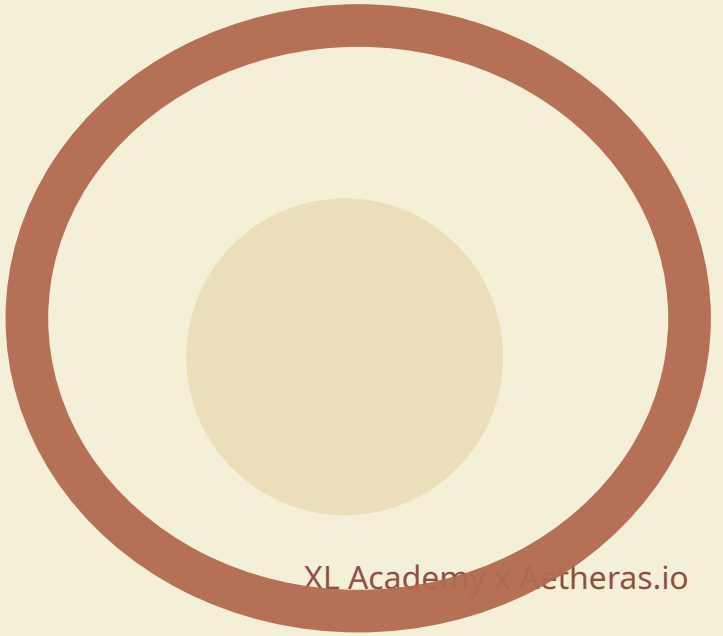
L06

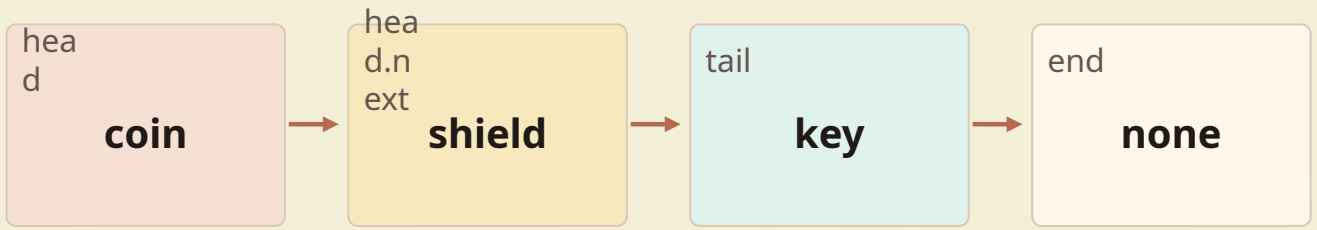
AI Game Development & Computer Science Summer Program

Lecture 6: ListNode Chains and Node Operations



A ListNode stores one value and one next step

Every operation comes down to reading or changing a `next` link.



One node knows

- Its own `value`.
- The next node.
- Nothing about earlier nodes.

One operation asks

- Which node am I holding?
- Which `next` link do I read?
- Which `next` link changes?

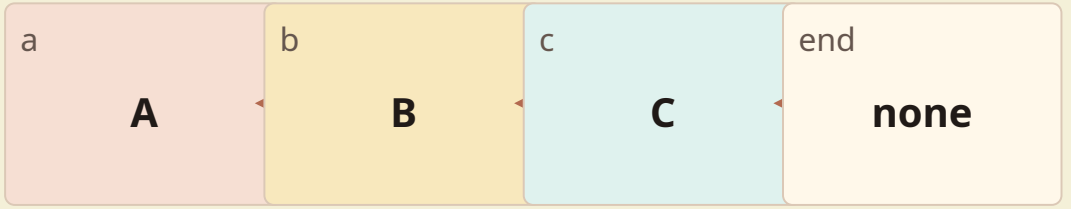
Boundary case

- The chain may be empty.
- The target may be first.
- The target may be missing.

