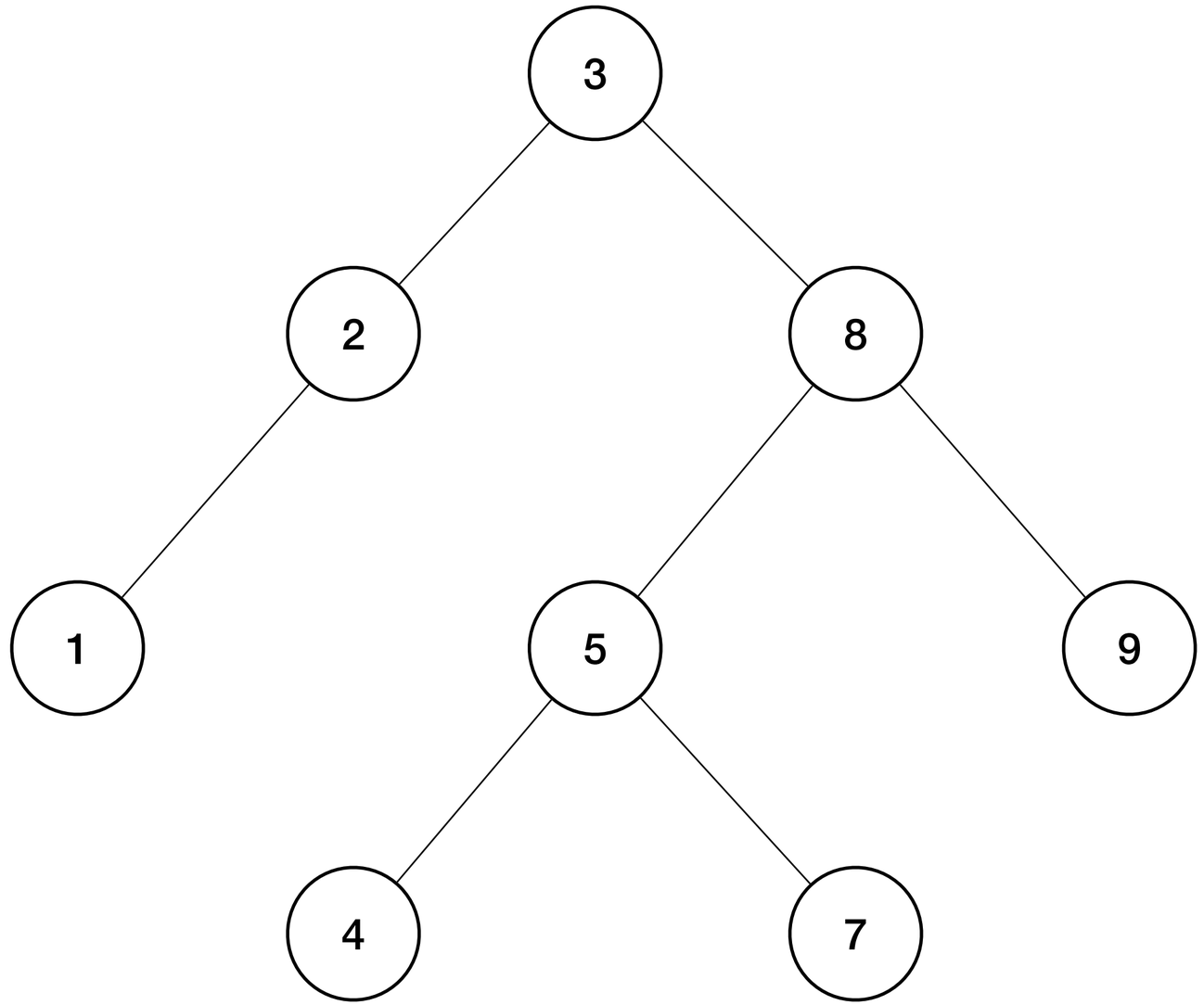
# ECE 220

## Lecture x0018
## BSTs and C++ examples

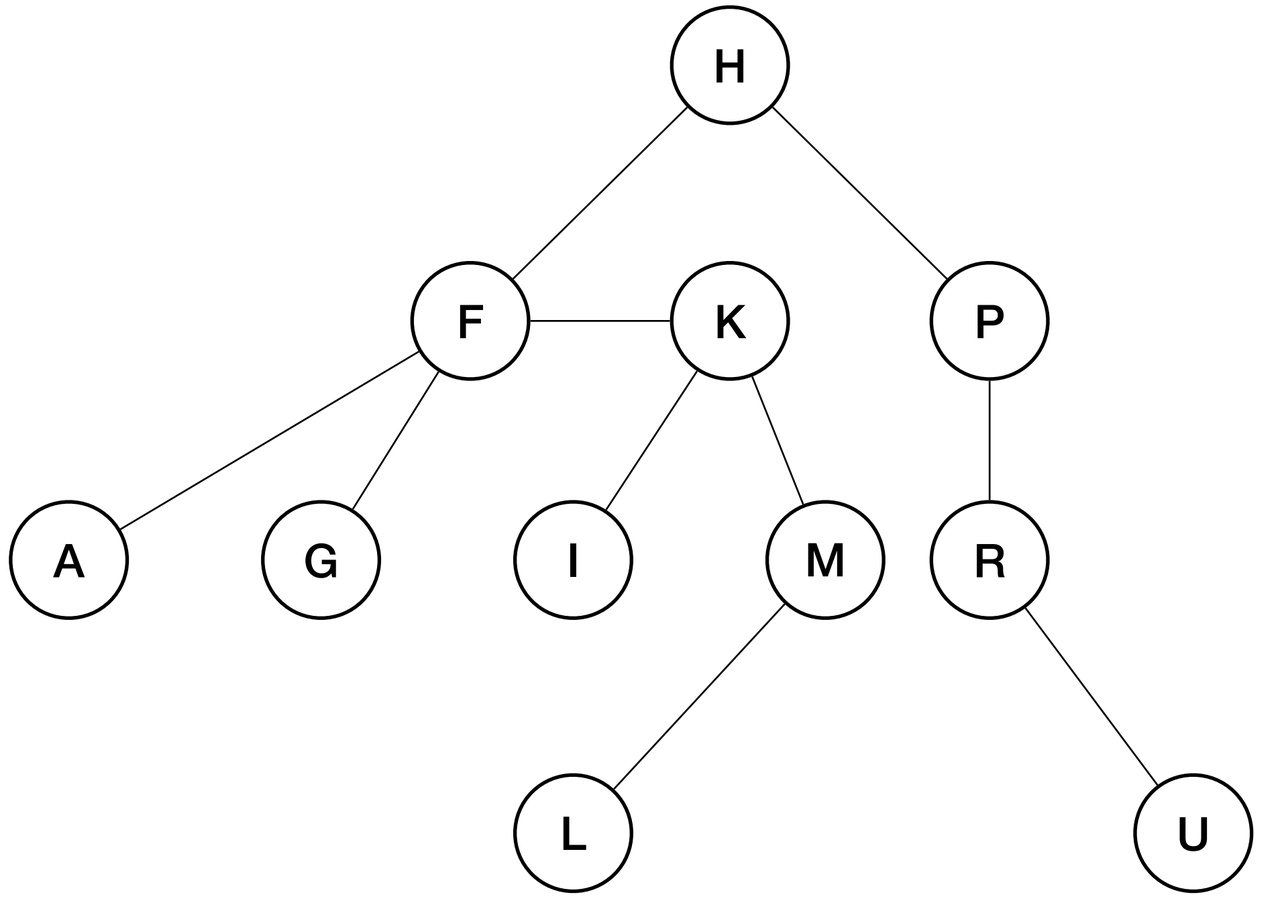
# Binary Search Trees

- Binary trees that have a particular *sorted* property are called **binary search trees** (BST)
  - All nodes in the **left subtree** of a given node are *lesser than or equal* to the node
  - All nodes in the **right subtree** of a given node are *greater* than that node



