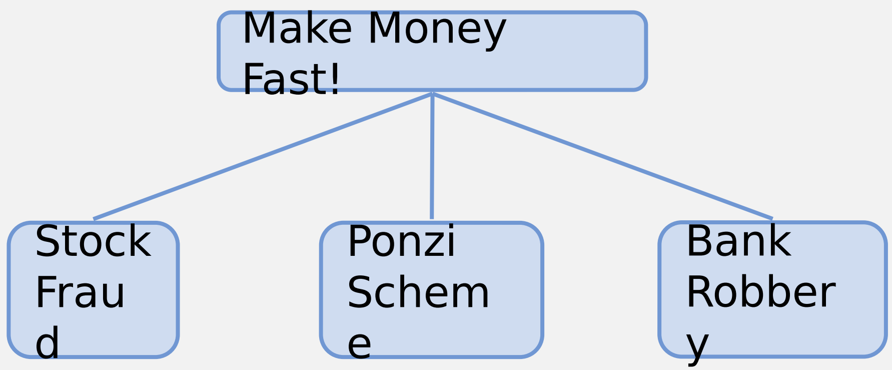
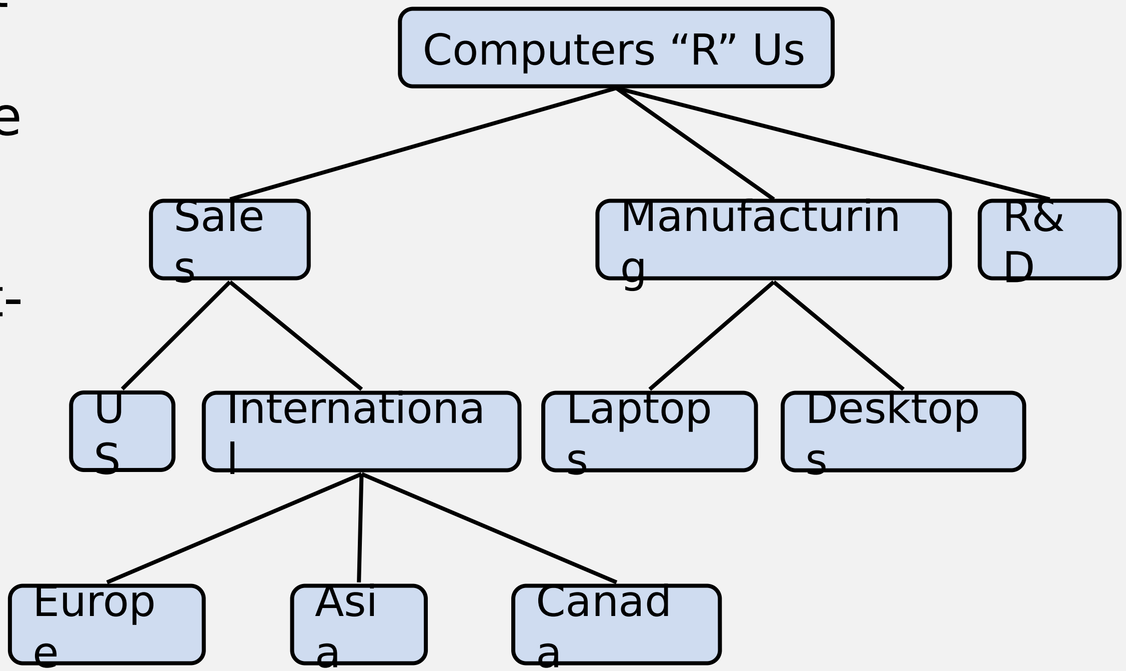
CPSC 131

Trees



What is a Tree?

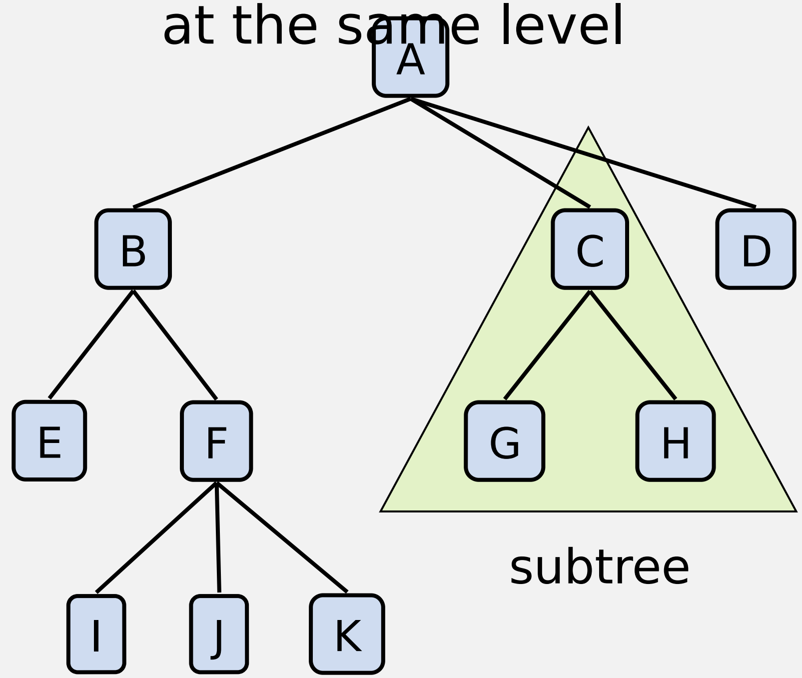
- A ***tree*** is an abstract model of a hierarchical structure
- A tree consists of ***nodes*** with a parent-child relation
- Applications:
  - Organization charts
  - File systems
  - Programming environments



