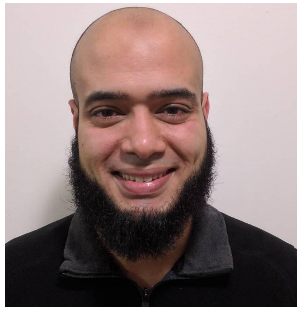
Data Structures

Binary Tree Traversal 1

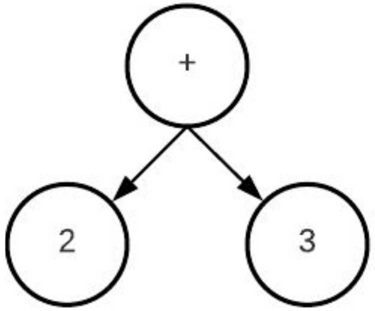
Mostafa S. Ibrahim
Teaching, Training and Coaching since more than a decade!

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



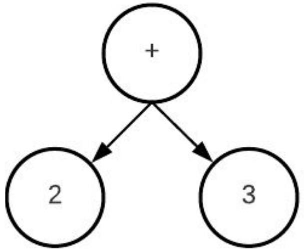
Tree Traversal

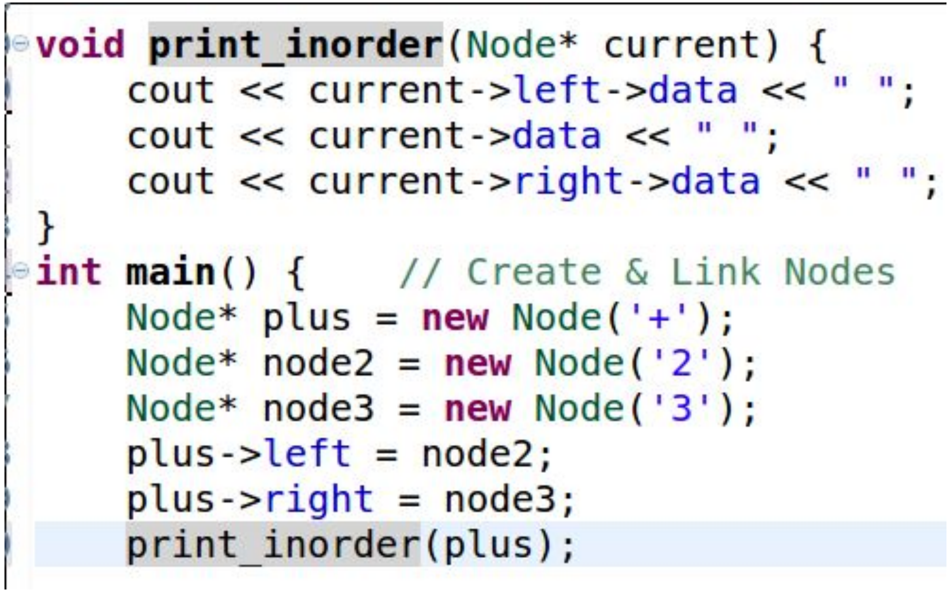
- **Traversal** Terminology: Walk through the elements of a data structure.
- We want to implement: void print(Node* current)
  - Goal: In a systematic way print content of subtree starting at current!
- Let's create an **Expression Tree** (leaves are operands and others operators)
  - Below represents 2 + 3
  - We can draw complex expressions: e.g. (2+3)*4
  - Assume we have such a simple 2-levels tree
    - Try to construct and implement print
    - It should print: **2 + 3**



Print Expression Tree: 2 + 3

- Simply print function prints left node value, then myself then right node value
- Let's call that LVR
  - L = left subtree (2)
  - V = Current node value (+)
  - L = right subtree (3)
  - This is inorder traversal
    - V = in the middle