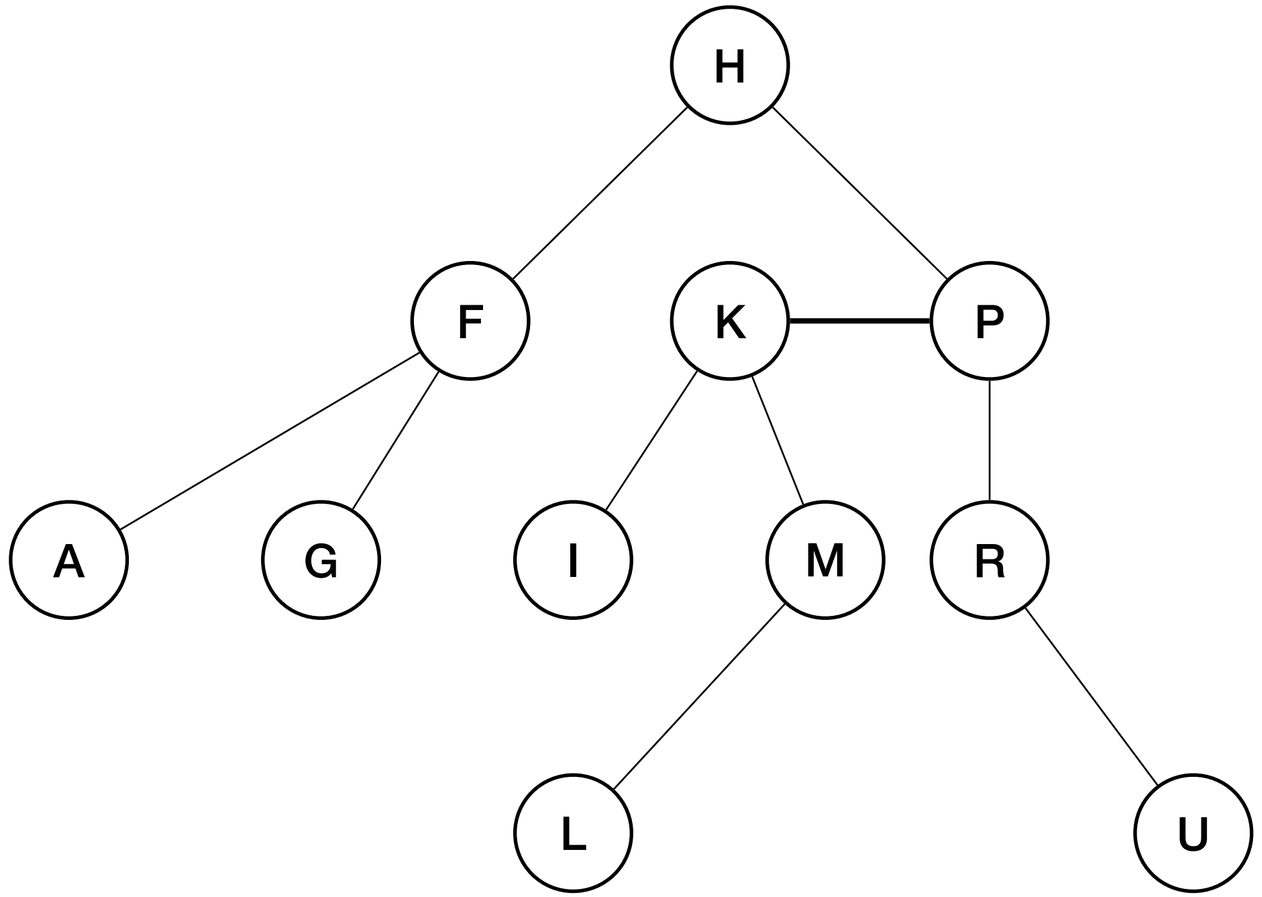
# Concept check

- Who are the siblings of R?
- What is the depth of node I?
- List the leaf nodes?
- What is the height of the tree?
- Is this a Binary Search Tree?



# Concept check

- Who are the siblings of R?
- What is the depth of node I?
- List the leaf nodes?
- What is the height of the tree?
- Is this a Binary Search Tree?



```
template <typename T>
struct treenode{
  T data;
  treenode *left;
  treenode *right;
};
```

# C++ Example

- Using classes in C++, create a BST class and perform or find:
  - Insertion
  - Searching
  - Traversal
  - Vectorization
  - Size of tree (# of nodes)
  - Find height of the tree
  - Deletion of tree

```
template <class N>
class bst{
private:
  …
  …
  …
public:
  bst();
  void insert(N data);
  treenode<N> *search(N data);
  void inorder();
  vector<N> vectorize();
  int node_count();
  int height();
  void print();
  ~bst();
};
```

```
template <typename T>
struct treenode{
  T data;
  treenode *left;
  treenode *right;
};
```