Tree Terminology

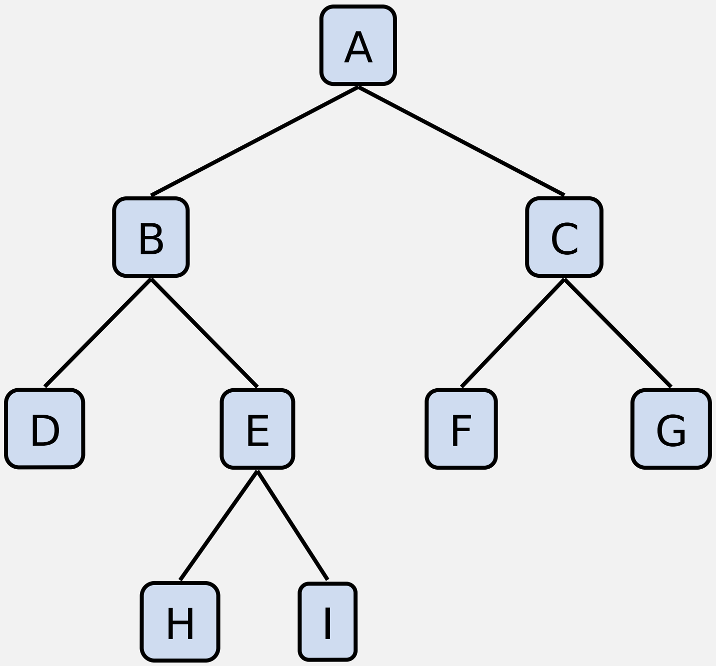
- Root: node without parent (A)
- Internal node: node with at least one child (A, B, C, F)
- External node (a.k.a. leaf): node without children (E, I, J, K, G, H, D)
- Ancestors of a node: parent, grandparent, grand-grandparent, etc.
- Depth of a node: number of ancestors (between the node and the root. Root has depth 0)
- Height of a tree: maximum depth

- Subtree: tree consisting of a node and its descendants
- Sibling: Nodes that share a parent; nodes at the same level

