CS 225

Data Structures

February 8 – Trees Theory

G Carl Evans

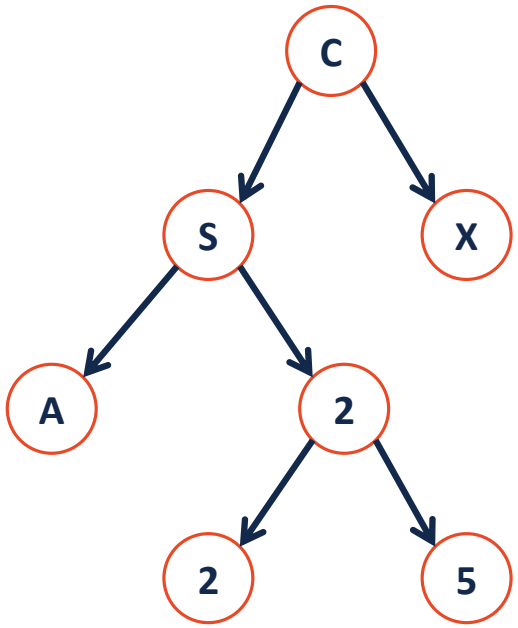
Binary Tree – Defined

A *binary tree* T is either:

- **$T = \emptyset$**

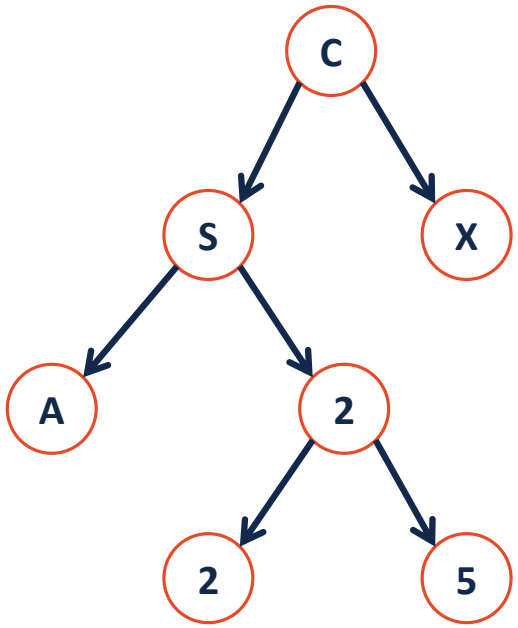
OR

- **$T = (r, T_L, T_R)$**



Tree Property: height

***height(T)*: length of the longest path from the root to a leaf**



Given a binary tree T:

$$height(T) = max(height(T_L),\ height(T_R)) + 1$$

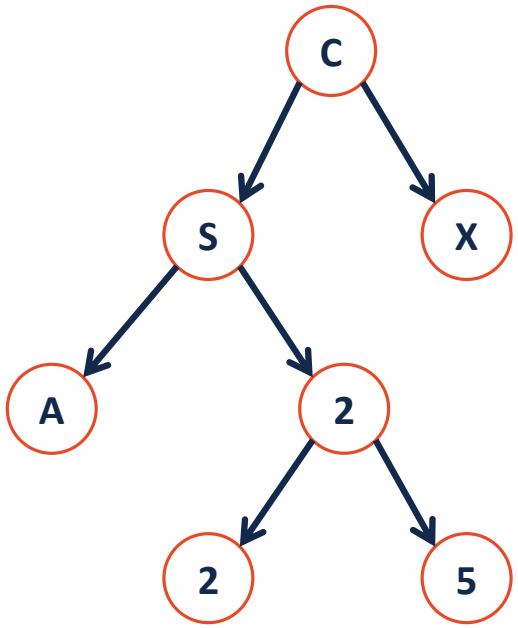
$$height(\emptyset) = -1$$

Tree Property: full

A tree ***F*** is **full** if and only if:

1.

2.