# C++ Example

- Using classes in C++, create a BST class and perform or find:
  - Insertion
  - Searching
  - Traversal
  - Vectorization
  - Size of tree (# of nodes)
  - Find height of the tree
  - Deletion of tree

```
template <class N>
class bst{
private:
  typedef treenode<N> node;
  node *root;
  void insert(N data, node **cursor);
  node *search(N key, node *cursor);
  void inorder(node *cursor);
  vector<N> vectorize(node *cursor, vector<N> &v);
  int countnodes(node *cursor);
  void print(node *cursor, int depth);
  int height(node *cursor);
  void delete_tree(node *cursor);

public:
  …
  …
```

# Structs vs. Classes

- In C++, the **only** difference between classes and structs is the default access permissions for members:
  - Classes — default is private
  - Structs — default is public
- And default inheritance type:
  - Classes — private inheritance by default
  - Structs — public inheritance by default

# No really …

```
template <typename T>
struct treenode{
  T data;
  treenode *left;
  treenode *right;
};
```

Useful if **T** ends up being a user defined type (class or struct!)

```
template <typename T> struct treenode {
  T data;
  treenode *left;
  treenode *right;

  // Yes this is a constructor !
  // Part after ':' calls data's constructor
  treenode(T data) : data(data) {
    this->data = T(data);

    // Initialize children to nullptr
    this->left = nullptr;
    this->right = nullptr;
  }
};
```

# Implementation time!

```
#include <iostream>
#include "bst.hpp"

using namespace std;

int main(){
  bst <int> tree1;
  cout<<"Building a Binary Search Tree"<<endl;

  tree1.insert(45);
  tree1.insert(50);
  tree1.insert(35);
  tree1.insert(30);
  tree1.insert(70);
  tree1.insert(20);
  tree1.insert(40);
  tree1.insert(80);
  tree1.insert(60);

  cout<<"Total number of nodes in this tree: ";
  cout<<tree1.node_count()<<endl;
  tree1.inorder();
  cout<<endl;
  tree1.print();
  cout<<"The tree height is: "<<tree1.height();
  cout<<endl;

  vector <int> v = tree1.vectorize();
  cout<<"Vectorized in order this is:"<<endl;
  for (auto it= v.begin(); it != v.end(); ++it)
    cout<<*it<<", ";
  return 0;
}
```

# More practice? Okey doke!

- Practicing on linked lists:
  - Stanford's Linked List Question Bank
- Practicing on trees:
  - Stanford's Binary Tree Question Bank (jump to section 2)
- Advanced material: Trees List Recursion

```
template <typename T>
struct treenode{
  T data;
  treenode *left;
  treenode *right;
};
```

# Some food for thought

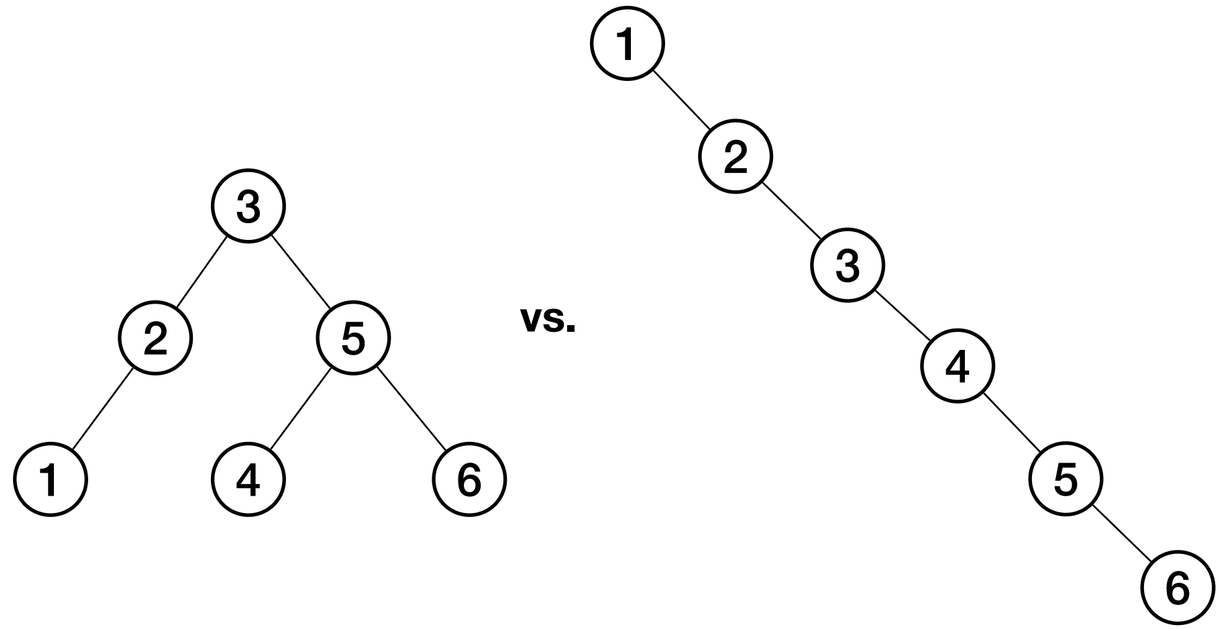
- Can you write a function to determine if a binary tree is a valid BST?