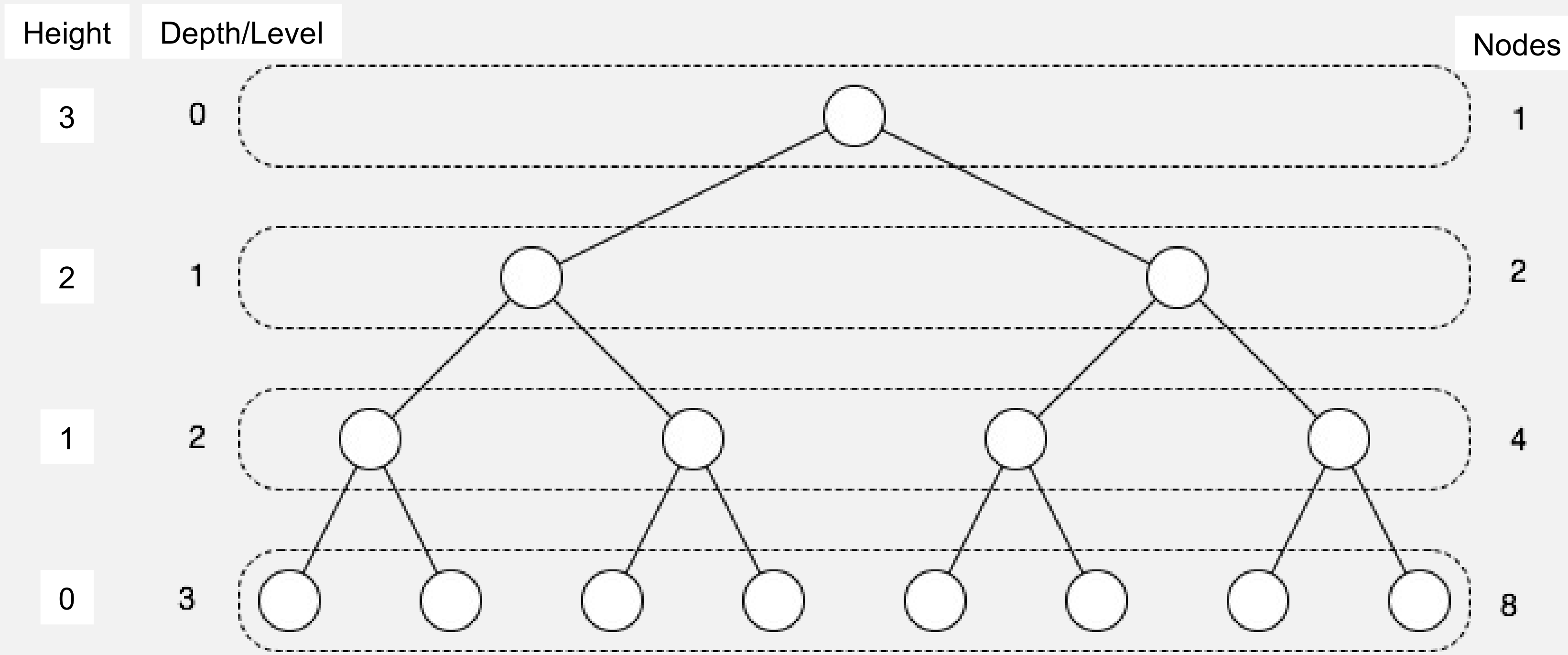


Binary Trees

- A binary tree is a tree with the following properties:
  - Each internal node has at most two children
  - The children of a node are an ordered pair
- We call the children of an internal node left child and right child
- Types of Binary Trees
  - Full: if every node has 0 or 2 children
  - Complete: all levels are full except possibly the last level
  - Perfect: all internal nodes have 2 children and leaf nodes are at the same level
- Applications:
  - arithmetic expressions
  - decision processes
  - searching



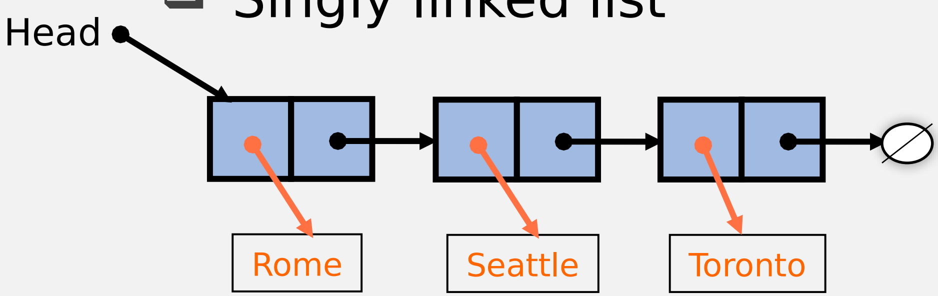
Properties of Binary Trees



nodes = n = 15;
h = height = 3
$h = \text{floor}(\ \log_2(n)\ ) = \text{floor}(\ \log_2(15)\ ) = \text{floor}(\ 3.9\ ) = 3$

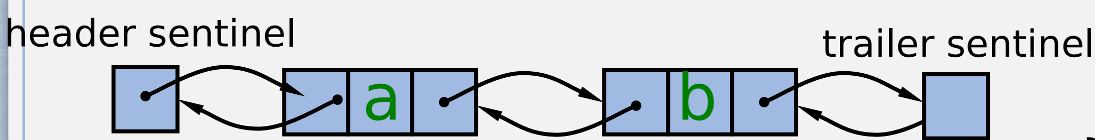
Defining a Node in C++

- Singly linked list



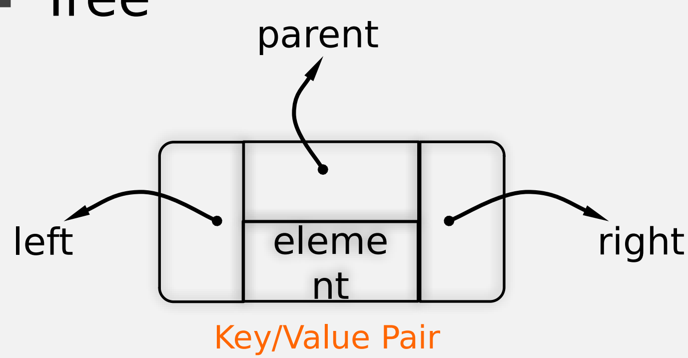

```
template <typename ELT>
class Node {
    ELT element;
    Node *next;
}
```

- Doubly linked list




```
template <typename ELT>
class Node {
    ELT element;
    Node *next;
    Node *prev;
}
```

- Tree




```
template <typename ELT>
class Node {
    ELT element;
    Node *left;
    Node *right;
    Node *parent;
}
```

Tree Traversal

- A traversal “visits” the nodes of a tree in a systematic manner
- Three variants
  1. Preorder
  2. Inorder
  3. Postorder
- Easiest to define these using recursion