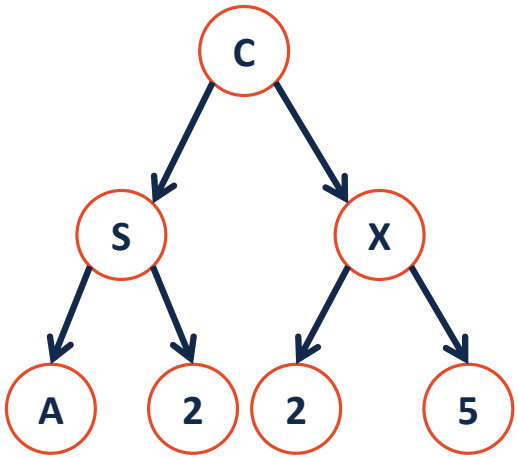
Tree Property: perfect

A **perfect** tree ***P*** is defined in terms of the tree's height.

Let $\mathbf{P_h}$ be a perfect tree of height **h**, and:

1.

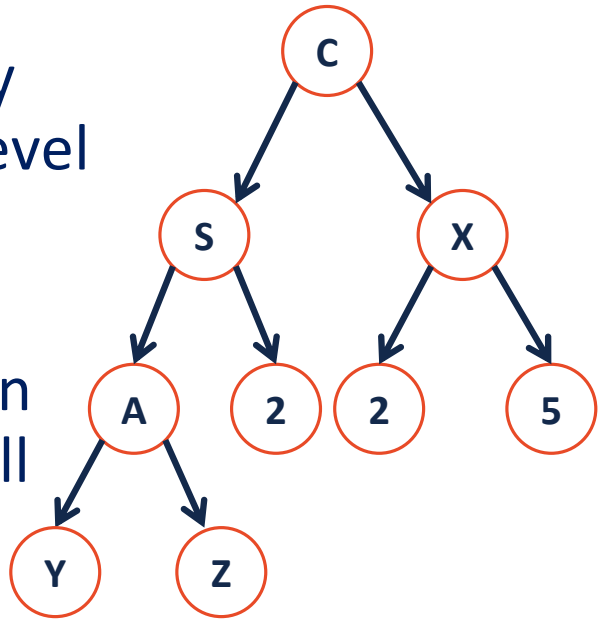
2.



Tree Property: complete

Conceptually: A perfect tree for every level except the last, where the last level if "pushed to the left".

Slightly more formal: For all levels k in [0, h-1], k has 2^k nodes. For level h, all nodes are "pushed to the left".



Tree Property: complete

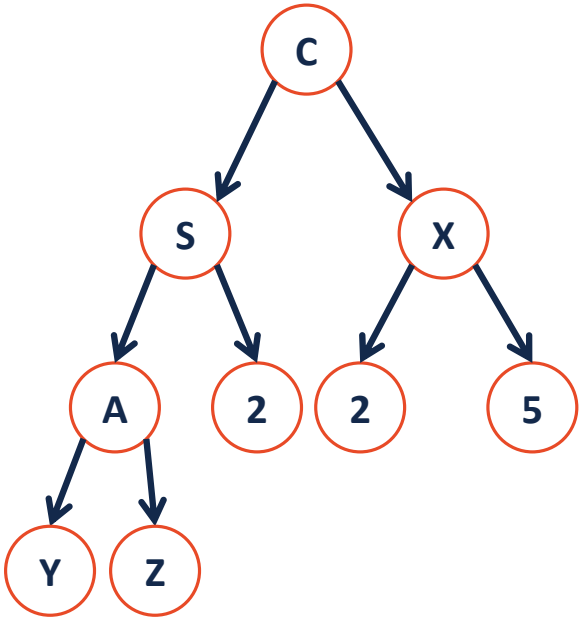
A **complete** tree ***C*** of height **h**, $\mathbf{C_h}$:

1. $\mathbf{C_{-1}} = \{\}$
2. $\mathbf{C_h}$ *(where h>0)* **= {r, $\mathbf{T_L}$, $\mathbf{T_R}$}** and either:

T_L is _____ and T_R is _____

OR

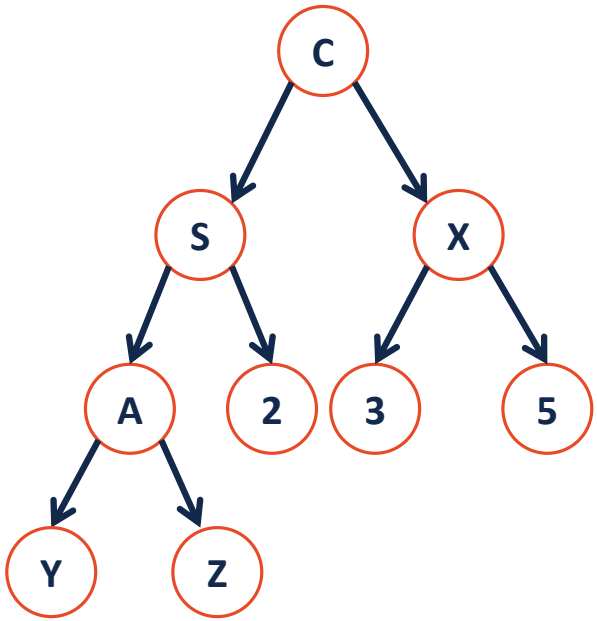
T_L is _____ and T_R is _____



Tree Property: complete

Is every **full** tree **complete**?

If every **complete** tree **full**?



Tree ADT

insert, inserts an element to the tree.

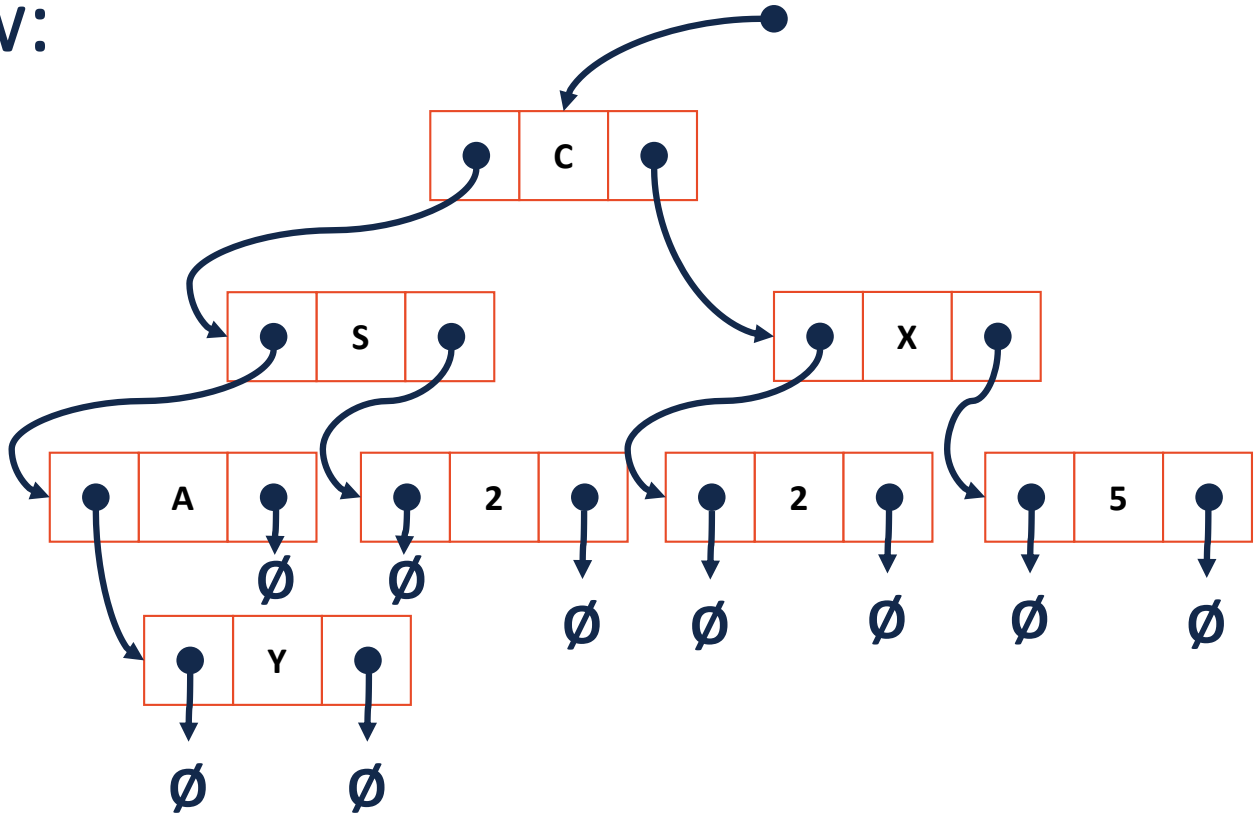
remove, removes an element from the tree.

