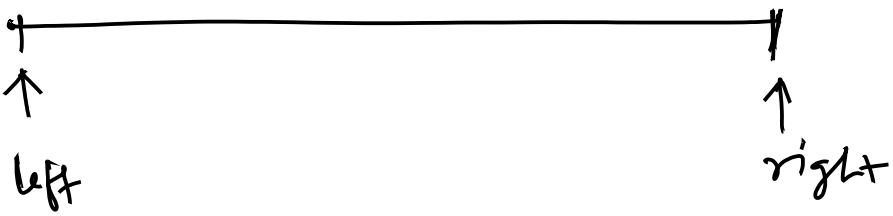
18/01/2023

Code Template

1. Two pointer : one input, opposite ends.

```
int fn( vector<int> &arr){
   int left =0;
   int right = int (arr.size()) -1;
   int ans =0;
    while ( left <right ) {
       // Prepare your logic Condition
      if (condition) {
          left ++;
       }
       else {
          right --;
        }

    }
   return ans ;
}
```



2. two pointer : two inputs

```
int fn( vector<int> &arr1 , vector<int> & arr2){
   int i=0, j=0, ans=0;
   while ( i< arr1.size() && j< arr2.size() ){
        // do some logic for generating Condition
        if (condition)  i++;
        else  j++
   }
   return ans;
}
```

On the basis of some logic

And Condition you need to move i or j one at a time.

Example

Merging of two sorted array

Merge step in Merge sort

1. Two Pointer: One input

```
int fn(vector<int> &arr) {
  int left = 0, right = 0;
  right = int(arr.size()) - 1;
  int ans = 0;
  while (left < right)
    // do some logic
    if (CONDITION) left++;
    else right--;
  }
  return ans;
}
```

2. Two pointer : two input

Merge method inside merge sort

```
int fn(vector<int> &arr1, vector<int> &arr2)
{
  int i = 0;
  int j = 0;
  int k = 0;
  While (i < arr1.size() && j < arr2.size()) {
    // do some logic here
    if (CONDITION) i++;
    else j++
  }
```

(left, right, mid)

```
while ( i < arr1.size() ){
    // do logic
    ++i;
}

while ( j < arr2.size() ){
    // do logic
    ++j;
}

// there might be some copying into
// array step also
return some ans. / no return.
}
```

* ③ Sliding window concept

```
int fn(vector<int> &arr){
  int left=0, ans=0, curr=0;

  for( int right=0; right < arr.size(); right ){
    // do some logic here to add arr[right] to curr
    while ( WINDOW_CONDITION_BROKEN){
        // remove arr[left] from curr
        left++;
    }

    // Update ans

  }
  return ans;
}
```

④ Build a prefix sum

```
vector<int> fn(vector<int>& arr){
 vector<int> prefix(arr.size());
 prefix[0] = arr[0];   <- initial state
 for(int i=1; i<arr.size(); i++){
   prefix[i] = prefix[i-1] + arr[i];
 }
 return prefix;
}
```

if you have

f(i-1) ↝ prefix sum until i-1

than adding ith value will be like

$$f(i) = f(i-1) + arr(i)$$

Recursive relation

⑤ Efficient string building

```
string fn(vector<char>& arr){
   return string(arr.begin(), arr.end());
}
```

Adding one by one character is not best for building string in c++ so giving the range of character will help code in optimization

⑥ Linked list : fast & slow pointer

```
int fn ( ListNode * head ) {
    ListNode * slow = head ;
    ListNode * fast = head;
    int ans=0;
    while ( fast != nullptr && fast→ next != nullptr) {
        //do logic
        slow = slow →next;
        fast = fast→ next→ next;
    }
    return ans ;
}
```

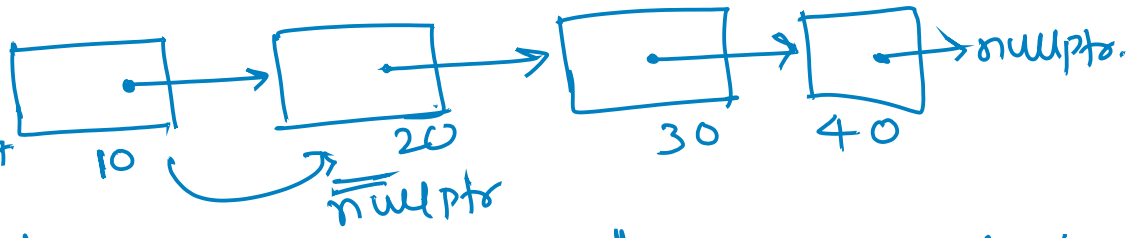
① finding a cycle in linked list

② One good application of this concept is breaking the loop in linked list, ③ find the mid point in Linked list etc.