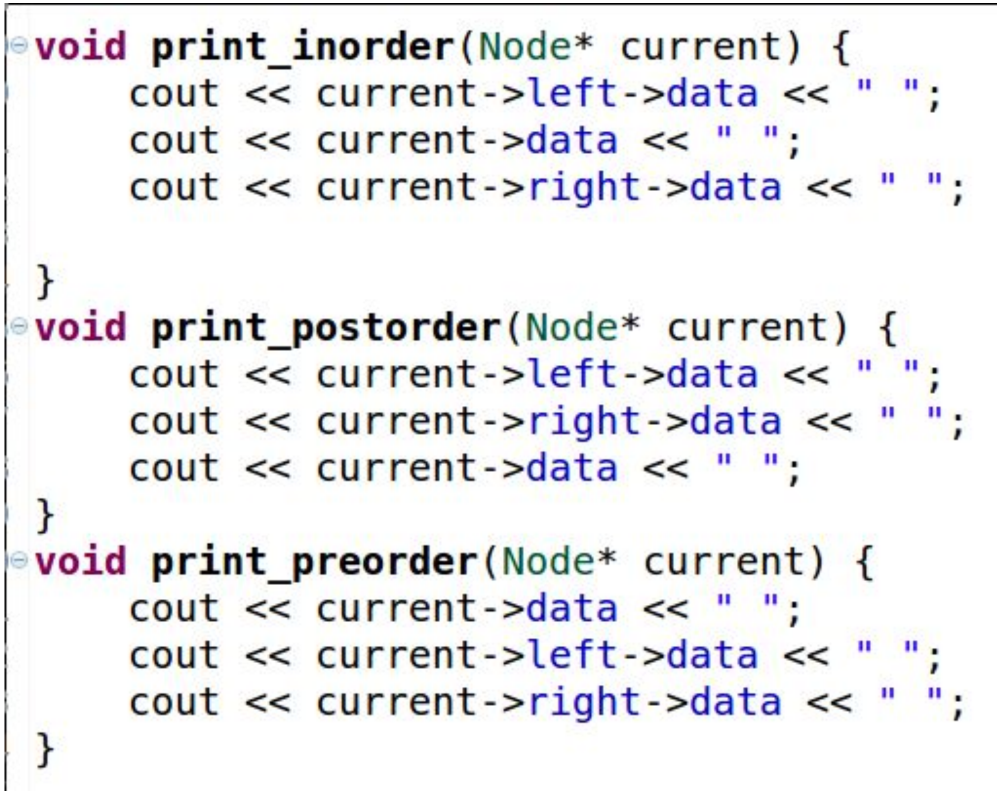




```
void print_inorder(Node* current) {
    cout << current->left->data << " ";
    cout << current->data << " ";
    cout << current->right->data << " ";
}
int main() {    // Create & Link Nodes
    Node* plus = new Node('+');
    Node* node2 = new Node('2');
    Node* node3 = new Node('3');
    plus->left = node2;
    plus->right = node3;
    print_inorder(plus);
```