access, access elements from the tree.

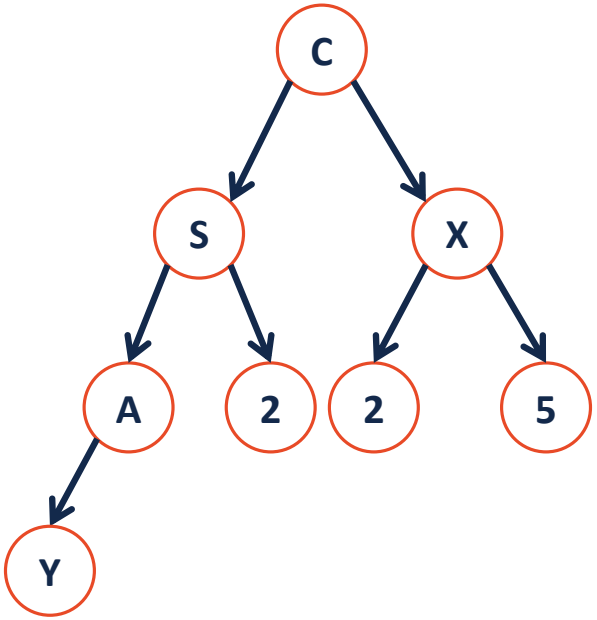
BinaryTree.h

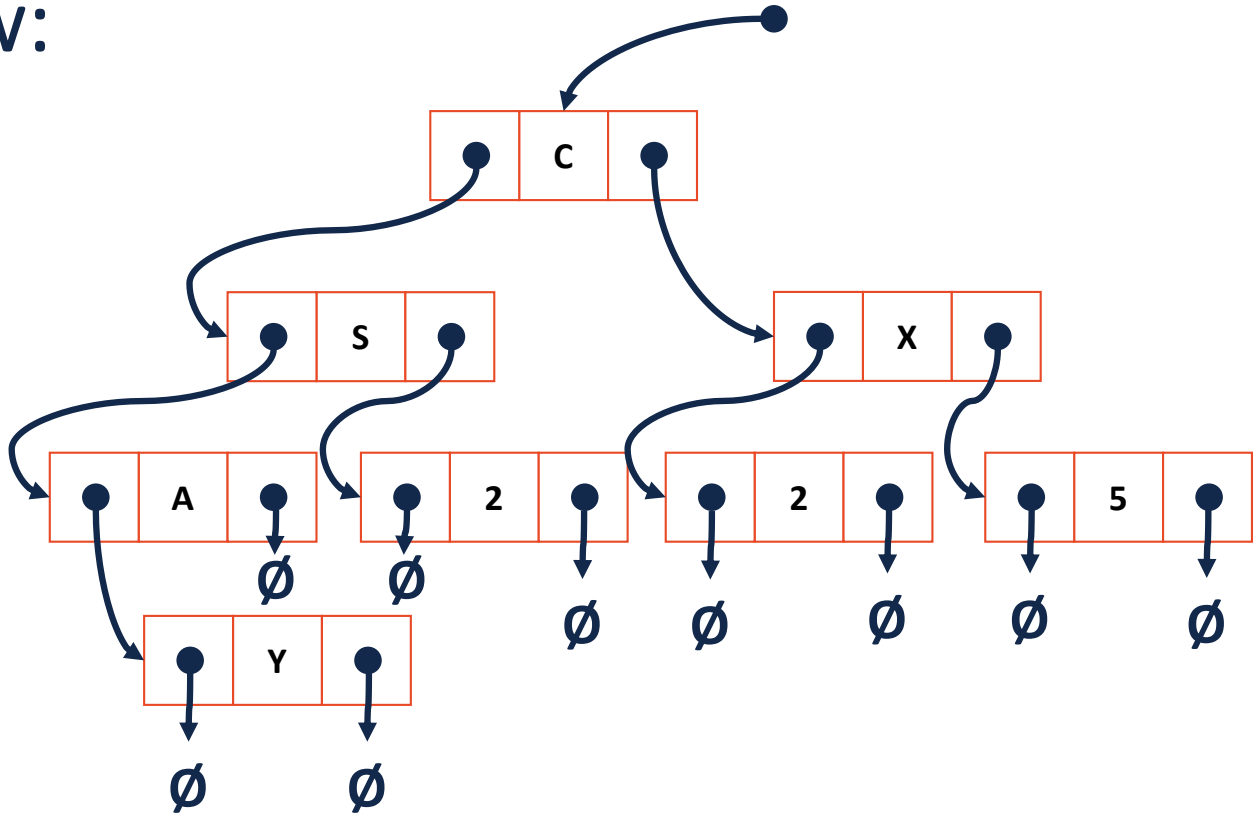
```
 1 #pragma once
 2
 3 template <class T>
 4 class BinaryTree {
 5   public:
 6     /* ... */
 7
 8   private:
 9
10
11
12
13
14
15
16
17
18
19 };
```

