Linked Lists

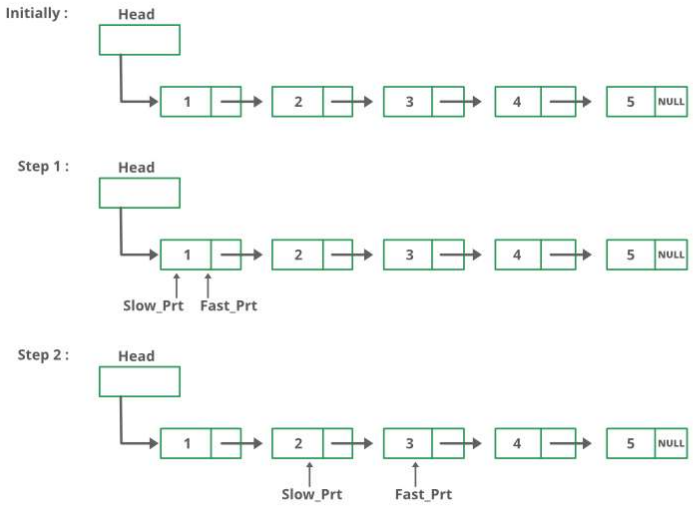
Exercises

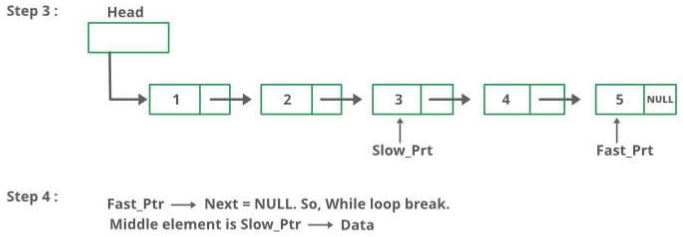
Linked List Problems

- http://cslibrary.stanford.edu/105/
- Interview questions
  - https://www.geeksforgeeks.org/top-20-linked-list-interview-question/
  - https://www.interviewbit.com/linked-list-interview-questions/
  - https://igotanoffer.com/blogs/tech/linked-list-interview-questions

Find the middle of a given linked list

- How will you find the middle element of a singly linked list without iterating the list more than once?



