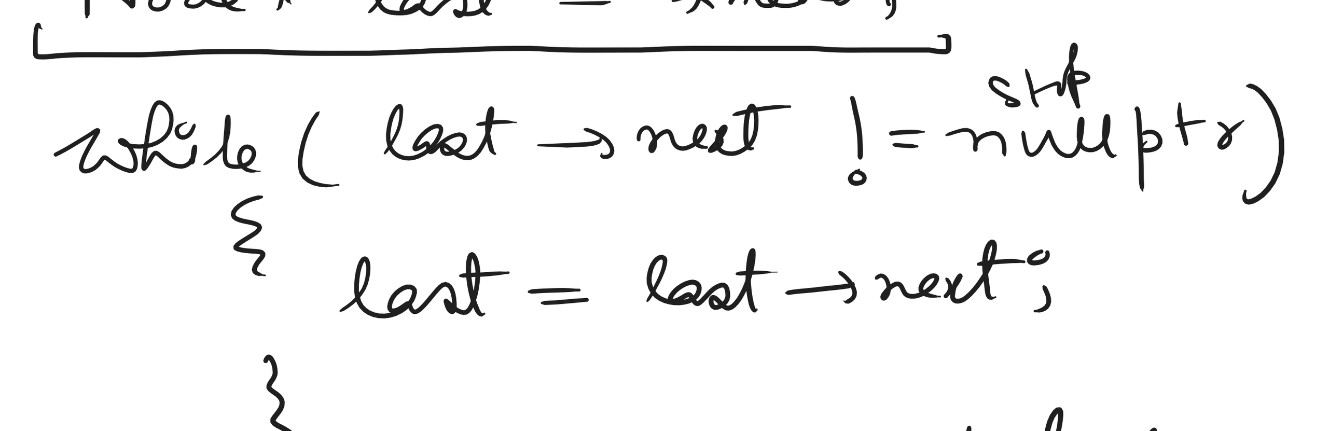
Insert At Tail Operation: →

step 1 : Create the node.
Node * newNode = new Node();
newNode → data = newValue;

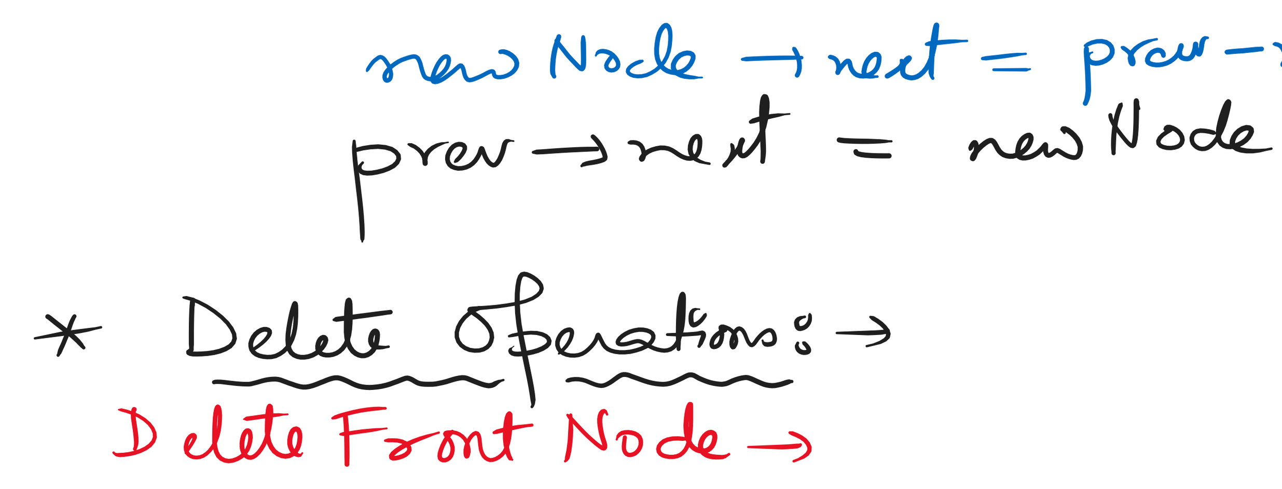
Step 2 : newNode → next = nullptr;
(x) → null
tail

↳ empty? if(*head == nullptr)
○ → null
head



Node* last = *head;
while (last → next != nullptr) stop
{
last = last → next;
}
last → next = newNode;

