Data Structures

The Node

Mostafa S. Ibrahim
Teaching, Training and Coaching since more than a decade!

Artificial Intelligence & Computer Vision Researcher
PhD from Simon Fraser University - Canada
Bachelor / Msc from Cairo University - Egypt
Ex-(Software Engineer / ICPC World Finalist)

Recall: Array and Vector

- Array is static. You can’t delete/insert/expand
- Vector was our way to get a dynamic array
  - The clue was a pointer to array in the data section
  - We then can simulate insert/delete over the structure
  - If we have to expand, we create new memory
- Vector **pros and cons**
  - Pros: Now more dynamic + still O(1) access to any position
  - Cons: Memory block reallocation and data copies during expansions = O(n)
  - Cons: Array is **contiguous** in memory, what if new requested memory is not available!?
- Can we avoids these memory issues?
  - E.g. **expanding** the content with a single element is always **O(1)**
  - Definitely pointers will be our friend in that!

Intuition

- We can create a single integer: **int* val = new int**
- We can create several **separate** integers the same way
- We can expand with more separate values
- But this is not useful so far!
- We need them to be **linked** not separate!

```
int *val1 = new int(6);
int *val2 = new int(10);
int *val3 = new int(8);
int *val4 = new int(15);
```

Intuition

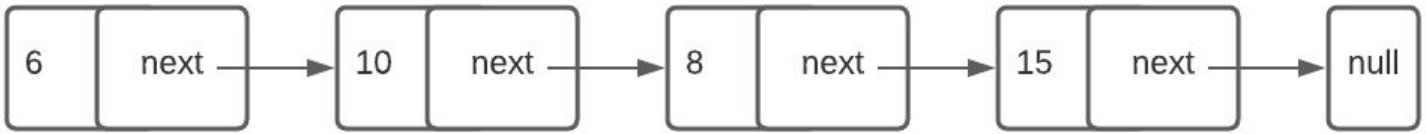
- Can we group them together?
- What if we also create other 4 pointers to **link** them together?
  - Link from 1 to 2
  - Link from 2 to 3
  - Link from 3 to 4
  - Link from 4 to ??? Flag to stop!
- What about a class that has 2 variables
  - The int value
  - The pointer to the link to the next value?!
  - Let's call it a **Node**

```
int *val1 = new int(6);
int *val2 = new int(10);
int *val3 = new int(8);
int *val4 = new int(15);
```

Node Data Structure

- If we created this class, we can easily create the 2 things
  - The new data
  - And its link (next) to the next value
  - But this link must point to something of also the same type (data/next)

```
struct Node {
    int data;
    Node* next;
    // Pointer to SAME type
    Node(int data) : data(data) {}
};
```

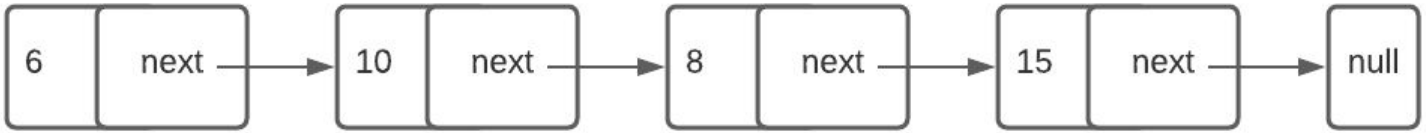


Create and Link!

- Let’s create 4 objects and set data
- Let’s link each object with next one
- To mark the last node
  - Usee nullptr

```
// Create 4 objects and set data
Node* node1 = new Node(6);
Node* node2 = new Node(10);
Node* node3 = new Node(8);
Node* node4 = new Node(15);

// Set 4 links
node1->next = node2;    //1-2 link
node2->next = node3;    //2-3 link
node3->next = node4;    //3-4 link
node4->next = nullptr;  //4-E link
```

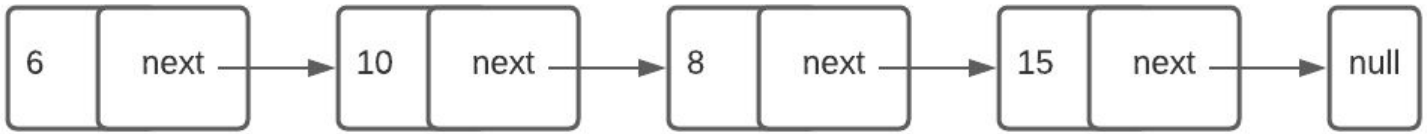


Navigate!

- Now, given **ONLY** the first node (**head**), we can move to any next node
  - next->next->next
- node1->next is node2
- and node2->next is node 3
- Then node1->next->next is node 3

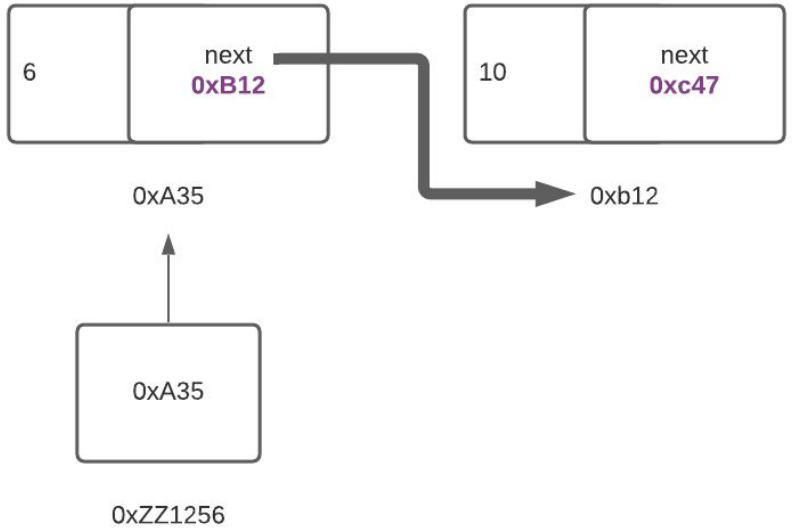
```
node1->next = node2;    //1-2 link
node2->next = node3;    //2-3 link
node3->next = node4;    //3-4 link
node4->next = nullptr;  //4-E link

// Output is 15 for all of them
cout<<node1->next->next->next->data<<"\n";
cout<<node2->next->next->data<<"\n";
cout<<node3->next->data<<"\n";
cout<<node4->data<<"\n";
```



Memory details

- There are 4 addresses in our model
- **Only** the node address itself is the most useful (which is used in the **next**)
- Others are usually useless addresses
  - Address of pointer looking to node (0xZZ1246)
  - Address of pointer looking to the next
  - Address of the data variable
  - Don’t distract yourself with them



Your turn

- Make sure you understand today content
- Create and link nodes
- Play with them
- Be careful the last node when its value is Null
  - Don’t print its data!
  - Don’t get its next

"Acquire knowledge and impart it to the people."

"Seek knowledge from the Cradle to the Grave."