Guideline: How to find file?

→There will be a link to a year2 repo, Inside repo u will see two folder (web_design and algorithm). Go to “algorithm” u will find lap_pratice folder it’s the one u looking for.

Challenge 1: Insert at the front

A: Insert a new node at the start of a link listed. The complexity is O(1).
Discuss: Inserting at the front of array easier because of we just set next of new node to head and set head to the address of new node.

```
//Challenge 1: Insert at the front
Node* new_node = new Node{8,nullptr};
new_node->next = head;
head = new_node;
n++;
```

Challenge 2: Insert at the end

A: Append a new node to the end of a link list. The complexity is O(n).
Discuss: Yes, we need to traverse the entire list. This differ from array because of we need to loop one by one node in list to find the end list unlike array we can just hop to the end of array by index.

```
//Challenge 2: Insert at the end
cur = head;
for (int i=0; i<n-1; i++){
  cur = cur->next;
}
Node* new_node1 = new Node{14,nullptr};
new_node1->next = cur->next;
cur->next = new_node1;
n++;
```

Challenge 3: Insert in the middle

Discuss: Two pointer needed to be change the one node before middle and a node after the middle. Compare to the shifting in array it’s faster and no need shift anything.

```
//Challenge 3: Insert at the middle
cur = head;
for (int i=0; i<(n/2); i++){
  cur = cur->next;
}
Node* new_node2 = new Node{99,nullptr};
new_node2->next = cur->next;
cur->next = new_node2;
n++;
```

Challenge 4: Delete from the front

Discuss: The head pointer changed to the next one of the delete node. The deleted node's memory disappear.

```
//Challenge 4: Delete from the front
cur = head;
Node* temp = cur;
cur = cur->next;
delete temp;
head = cur;
n--;
```

Challenge 5: Delete from the end

Discuss: By looping to the node before end of list.

```
//Challenge 5: Delete from the end
cur = head;
for (int i=0; i<n-2; i++){
  cur = cur->next;
}
temp = cur->next;
cur->next = nullptr;
delete temp;
n--;
```

Challenge 6: Delete from the middle

Discuss: The pointer next of node before middle change to one after middle node. If we forget to free memory this can be causing bug to the code.

```
//Challenge 6: Delete from the middle
cur = head;
for (int i=0; i<(n/2); i++){
  cur = cur->next;
}
Node* cur1 = cur->next;
cur->next = cur1->next;
delete cur1;
n--;
```

Challenge 7: Traverse the list

Discuss: They all point to other in list.

```
//Challenge 7: Traverse list
cur = head;
while (cur){
  cout << cur->value << " ";
  cur = cur->next;
}
cout << "\n";
```

Challenge 8: Swap two nodes

Discuss: My opinion swap value faster than swap address back and forth.

```
//Challenge 8: Swap two Node
cur = head;
cur = cur->next;
head->next = cur->next;
cur->next = head;
head = cur;
```

Challenge 9: Search in link list

Discuss: Search in linked list is like linear search (O(n)), but arrays are faster for random access.

```
//Challenge 9: Search the list
cur = head;
while (cur->value != 3){
  cur = cur->next;
}
cout << cur->value << "\n";
```

Challenge 10: Compare with array

1. **O(1) in linked list but O(n) in array:** Insert or delete at the beginning.
2. **Faster in arrays:** Accessing elements by index (random access).
3. **Memory management:** Each node uses extra memory for pointers; memory must be allocated and freed manually.
4. **Head pointer:** Points to the first node; needed to access the list.
5. **If head is lost:** The whole list becomes inaccessible.

Scenario	Better Choice	Reason
1. Real-time scoreboard	**Linked List**	Easy add/remove at ends.
2. Undo/Redo feature	**Linked List**	Easy insert/delete at front.
3. Music playlist	**Linked List**	Add/remove songs flexibly.
4. Large dataset search	**Array**	Fast random access.
5. Bank queue simulation	**Linked List**	Insert at end, delete at front (FIFO).
6. Inventory system	**Array**	Quick index lookup.
7. Polynomial addition	**Linked List**	Dynamic insertion/deletion.
8. Student roll-call system	**Array**	Fixed order and easy indexing.