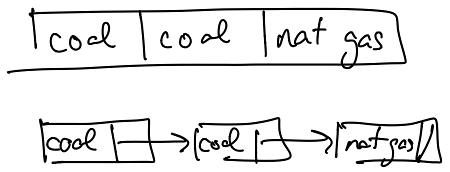
- Care about – the order of the cars on the train.

  How would I represent the order of the cars?

  $\underbrace{\text{List<String>}}_{\text{cargo}}$

  ⇑

  ArrayList/LL



- Suppose every train has all of the same kind of cargo.
- " I give each train a unique 3-digit ID.
- " I want to type in a train ID & get back the type of cargo.

  Map – Key – train ID (int)

  Value – string (type of cargo)

- Suppose I want to type in the cargo & get back the train ID.

  Map – Key – string (cargo)

  Value – train ID (int)

- What if there are multiple trains carrying the same cargo?

  Map – Key – string (cargo)

  Value – List <Integer> (multiple train IDs)

- Care about – order that the trains leave the station.

  $\underbrace{\text{List<Integer>}}$

  └ Train ID

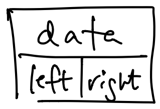
  What if we need to keep track of $\boxed{\text{times}}$

  Map : Key – train ID, Value – Time

  Map : Key – Time, Value – Train ID

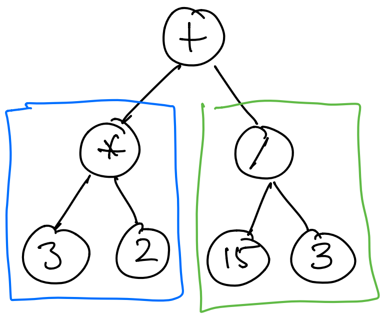
Linked node structure of a binary tree ← only has 0, 1, 2 children.

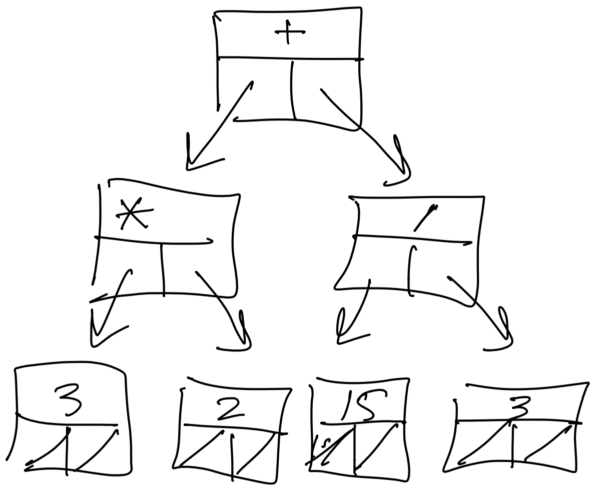
```
class Node {
    int/string/object data;
    Node left;
    Node right;
}
```



Expression Tree

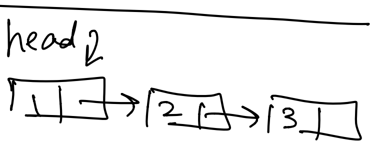
$(3 * 2) + (15 / 3)$





Recursion?

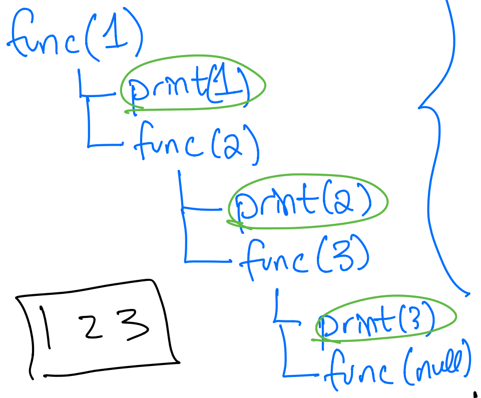
(Linked lists):



```
void func(LLNode n) {
    if (n == null)
        return
    else {
        print(n.data)
        func(n.next)
}
```

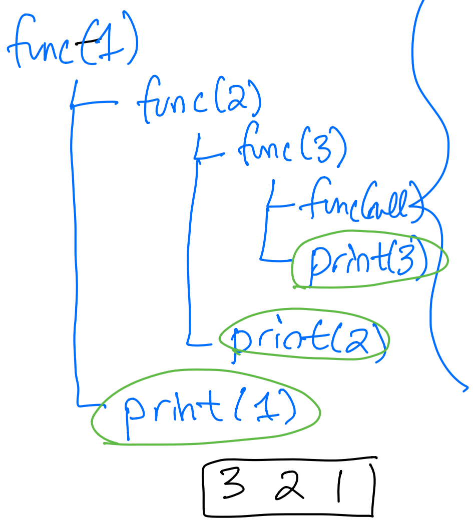
func(head)