⑦ Reversing a linked list (Singly list)

```
ListNode * fn( ListNode* head) {
    ListNode * curr = head;
    ListNode * prev = nullptr;
    while ( curr != nullptr) {
        ListNode * nextNode = (curr→next) ;
        curr→next = prev ;
        prev = curr
        curr = nextNode ;
    }
    return prev;
}
```

Input



head
|||
curr

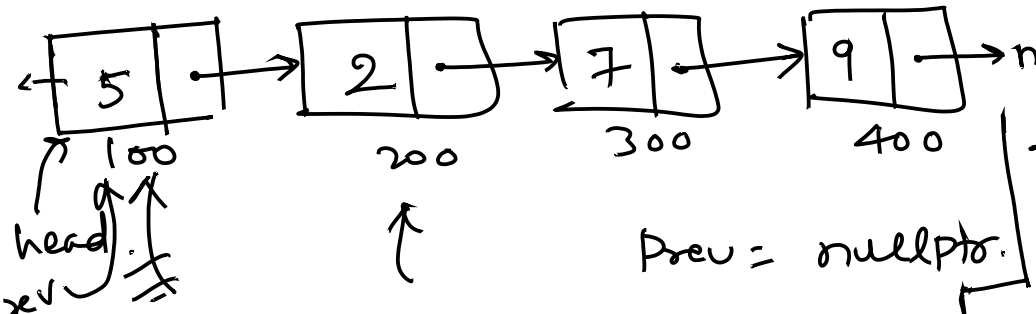
curr next = curr->next;
curr->next = prev;
prev = curr;
curr = next;

prev = nullptr :

prev = 10

nextNode = 20

10->next = nullptr;
prev = 10
curr = nextNode (20)



curr = 100

iterate (curr != nullptr)

backup = 200

curr->next = prev

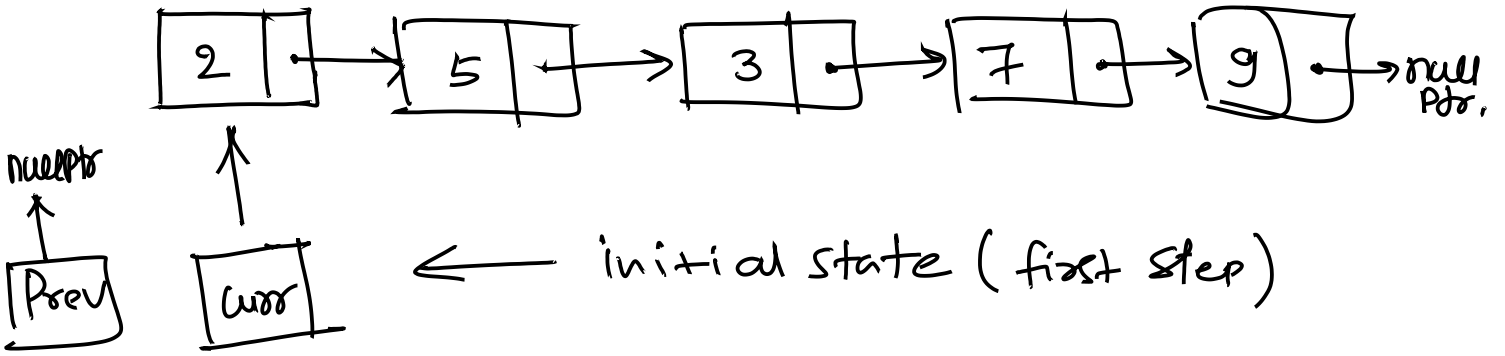
prev = 100 ;
curr = backup ;

prev = nullptr.