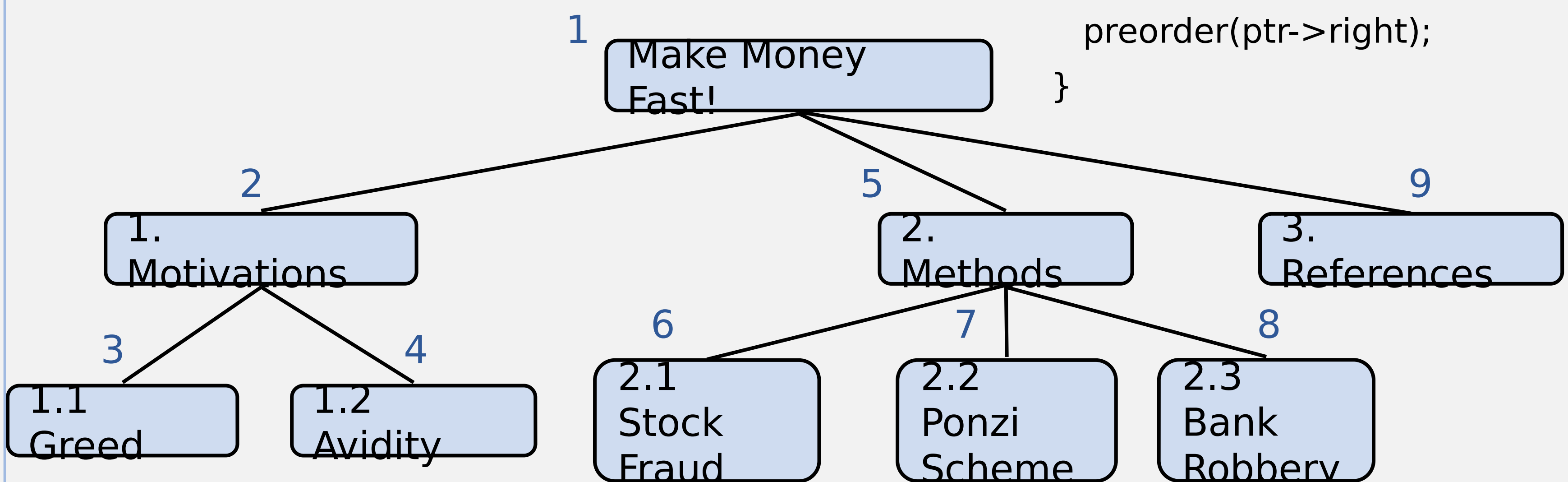
Preorder Traversal

- In a preorder traversal, a node is visited before its descendants
- Application: print a structured document

Algorithm ***preOrder***(***v***)
    visit(***v***)
    for each child ***w*** of ***v***
        preorder (***w***)

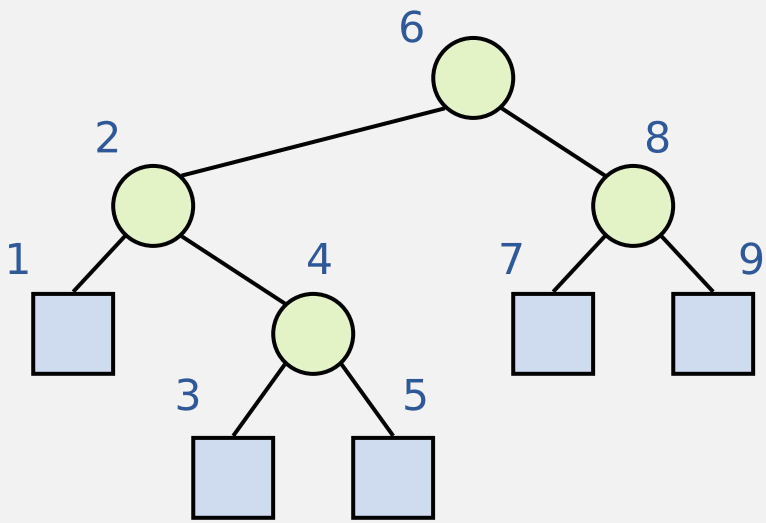
Code for a binary tree

```
void preorder (Node *ptr) {
  if (ptr == nullptr) return;
  cout << ptr->element;  // the "visit"
  preorder(ptr->left);
  preorder(ptr->right);
}
```



Inorder Traversal

- In an inorder traversal a node is visited after its left subtree and before its right subtree
- Application: draw a binary tree
  - x(v) = inorder rank of v
  - y(v) = depth of v




```
Algorithm inOrder(v)
    if ¬ v.isExternal()
        inOrder(v.left())
    visit(v)
    if ¬ v.isExternal()
        inOrder(v.right())
```

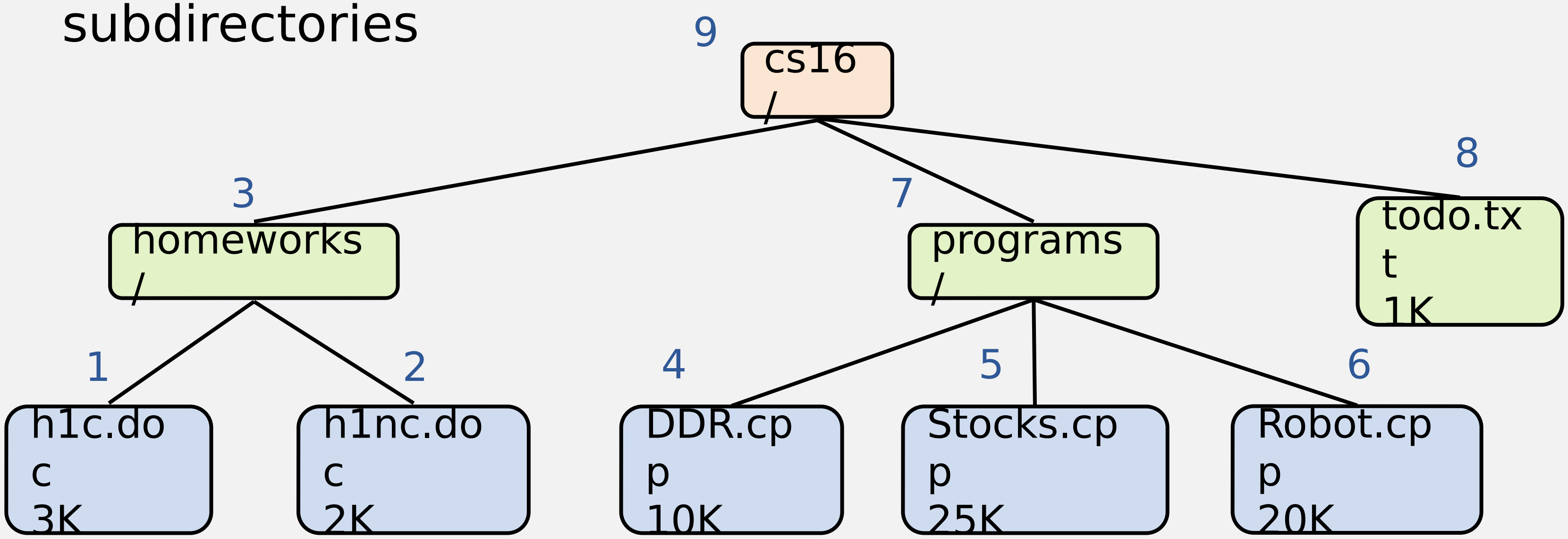
Code for a binary tree

```
void inorder (Node *ptr) {
  if (ptr == nullptr) return;
    inorder(ptr->left);
    cout << ptr->element;    // the "visit"
    inorder(ptr->right);
}
```