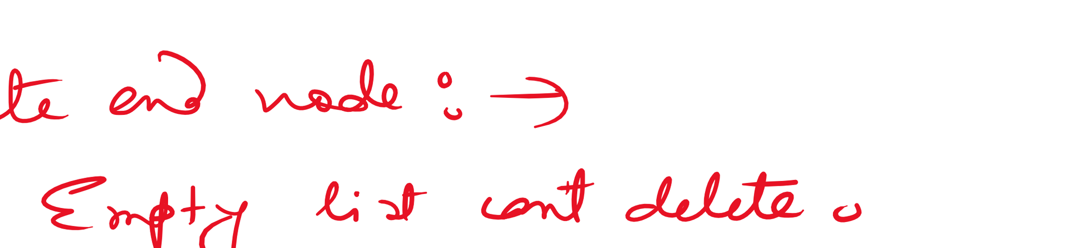
Insert After Specific:



newNode → next = prev→next
prev → next = newNode

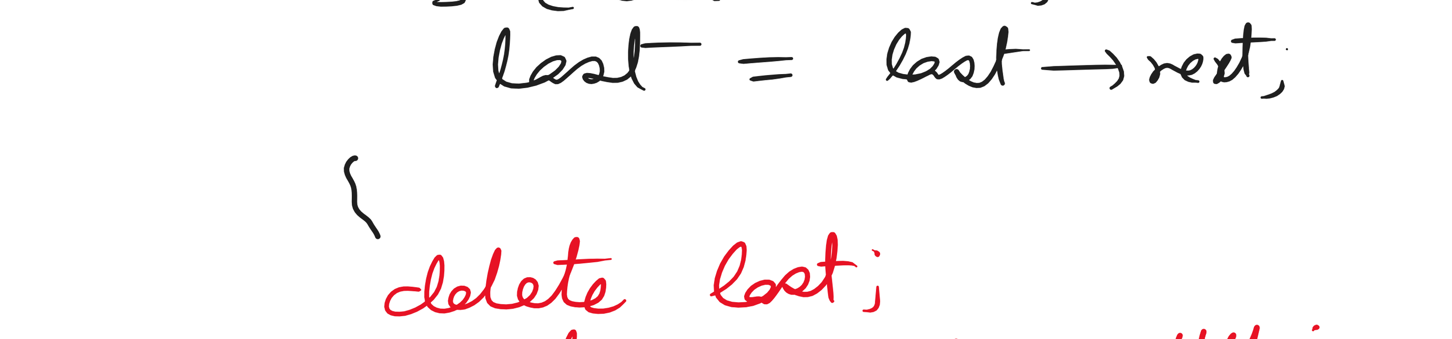
* Delete Operations: →
Delete Front Node →
Delete End Node →
Delete Node With Specific Target.

Empty!
×

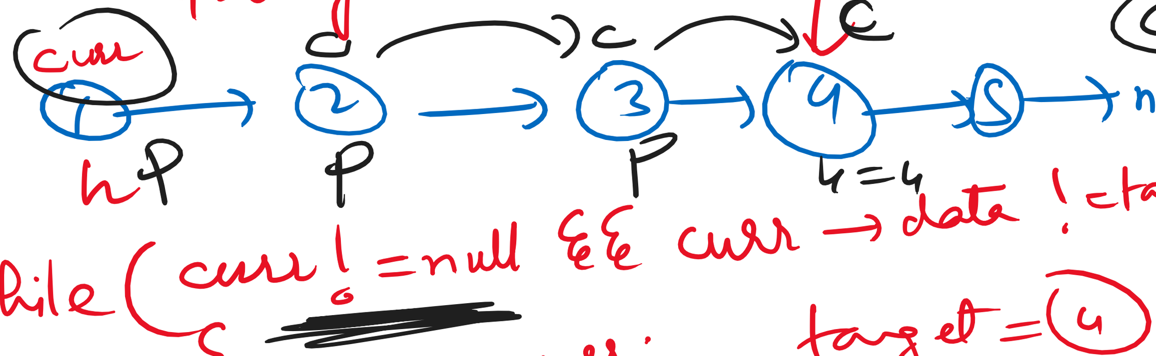


Node* temp = *head;
*head = (*head)→next;
delete temp;

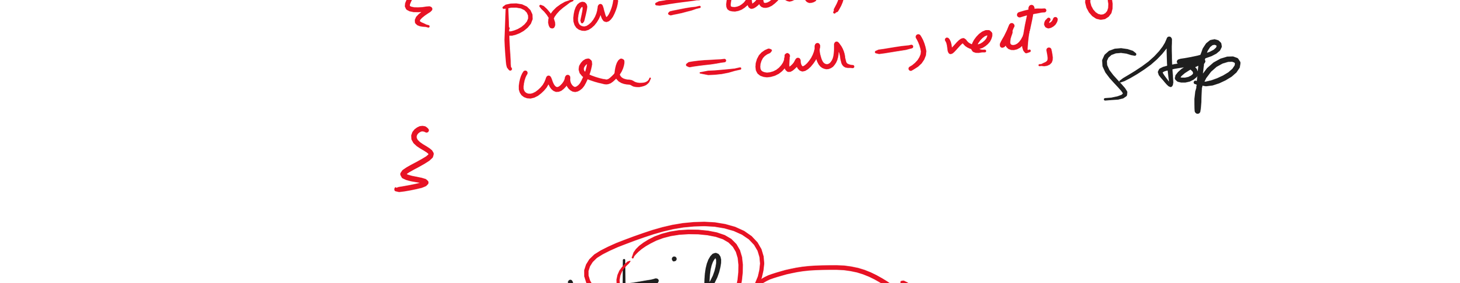
Delete end node: →
① Empty list cant delete.
② Single node in the list.
delete head.
(1) → null
head
if ((*head)→ next == nullptr){
}