Trees aren’t new:



Trees aren't new:





How many NULLs?

Theorem: If there are **n** data items in our representation of a binary tree, then there are _____ NULL pointers.

How many NULLs?

Base Cases:

n = 0:

n = 1:

n = 2:

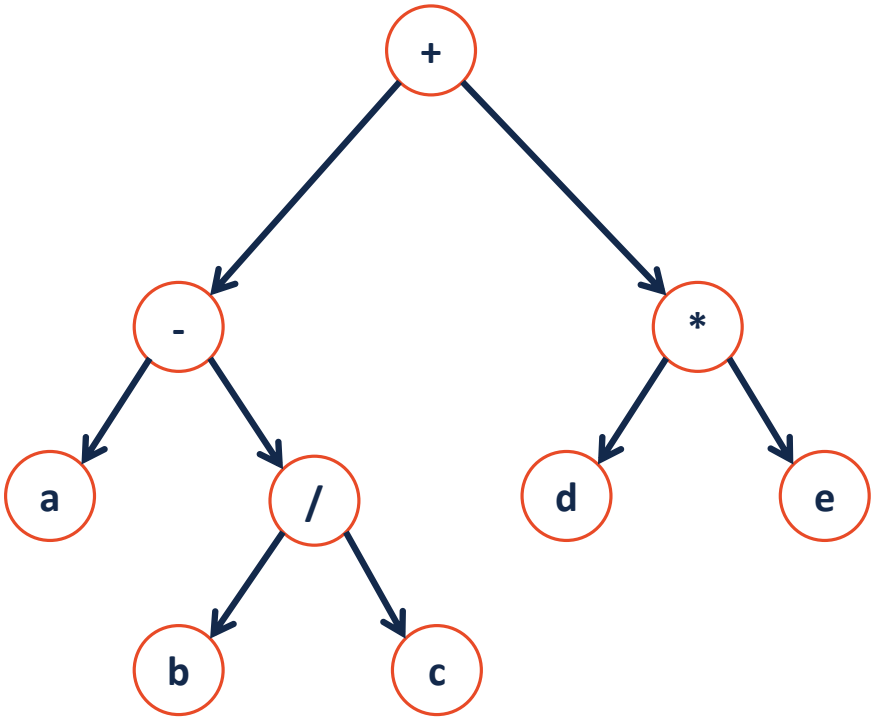
How many NULLs?