Postorder Traversal

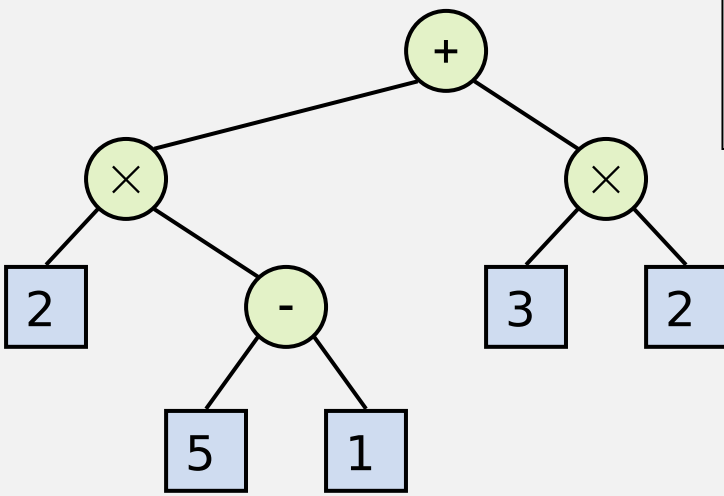
- In a postorder traversal, a node is visited after its descendants
- Application: compute space used by files in a directory and its subdirectories

Algorithm ***postOrder***(***v***)
    for each child ***w*** of ***v***
        postOrder (***w***)
    visit(***v***)



Evaluate Arithmetic Expressions

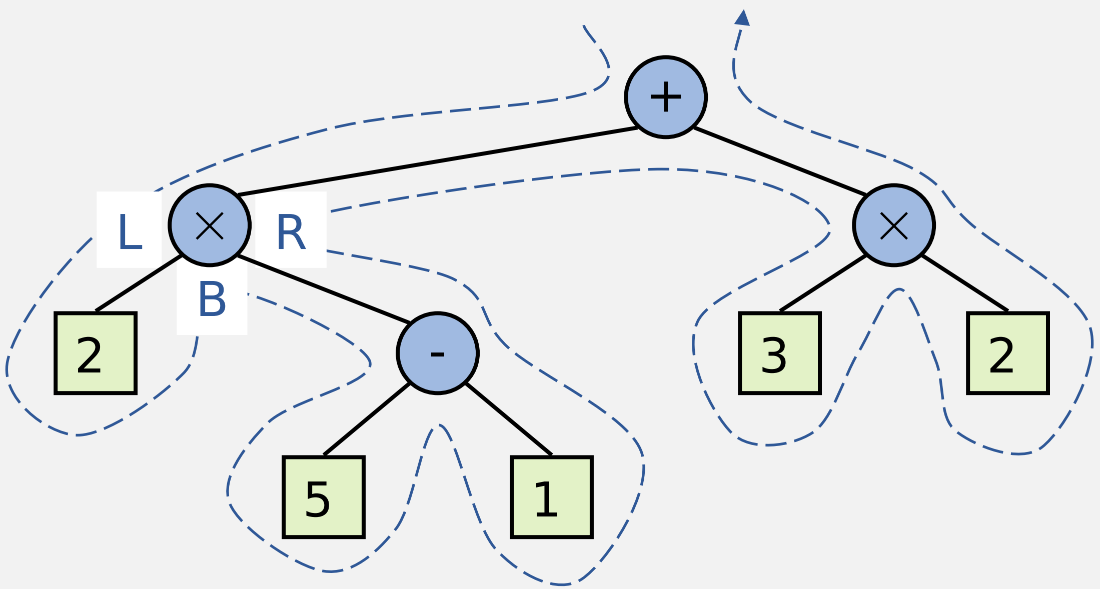
- Specialization of a postorder traversal
  - recursive method returning the value of a subtree
  - when visiting an internal node, combine the values of the subtrees