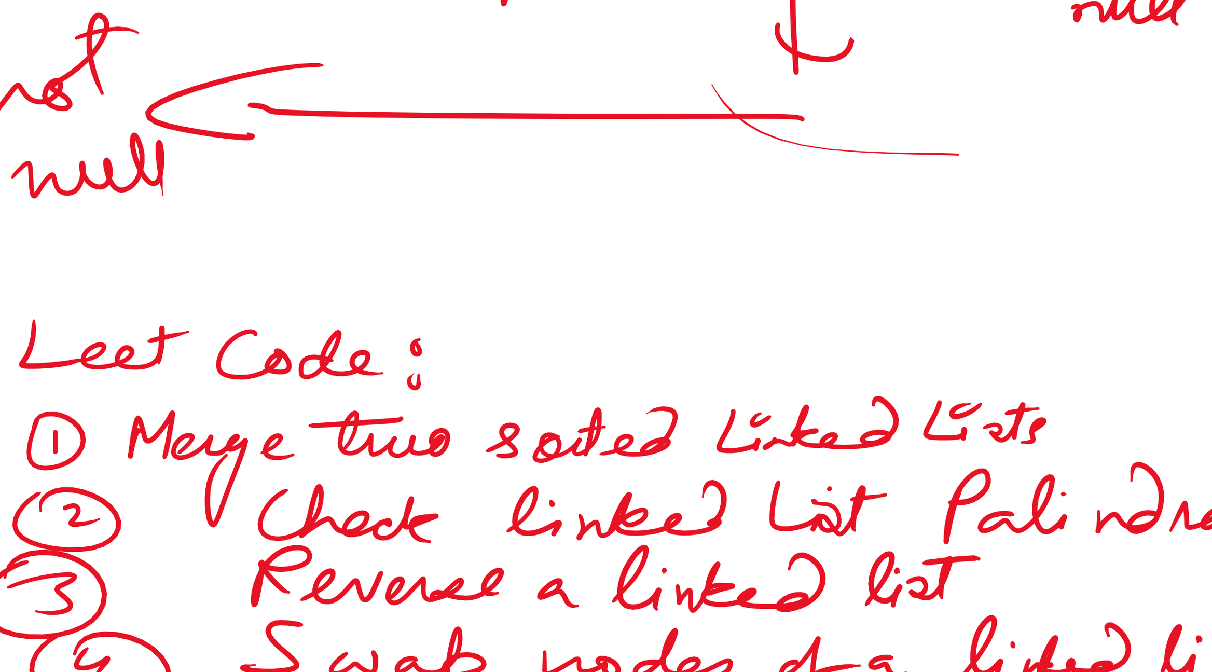
while (last → next != nullptr) {
sec last = last;
last = last → next;
}
delete last;
Sec last → next = null;



target = 1



while(curr != null && curr → data != target)
{
prev = curr;
curr = curr → next;
}
target = 4
stop

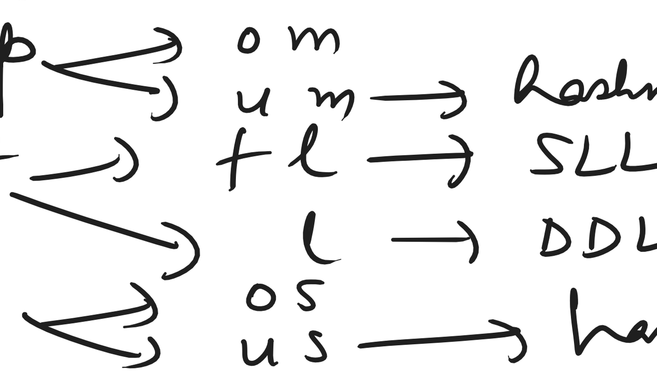


* Leet Code:
① Merge two sorted Linked Lists
② Check linked List Palindrome
③ Reverse a linked list
④ Swap nodes of a linked list
⑤ Sort a given linked list
⑥ Middle of a linked list.

stack

#in. <stack>
STL

* Circular linked List: ADA
9377



Hashtable

STL → Standard Template Library
stack
queue
vectors<int>
map → om
map → um → hashmap ***
list → fl → SLL
list → l → DDL
sets → os
sets → us → hashmap **