OUTPUT: 1 2 3



```
void func(LLNode n) {
    if (n == null)
        return
    else {
        ~~print(n.data)~~    func(n.next)
        ~~func(n.next)~~     print(n.data)
}
```

OUTPUT



```
func(1)
 ├─ print(1)
 └─ func(2)
 :     ├─ print(2)
 :     └─ func(3)
 :           ├─ print(3)
 :           ├─ func(null)
 :           └─ print(3)
 :
 └─ print(1)
```

```
void func(LLNode n) {
    if (n == null)
        return
    else {
        print(n.data)
        func(n.next)
        print(n.data)
```

OUTPUT

1 2 3 3 2 1

Traverse a binary tree?

Preorder, Inorder, Postorder

Pre-order (binary tree)

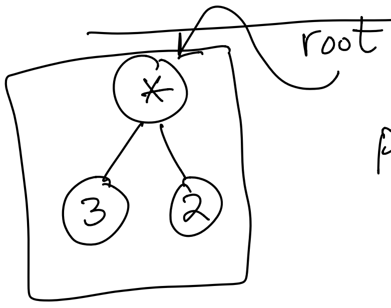
```
preorder(BTNode n)
{
    "visit" node n.data
    preorder(n.left)
    preorder(n.right)
}
```

Inorder

```
inorder(BTNode n)
{
    inorder(n.left)
    "visit" n.data
    inorder(n.right)
}
```

Postorder

```
postorder(BTNode n)
{
    postorder(n.left)
    postorder(n.right)
    "visit" data
}
```



Preorder → visit = print

```
preorder(*)
 ├─ print(*)
 └─ preorder(3)
 │     ├─ print(3)
 │     ├─ preorder(null)
 │     └─ preorder(null)
 └─ preorder(2)
```

$\boxed{*\ 3\ 2}$

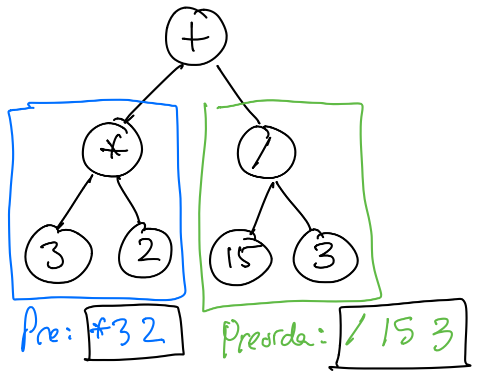
Inorder

$\boxed{3\ *\ 2}$

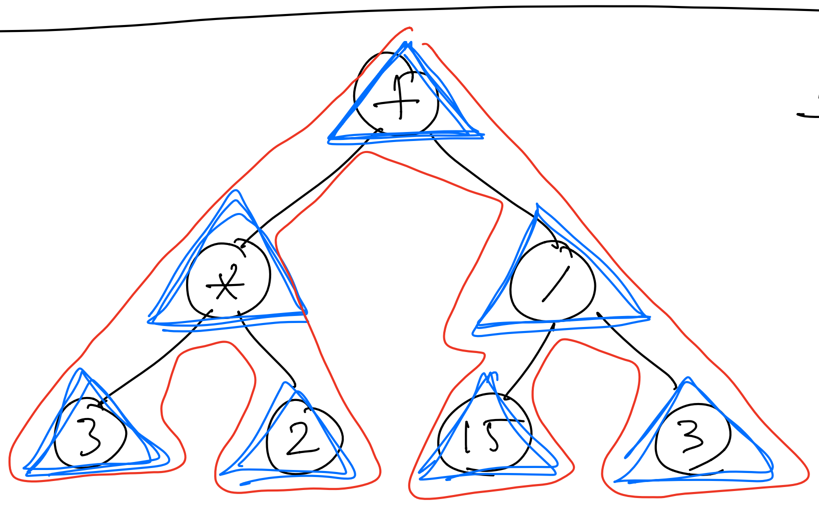
Postorder

$\boxed{3\ 2\ *}$

```
└ print(2)
  └ preorder(null)
   └ preorder(null)
```



Preorder: $\underset{\text{root}}{+}\ \underset{\text{left}}{\boxed{*\ 3\ 2}}\ \underset{\text{right}}{\boxed{/\ 15\ 3}}$



Follow the red line

- Preorder : visit the node when you pass the left side of $\triangle$
- Inorder : visit when you pass the bottom of $\triangle$
- Postorder – visit when you pass the right side of $\triangle$

Preorder: + * 3 2 / 15 3

Inorder: 3 * 2 + 15 / 3

Postorder: 3 2 * 15 3 / +