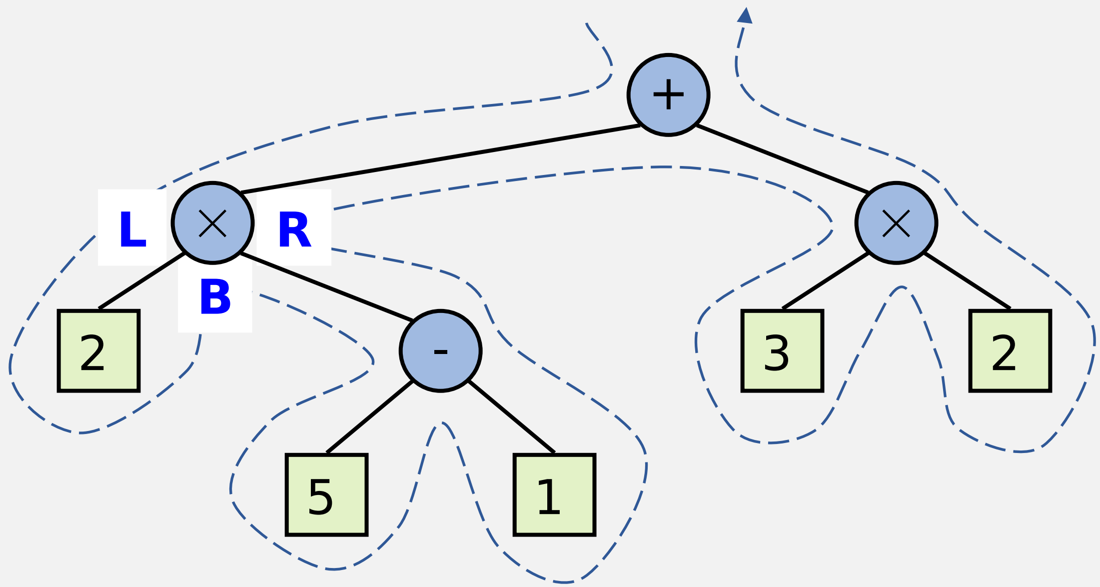
Algorithm ***evalExpr***(***v***)
    if ***v.isExternal***()
        return ***v.element***()
    else
        x ← ***evalExpr***(***v.left***())
        y ← ***evalExpr***(***v.right***())
        ◊ ← operator stored at ***v***
        return ***x*** ◊ ***y***

Euler Tour Traversal Technique

- Generic traversal of a binary tree
- Includes as special cases the preorder, postorder and inorder traversals
- Walk around the tree and visit each node three times:
  - preorder: visit on the left side
  - inorder: visit from below (between the children)
  - postorder: visit on the right side



Euler Tour Traversal Technique (cont)