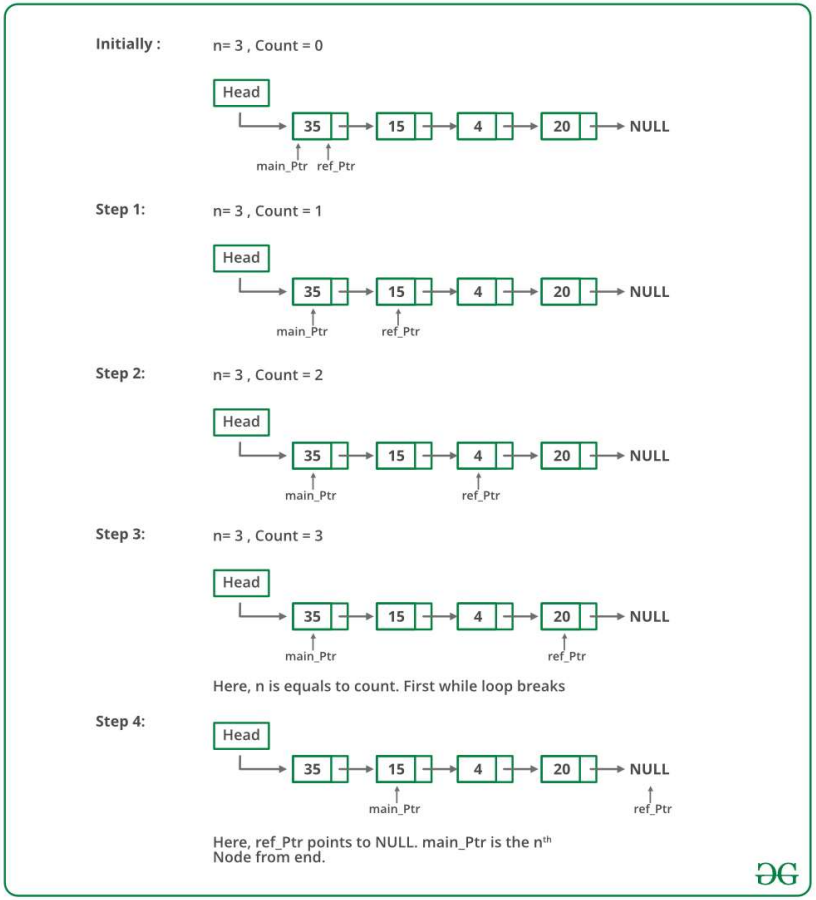


Code to find middle node

```
Node* getMiddle(Node *head)
{
    Node *slow = head;
    Node *fast = head;

    if (head)
    {
       while (fast != NULL && fast->next != NULL)
       {
          fast = fast->next->next;
          slow = slow->next;
       }
    }
    return slow;
}
```

Find n’th node from the end of a Linked List



Print a singly-linked list in reverse order

```
void printReverse(Node* head)
{
    // Base case
    if (head == NULL)
    return;

    // print the list after head node
    printReverse(head->next);

    // After everything else is printed, print head
    cout << head->data << " " ;
}
```

Trace

- What is the output of the following function for head pointing to the first node of the following linked list? 1->5->9->4->7->6?

```
void fun(Node* head){
  if(head== nullptr)
    return;
  cout << head->data << " ";

  if(head->next != nullptr )
    fun(head->next->next);
  cout << head->data << " ";
}
```

Reverse a singly-linked list

- Given pointer to the head node of a linked list, the task is to reverse the linked list. We need to reverse the list by changing the links between nodes.

Iterative code to reverse a linked list

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


while (current != NULL) {
    // Store next
    next = current->next;


    // Reverse current node's pointer
    current->next = prev;


    // Move pointers one position ahead.
    prev = current;
    current = next;
}
head = prev;
```

Merge two linked lists at alternate positions

- Given two linked lists, insert nodes of second list into first list at alternate positions of first list.
- If the first list is 5->7->17->13->11 and second is 12->10->2->4->6, the first list should become 5->12->7->10->17->2->13->4->11->6 and second list should become empty.
- If the first list is 1->2->3 and second list is 4->5->6->7->8, then first list should become 1->4->2->5->3->6 and second list to 7->8.

Code for alternating merge

```
Node *p = head1, *q= head2;
Node *pn, *qn;


// While there are available positions in p
while (p!= NULL && q!= NULL)
{
    // Save next pointers
    pn = p->next;
    qn = q->next;


    // Make q as next of p
    q->next = pn; // Change next pointer of q
    p->next = q; // Change next pointer of p


    // Update current pointers for next iteration
    p = pn;
    q = qn;
}


head2 = q; // Update head pointer of second list
```

Alternating split

- Split a given list into two smaller lists L1 and L2, which are made from alternating elements in the original list.
- If the original list is 0->1->0->1->0->1 then one sublist should be 0->0->0 and the other should be 1->1->1

Code for alternating split

```
Node aDummy;
Node* aTail = &aDummy;          // points to the last node in `a`
aDummy->next = NULL;
L1 = aTail;

Node bDummy;
Node* bTail = &bDummy;          // points to the last node in `b`
bDummy->next = NULL;
L2 = bTail;

Node* current = head;           // points to the original list head
while (current != NULL)    {
   aTail->next =current ;       // add at `a` tail
   aTail = aTail->next;         // advance the `a` tail
   current = current->next;
   if (current != NULL) {
      bTail->next = current;    // add at `b` tail
      bTail = bTail->next;      // advance the `b` tail
      current = current->next;
   }
}
aTail->next = NULL; bTail->next = NULL;
```

Swap two adjacent nodes in a doubly linked list

You are given a pointer **p** to the node that is going to be swapped with the node next to it. Assume **p** points to a node where both next and previous nodes exist.

```
Node *t = p->next;
p->next = t->next;
t->next->prev = p;
t->prev= p->prev;
p->prev->next = t;
p->prev = t;
t->next = p;
```