Make the node type small enough to trace

Start with strings while learning the pointer moves.

```
type ListNode = {
  value: string;
  next?: ListNode;
};

const a: ListNode = { value: "A" };
const b: ListNode = { value: "B" };
const c: ListNode = { value: "C" };

a.next = b;
b.next = c;
```

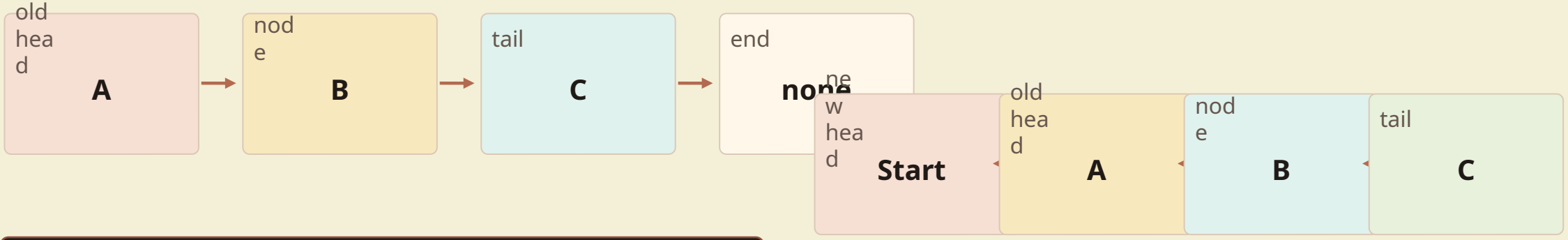


Trace rule

- First draw the chain.
- Then write the code.
- After each assignment, redraw what changed.

Adding at the front changes the head

The new node points at the old first node.




```
const addFront = (
  head: ListNode | undefined,
  value: string,
): ListNode => {
  return { value, next: head };
};

const head = addFront(a, "Start");
```

What changed?

- The returned head is new.
- The old head is still in the chain.
- No old node had to change.

Search by walking one node at a time

F xt`.

```
const findNode = (
  head: ListNode | undefined,
  target: string,
): ListNode | undefined => {
  let current = head;
  while (current) {
    if (current.value === target) return current;
    current = current.next;
  }
  return undefined;
};
```

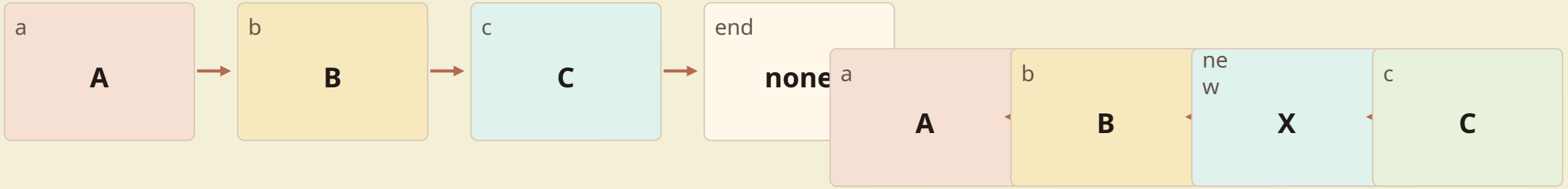
current	value	target B?	next
a	A	no	b
b	B	yes	return b
c	C	-	not reached

Missing target

- If the loop reaches no node, return `undefined`.

Insert between two nodes by saving the old next

The new node points forward before the previous node points to it.




```
const insertAfter = (node: ListNode, value: string): void
=> {
  const after = node.next;
  const inserted: ListNode = { value, next: after };
  node.next = inserted;
};

insertAfter(b, "X");
```

Order matters

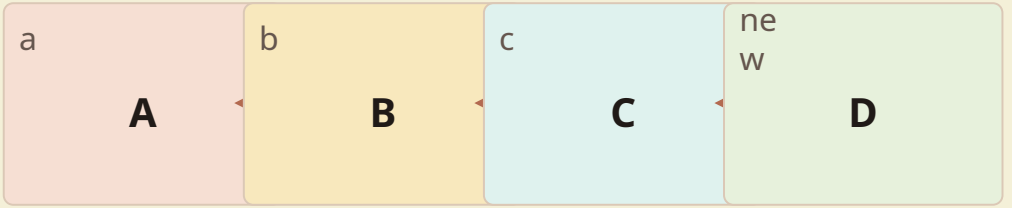
- Save `node.next` first.
- Set `inserted.next` to the saved node.
- Set `node.next` to the inserted node.