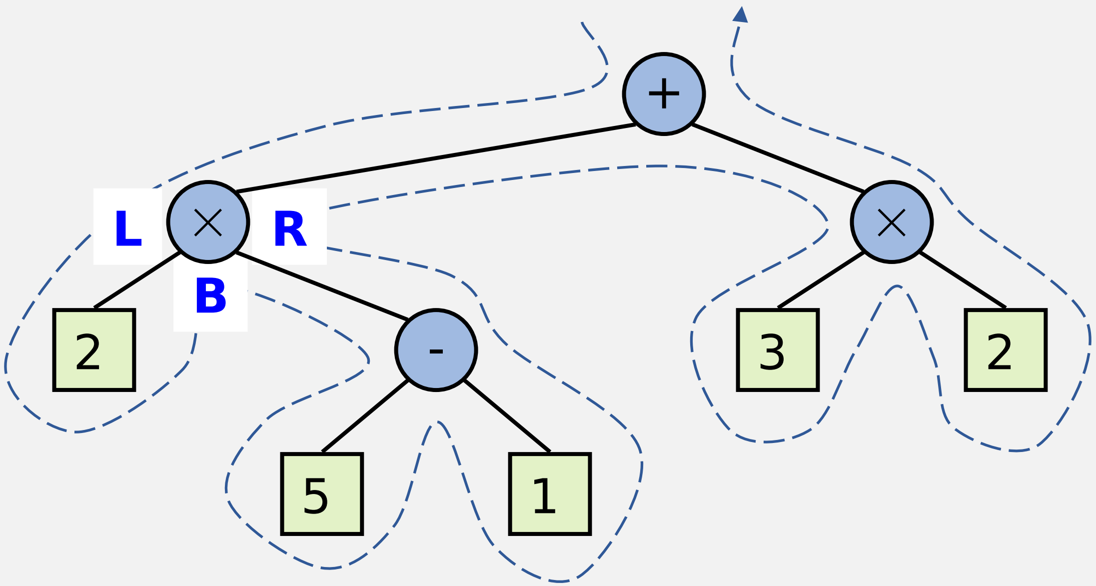


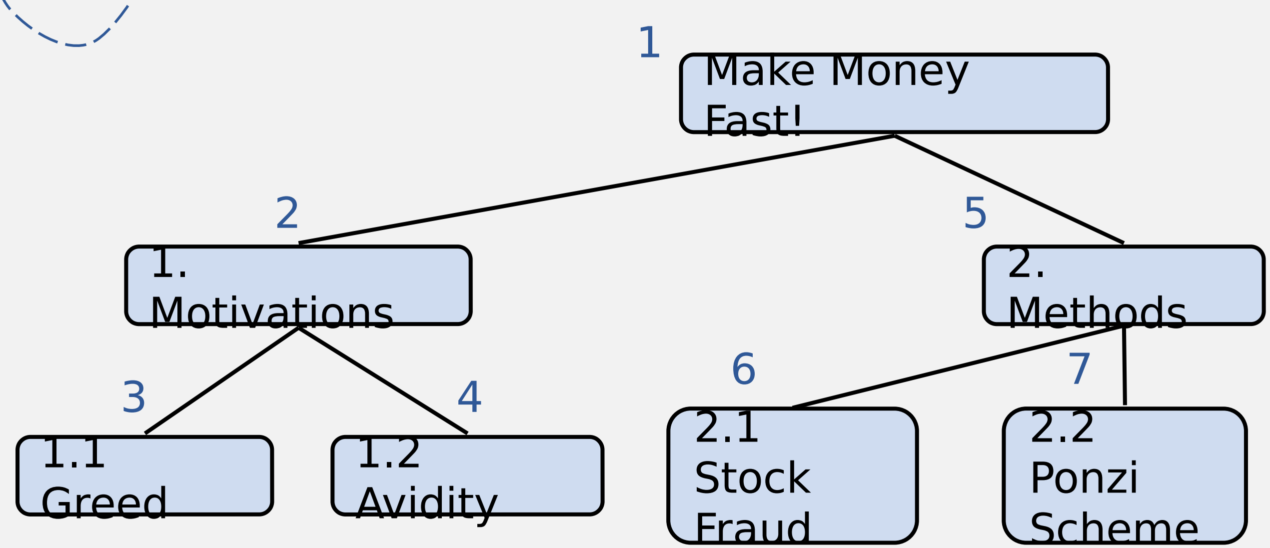
preorder(node):
if node == NULL
return
visit(node)
preorder(node->left)
preorder(node->right)

inorder(node):
if node == NULL
return
inorder(node->left)
visit(node)
inorder(node->right)

postorder(node):
if node == NULL
return
postorder(node->left)
postorder(node->right)
visit(node)

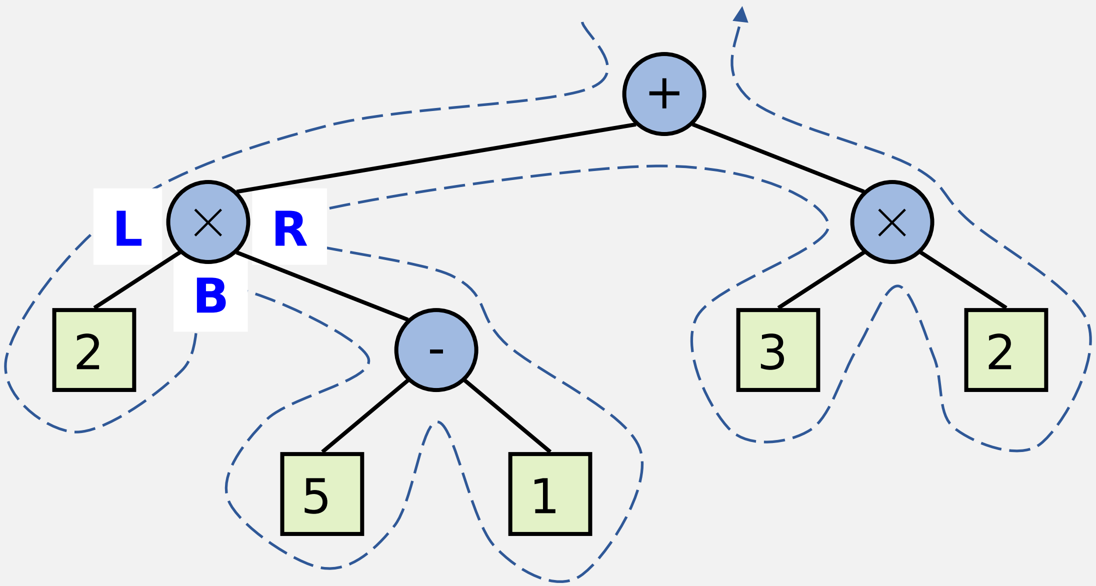
Preorder Traversal

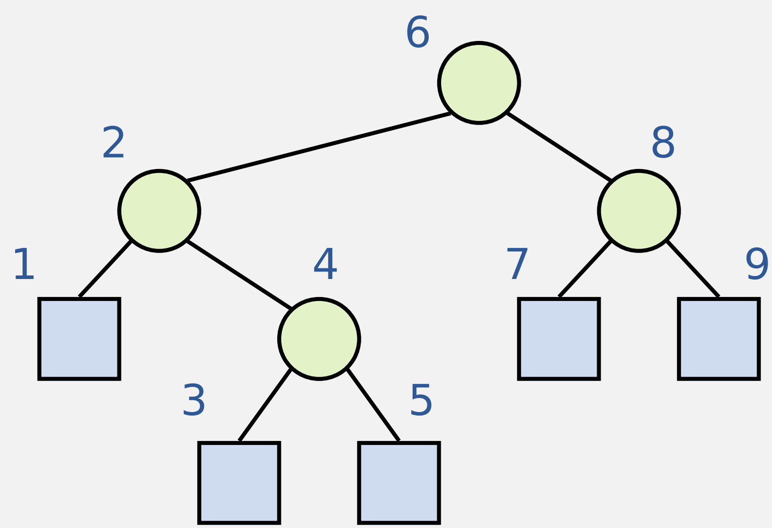




preorder(node):
if node == NULL
return
visit(node)
preorder(node->left)
preorder(node->right)