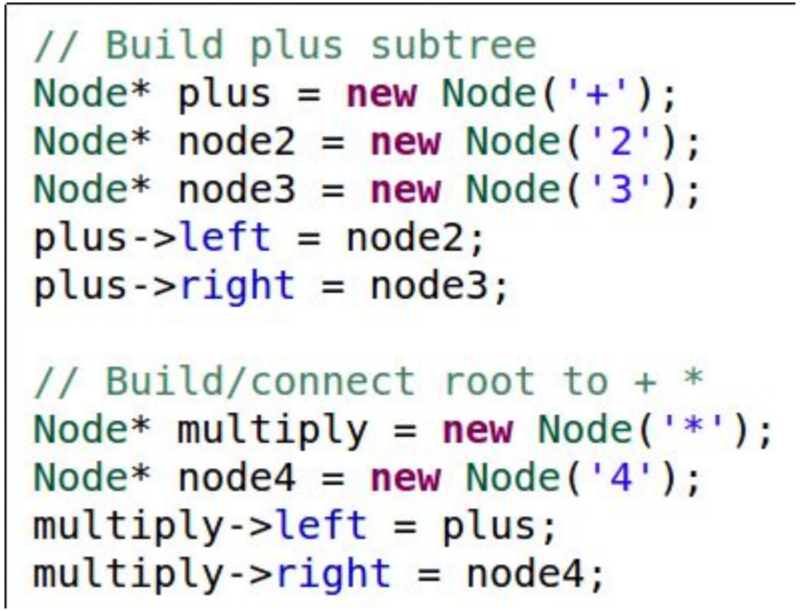
Print Expression Tree: 2 + 3

- Based on where we print the current node value we can get
  - 2 + 3 [in-order = infix]
  - 2 3 + [post-order = postfix]
  - - 2 3 [pre-order = prefix]
- We can summarize as
  - In-order = LVR
  - Post-order = LRV
  - Pre-order = VLR
- Other variants are not useful
  - LRV, RLV, VRL

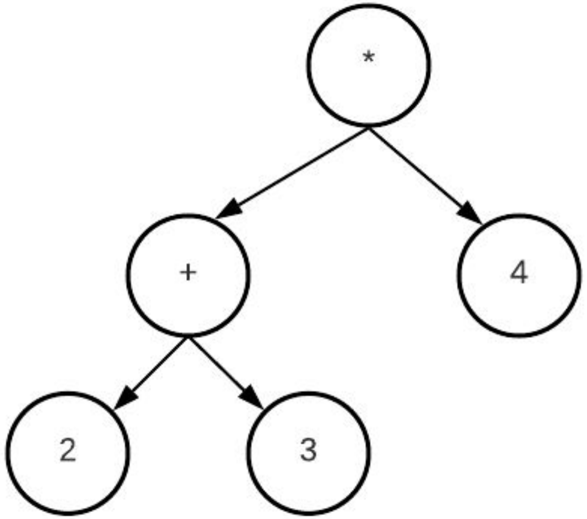


```
void print_inorder(Node* current) {
    cout << current->left->data << " ";
    cout << current->data << " ";
    cout << current->right->data << " ";

}
void print_postorder(Node* current) {
    cout << current->left->data << " ";
    cout << current->right->data << " ";
    cout << current->data << " ";
}
void print_preorder(Node* current) {
    cout << current->data << " ";
    cout << current->left->data << " ";
    cout << current->right->data << " ";
}
```

Print Expression Tree: (2 + 3) * 4

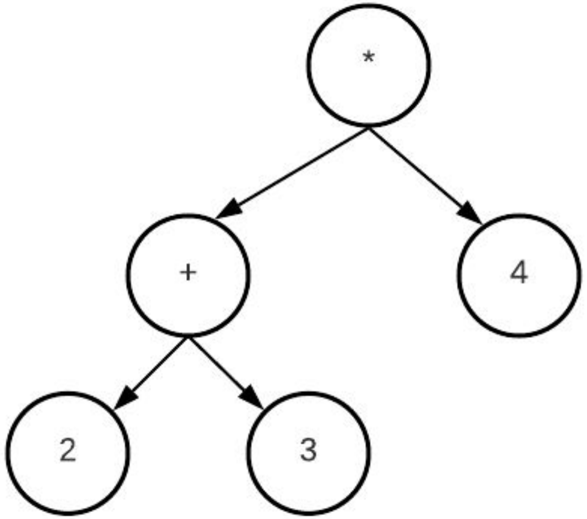


```
// Build plus subtree
Node* plus = new Node('+');
Node* node2 = new Node('2');
Node* node3 = new Node('3');
plus->left = node2;
plus->right = node3;

// Build/connect root to + *
Node* multiply = new Node('*');
Node* node4 = new Node('4');
multiply->left = plus;
multiply->right = node4;
```



Print Expression Tree: (2 + 3) * 4

- How can we print such a complex tree in **post-order**?
- We know the right subtree is 23+
- We need recursive thinking here!
- Instead of print left value, we need print **right sub-tree**



"Acquire knowledge and impart it to the people."

"Seek knowledge from the Cradle to the Grave."