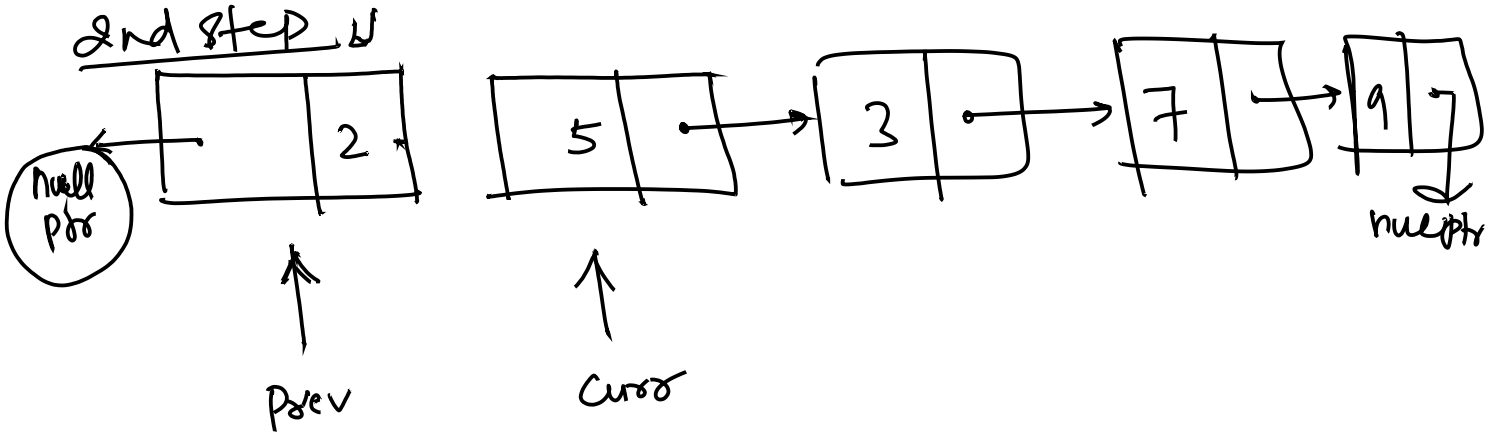
- Explain the code template

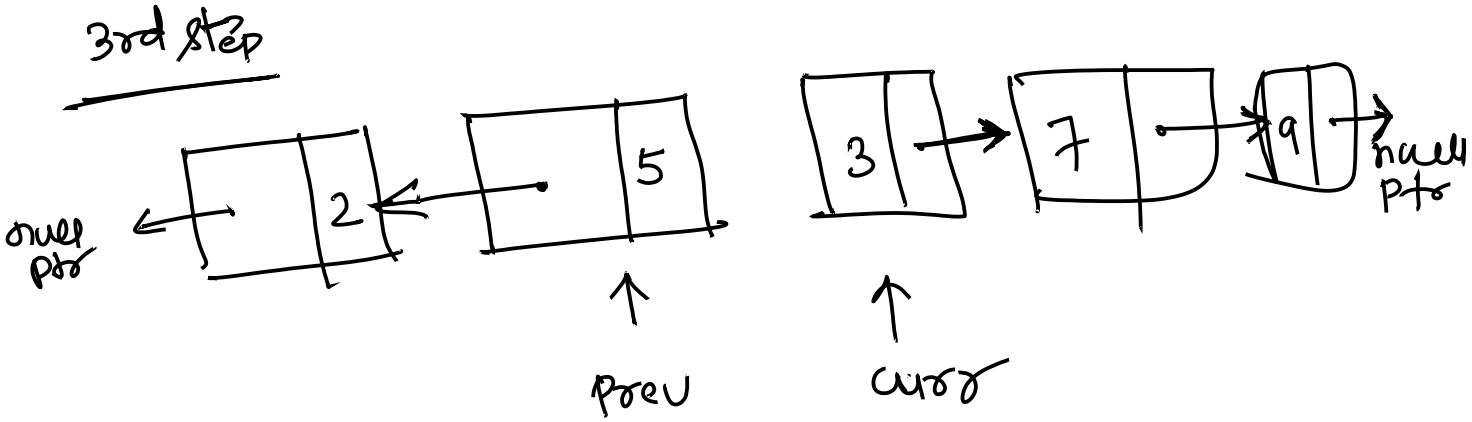
But still not clear?