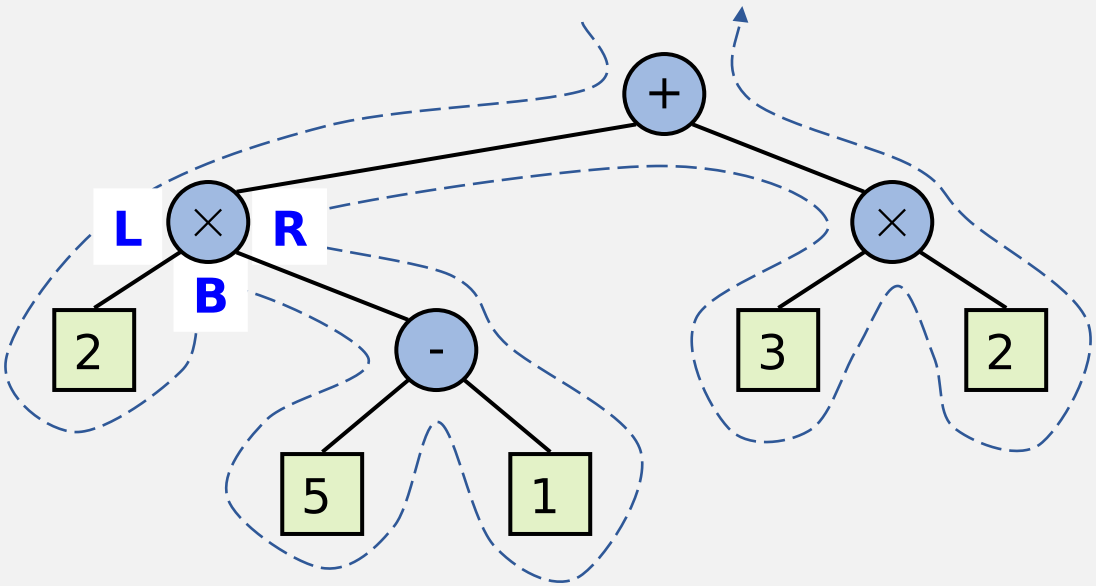
Inorder Traversal

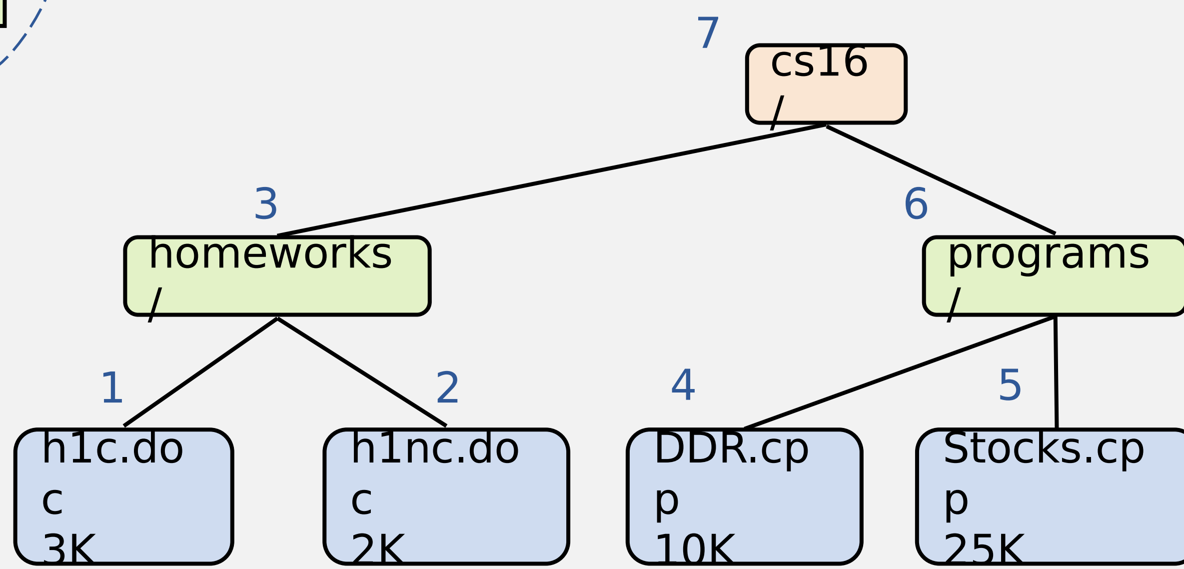




inorder(node):
if node == NULL
return
inorder(node->left)
visit(node)
inorder(node->right)

Postorder Traversal





postorder(node):
if node == NULL
return
postorder(node->left)
postorder(node->right)
visit(node)

References

- Data Structures and Algorithms in C++, 2nd Edition by Goodrich, Tamassia, and Mount
- Section: 7.3