Append at the end by finding the tail

T

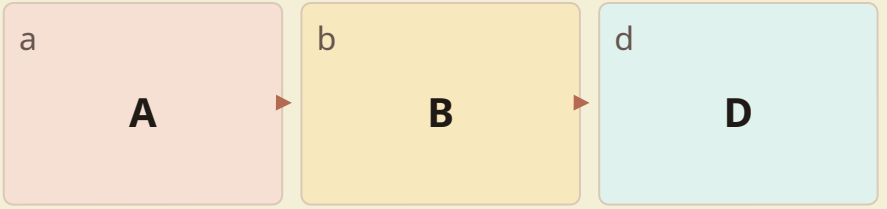
```
const append = (head: ListNode, value: string): void
=> {
  let current = head;
  while (current.next) {
    current = current.next;
  }
  current.next = { value };
};

append(a, "D");
```

current	current.next?	action
a	b	move
b	c	move
c	none	attach D



Remove a middle node by skipping it

The previous node points around the removed node.




```
const removeAfter = (node: ListNode): ListNode |
undefined => {
  const removed = node.next;
  if (!removed) return undefined;
  node.next = removed.next;
  return removed;
};

removeAfter(b);
```

What changed?

- `b.next` used to point at C.
- `b.next` now points at D.
- C still exists, but the chain no longer reaches it.

Removing the head is a different case

```
const removeFirst = (
  head: ListNode | undefined,
  target: string,
): ListNode | undefined => {
  if (!head) return undefined;
  if (head.value === target) return head.next;
  // otherwise search for the previous node
  return head;
};
```

case	operation	return
empty chain	nothing to remove	undefined
target is head	skip head	old head.next
target later	change previous.next	same head

Function contract

- Any operation that may change the first node should return the head.

Use object values after the pointer moves are clear

The same operations work when each value has fields.

Question	Needed fields
Can this item be used?	id, quantity
What should be shown?	name, quantity
How rare is it?	rarity
Can it stack?	maxStack

```
type InventoryItem = {
  id: string;
  name: string;
  quantity: number;
};

type ItemNode = {
  value: InventoryItem;
  next?: ItemNode;
};
```

Design habit

- Pick fields from real questions.
  Then reuse the same add/search/insert/remove patterns.

Trace operations with a before and after chain

A correct explanation names exactly one changed link.

operation	must identify	common mistake
add front	new head	forget to return it
append	tail node	stop one node too late
insert	previous and after	lose the old next
remove	previous and removed	skip the wrong node

Trace sentence

- Before: B.next points to C.
- Operation: insert X after B.
- After: B.next points to X, and X.next points to C.

Code review question

- Which `next` assignment changed the chain?

AI can draft helpers, but you verify the links

Treat generated code as a trace candidate, not as proof.

Prompt

- Here is my `ListNode` type.
- Write `append`, `insertAfter`, and `removeAfter`.
- Include empty, head, middle, and missing cases.

Your review

- Draw the before chain.
- Draw the after chain.
- Circle each changed `next` link.
- Reject code that loses the old next node.

If you cannot trace the pointer change, do not trust the helper yet.

Studio build: complete a node-operation helper

```
type ListNode = { value: string; next?: ListNode };

const nodeValues = (head: ListNode | undefined): string[]
=> {
  const values: string[] = [];
  for (let current = head; current; current =
current.next) {
    values.push(current.value);
  }
  return values;
};
```

Choose one

- Add a node at the front.
- Search for a node.
- Insert between two nodes.
- Append at the end.
- Remove a node.

Check with values

- Before: `["A", "B", "C"]`.
  After: write the exact array you expect.

Lab plan: practice, build, apply

Use the main lab block to turn the concept into working evidence.

Guided practice

- Draw `A -> B -> C` before writing code.
- Trace search by moving `current = current.next`.
- Name the exact `next` link changed by each operation.

Build target

- Complete two helpers: `addFront`, `insertAfter`, `append`, or `removeAfter`.
- Check each helper by listing node values before and after.

Project time

- Choose one operation that could support P1 data.
- Save one before/after chain trace as evidence.

P1: Game Rules Milestone

L03-L15

Bonus activities

Choose one if the main lab is solid and tested.

Stretch options

- Write `removeFirst(head, target)` with empty, head, middle, and missing cases.
- Use object values after the string-node operations work.
- Ask AI for sample tests and add the missing edge case.

Done when

- The main lab still works.
- You can explain the tradeoff or edge case.
- You saved one proof object: test, trace, screenshot, or note.

P1: Game Rules Milestone

L03-L15

Checkpoint

Move on when the data shape makes the program easier to explain.

Exit ticket

- One ListNode shape.
- One object type with three useful fields.
- One search or count helper over a chain.
- One AI-generated sample you accepted or rejected.

Next class preview

- Higher-order functions.
- Callbacks that process one node at a time.
- Transforming collection logic without hiding traversal.