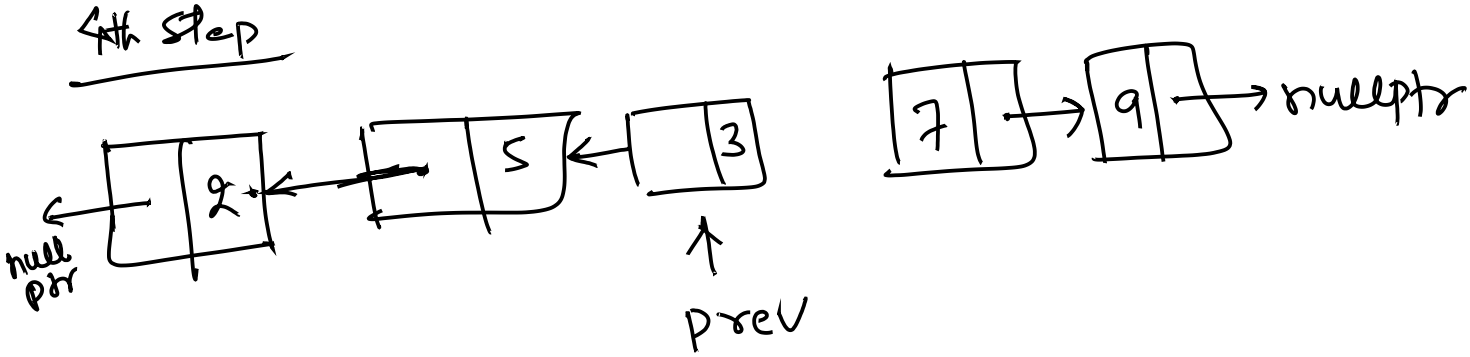
Still not very efficiently explained.

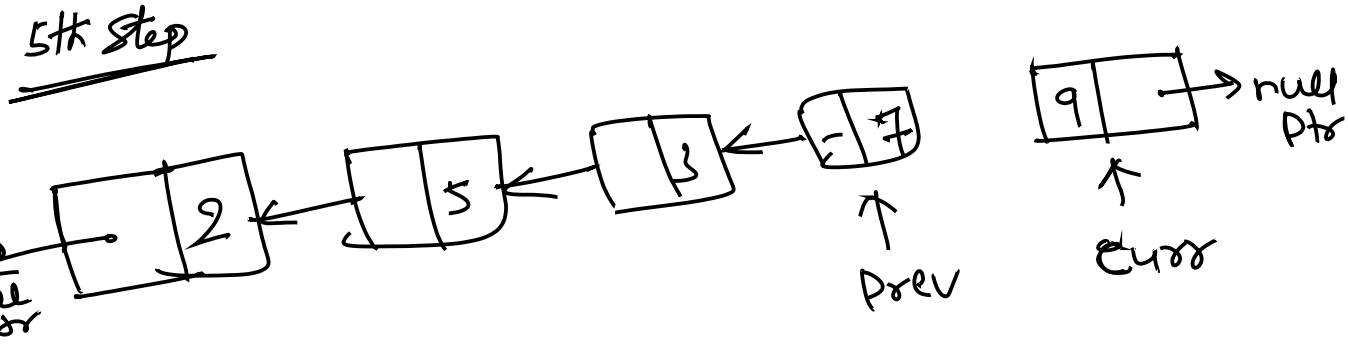


← initial state (first step)

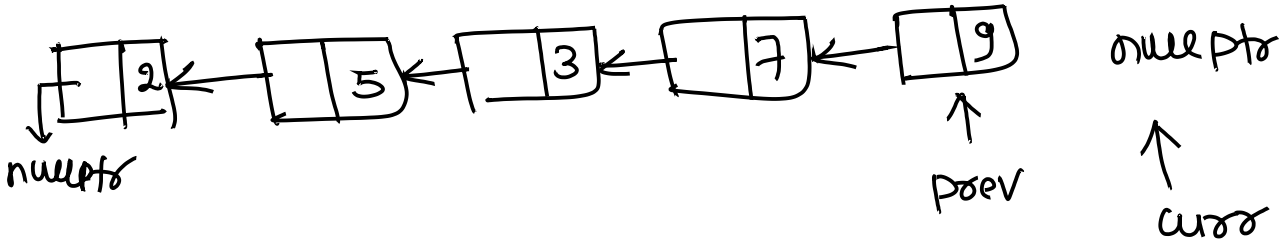








6th step



As soon as your curr reached to nullptr your iteration will end

And so, you have to return your prev pointer from here as a new head