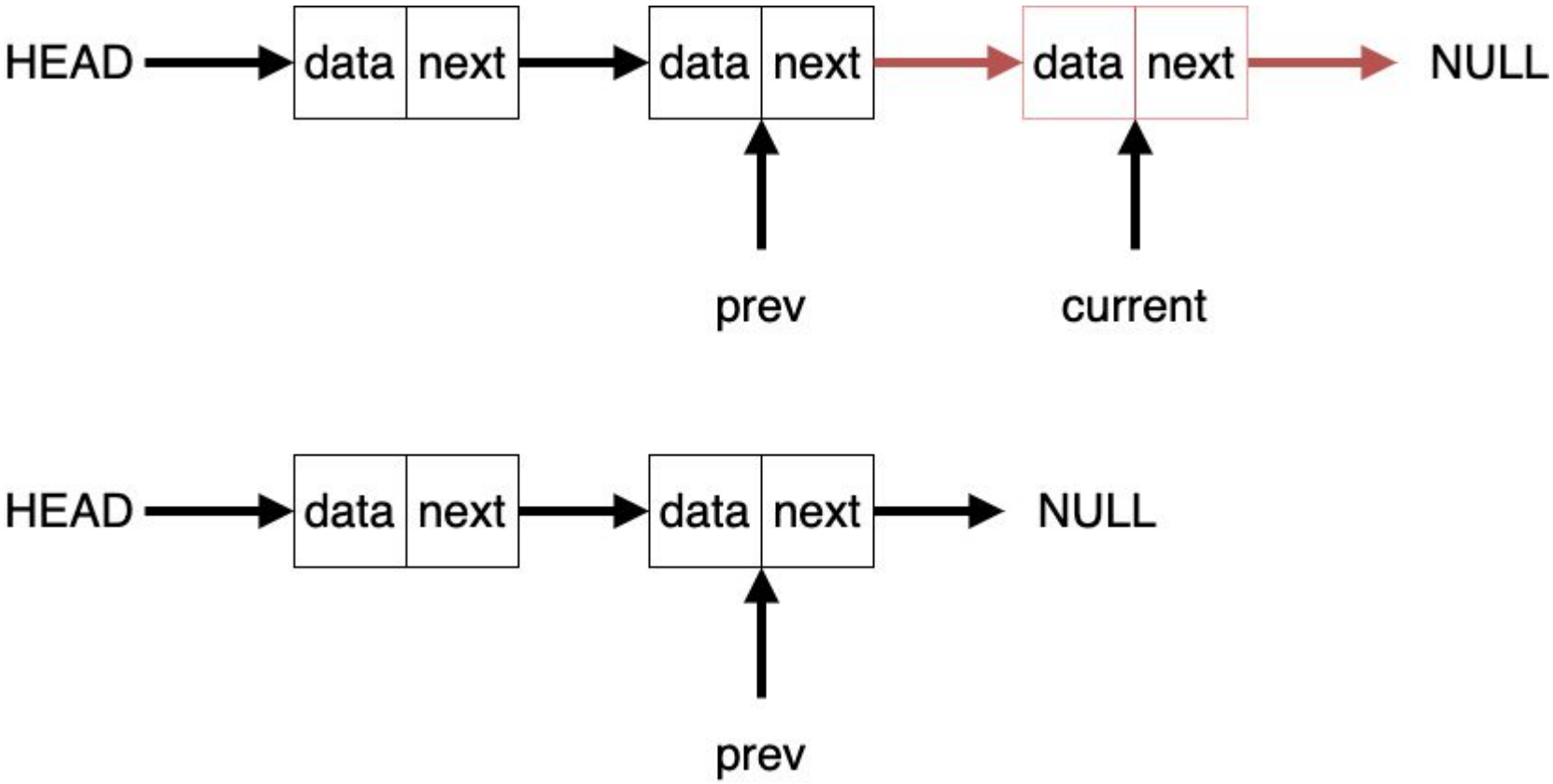
Output:

```
Print out list after deleting at head...
Node 0: 5
Node 1: 7
Node 2: 10
Node 3: 12
```

Deleting the tail node

- If the list is NULL/empty, we simply return NULL/empty
- If the list contains one node only, we free the node and return NULL/empty
- If the list not NULL/empty, we'll need to traverse list until we've reached the last two nodes
  - We need to change the pointer of the 2nd last node in the list to NULL, ending the list here instead
  - And we need to free the last node given it will no longer be used
  - We can use two pointers to keep track of the current and previous node
  - We return a pointer to the existing head node

Deleting at the tail of a list with more than one node



Deleting a node at the tail...

```
Node* delete_at_tail(Node *head)
{
  if (head == NULL) return NULL;
  else if (head->next == NULL)
  {
    free(head);
    return NULL;
  }
  else
  {
    Node *current = head;
    Node *prev = NULL;

    while (current->next != NULL)
    {
      prev = current;
      current = current->next;
    }
    prev->next = NULL;
    free(current);
    return head;
  }
}
```

Testing the function...

```
list1_head = delete_at_tail(list1_head);
printf("\nPrint out list after deleting at tail...\n");
print_list(list1_head);
```

Output:

```
Print out list after deleting at tail...
Node 0: 5
Node 1: 7
Node 2: 10
```