Singly Linked List - Deletion

In this lesson, we'll discuss the deletion operation on a singly linked list.

We'll cover the following

Structure

Each node contains a value and a pointer to the next node.

```
struct Node {
    int val;
    Node* next;

    Node (int val) {
        this->val = val;
        this->next = NULL;
    }
}
```

Deletion

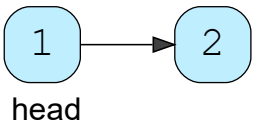
There are 2 cases:

- Delete head
- Delete node at a given position

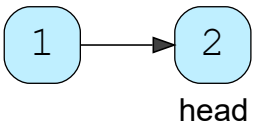
The worst-case for deleting a node is $O(N)$.

Delete head

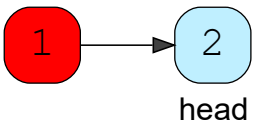
This is a straightforward case. Just point the head to the next node.



1 of 4



2 of 4

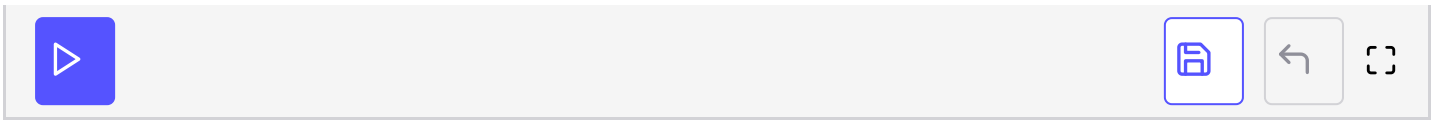


3 of 4

4 of 4

```
#include<iostream>

using namespace std;

struct Node {
  int val;
  Node* next;

  Node(int val) {
    this -> val = val;
  }
};

void print_list(Node* head) {
  struct Node* pCrawl = head;
  cout << " -> ";
  while (pCrawl != NULL) {
```

```
    cout << (pCrawl -> val) << " -> ";
    pCrawl = pCrawl -> next;
  }

  cout << "\n";
}

void insert_at_end(Node* &head, int val) {
  // List is empty
  if (head == NULL) {
    head = new Node(val);
    return ;
  }
  struct Node* pCrawl = head;
  while(pCrawl->next != NULL) {        // iterate to last node
    pCrawl =  pCrawl -> next;
  }
  pCrawl -> next = new Node(val);
}

void delete_head(Node* &head) {
  Node* temp = head;
  head = head->next;
  delete temp;
}


int main() {
  Node* head = NULL;
  insert_at_end(head, 1);
  insert_at_end(head, 2);
  insert_at_end(head, 3);

  print_list(head);
  delete_head(head); print_list(head);
  delete_head(head); print_list(head);
}
```

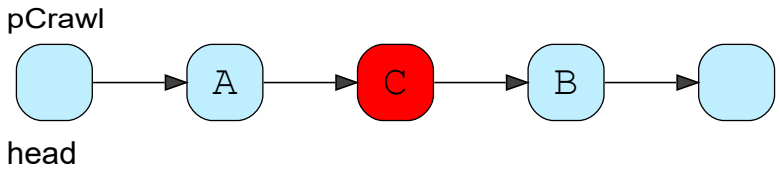


Delete at given position

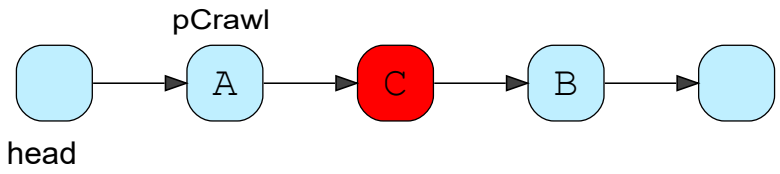
Let the node to be deleted be C, the previous node is A and the next node is B.

- Iterate to C
- A->next point to B
- Free C

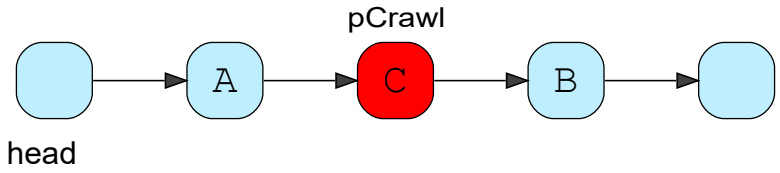
Task: Try to see how this algorithm works when you want to delete the last node.



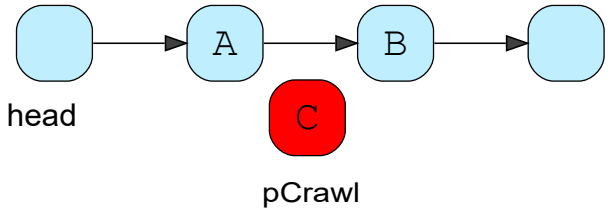
1 of 4



2 of 4



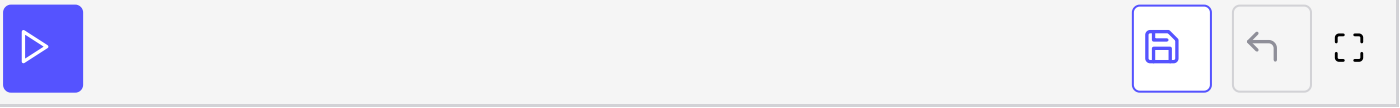
3 of 4



4 of 4

```
#include<iostream>

using namespace std;

struct Node {
  int val;
```

```
  Node* next;

  Node(int val) {

    this -> val = val;
  }
};

void print_list(Node* head) {
  struct Node* pCrawl = head;
  cout << " -> ";
  while (pCrawl != NULL) {
    cout << (pCrawl -> val) << " -> ";
    pCrawl = pCrawl -> next;
  }
  cout << "\n";
}

void insert_at_end(Node* &head, int val) {
  // List is empty
  if (head == NULL) {
    head = new Node(val);
    return ;
  }
  struct Node* pCrawl = head;
  while(pCrawl->next != NULL) {        // iterate to last node
    pCrawl =  pCrawl -> next;
  }
  pCrawl -> next = new Node(val);
}

void delete_at_position(Node* &head, int pos) {
  struct Node* A = head;
  for (int i = 0; i < pos - 1; i++) {
    A =  A->next;
  }
  Node *C = A->next;
  Node *B = A->next->next;
  A->next = B;
  delete C;
}

int main() {
  Node* head = NULL;
  insert_at_end(head, 1);
  insert_at_end(head, 2);
  insert_at_end(head, 3);
  insert_at_end(head, 4);
  insert_at_end(head, 5);

  print_list(head);
  delete_at_position(head, 2); print_list(head);  // 0-based position
  delete_at_position(head, 3); print_list(head);
}
```



With this, we'll end the chapter on linked lists. I'd suggest reading about *double*

and *circular* linked lists as well, a quick google search should do it.

Once you understand how to manipulate pointers to perform operations on singly linked lists, extending to other types of linked lists will be trivial.