Induction Hypothesis:

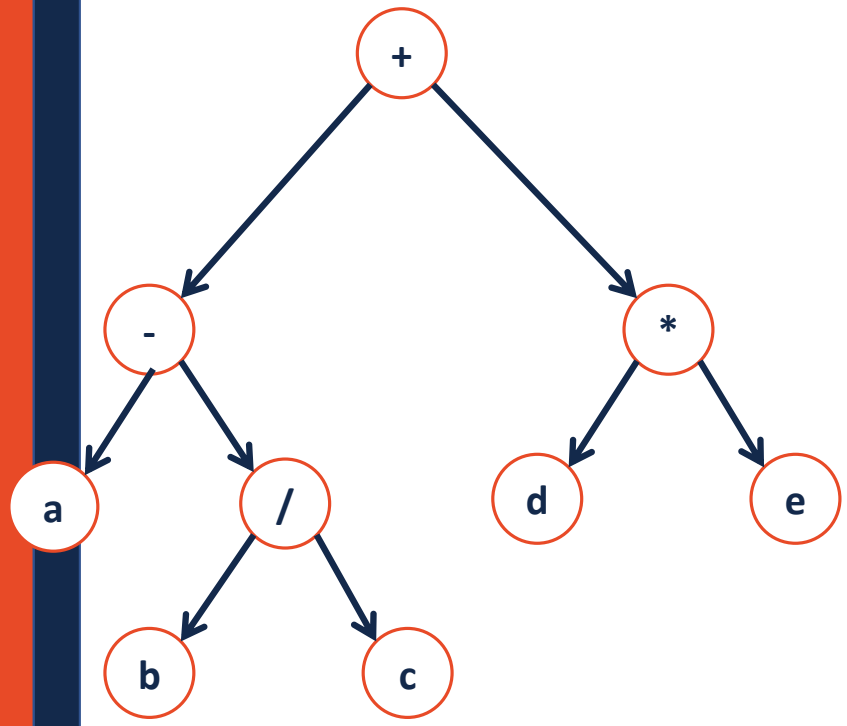
How many NULLs?

Consider an arbitrary tree **T** containing **n** data elements:

Access All the Nodes - Traversals

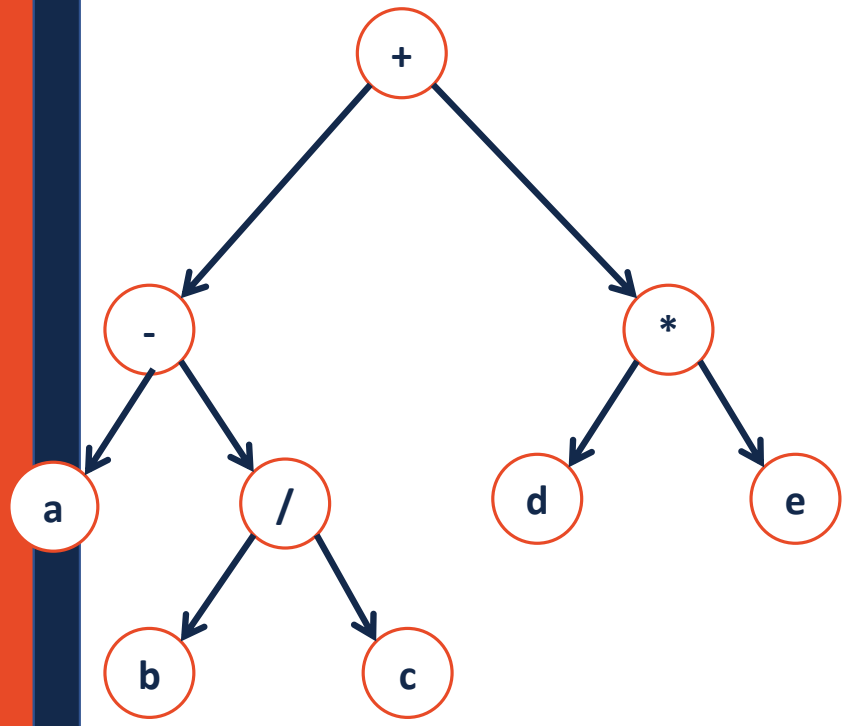


Traversals



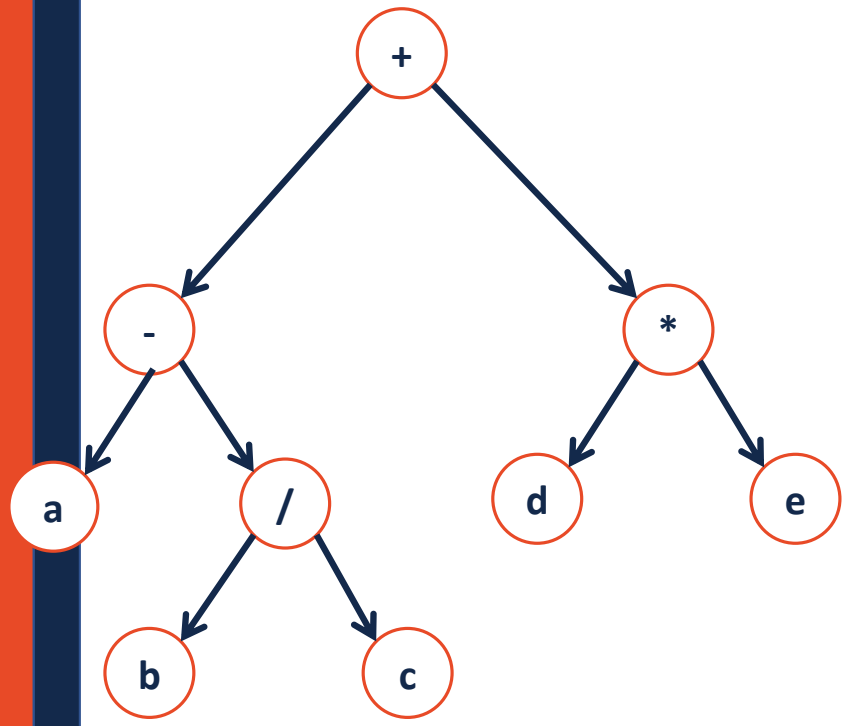

```
49  template<class T>
50  void BinaryTree<T>::__Order(TreeNode * cur)
51  {
52
53
54
55
56
57
58  }
```

Traversals



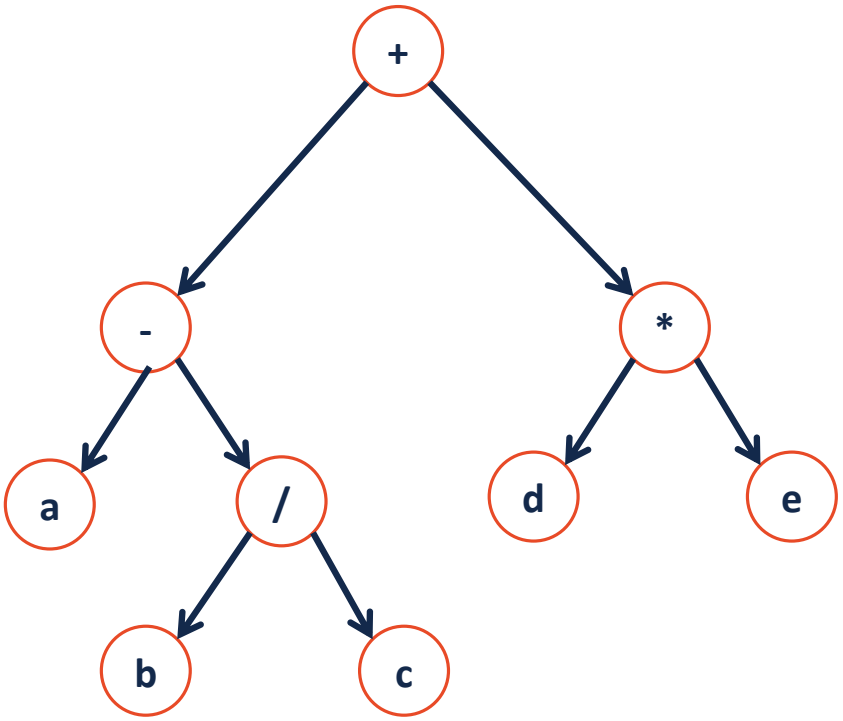

```
49  template<class T>
50  void BinaryTree<T>::___Order(TreeNode * cur) {
51     if (cur != NULL) {

52        ____________________________;

53        ___Order(cur->left);

54        ____________________________;

55        ___Order(cur->right);

56        ____________________________;

57     }
58  }
```

Traversals

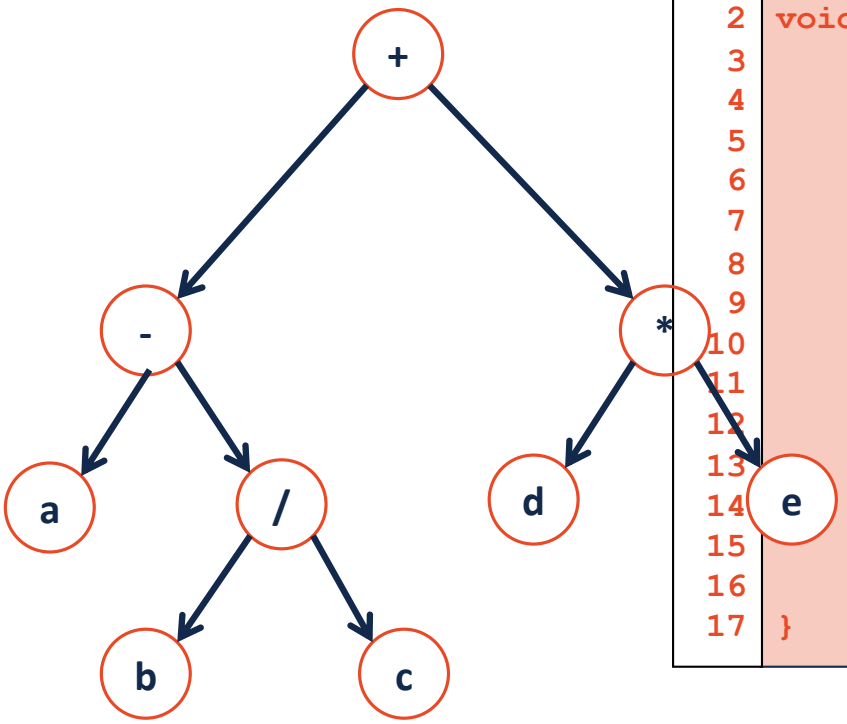



```
49  template<class T>
50  void BinaryTree<T>::___Order(TreeNode * cur) {
51     if (cur != NULL) {

52        ____________________________;

53        ___Order(cur->left);

54        ____________________________;

55        ___Order(cur->right);

56        ____________________________;

57     }
58  }
```

A Different Type of Traversal



A Different Type of Traversal




```
1  template<class T>
2  void BinaryTree<T>::levelOrder(TreeNode * root) {
3
4
5
6
7
8
9
10
11
12
13
14
15
16
17 }
```

Traversal vs. Search

Traversal

Search

Search: Breadth First vs. Depth First

Strategy: Breadth First Search (BFS)

Strategy: Depth First Search (DFS)