- Can you reconstruct a BST from its vectorized version?

  - *Traversals matter*: pre-order, in-order, post-order

- How to modify struct definition if tree is no longer binary?

Inorder: 1,2,3,4,5,6,

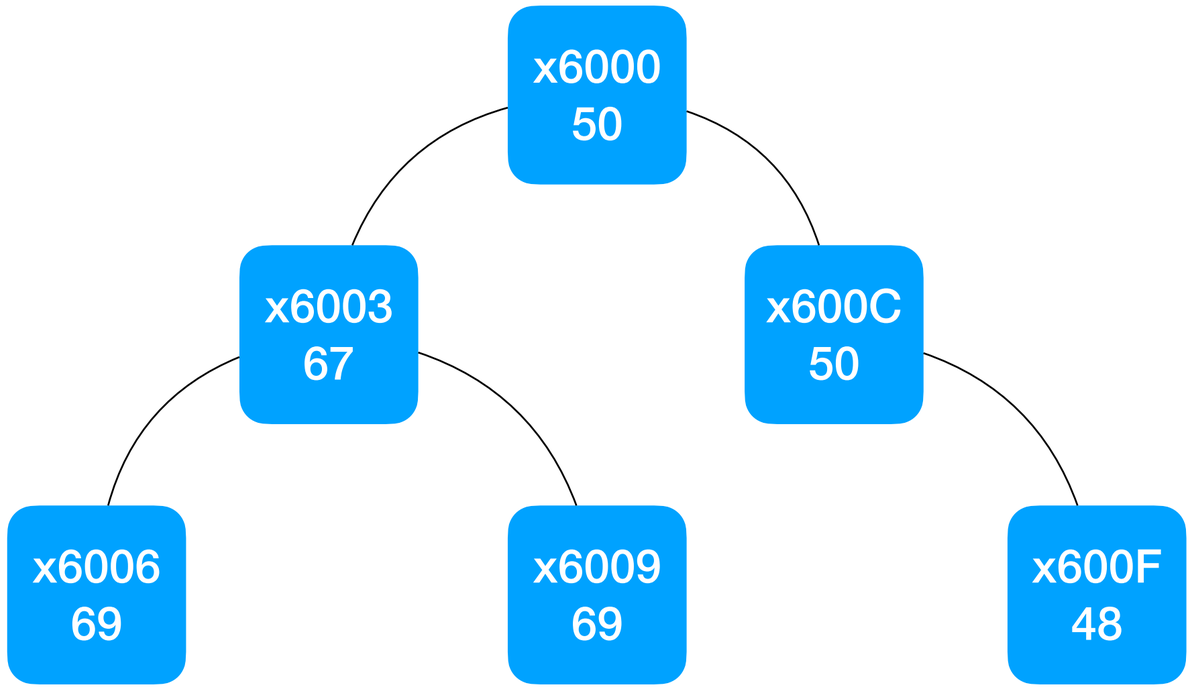


# Tree Traversal in LC3

```
typedef struct Node{
    char symbol;
    struct Node *left;
    struct Node *right;
}node;
```

# Draw the tree

```
.ORIG x6000
.FILL 50        x6000
.FILL x6003     x6001
.FILL x600C     x6002
.FILL 67        x6003
.FILL x6006     x6004
.FILL x6009     x6005
.FILL 69        x6006
.FILL 0         x6007
.FILL 0         x6008
.FILL 69        x6009
.FILL 0         x600A
.FILL 0         x600B
.FILL 50        x600C
.FILL 0         x600D
.FILL x600F     x600E
.FILL 48        x600F
.FILL 0         x6010
.FILL 0
.END
```




What does inorder traversal of this tree print out?

# Weekend exercise

```
void traverse_inorder(node *cursor){
  if (cursor==NULL)
    /* Tree empty; do nothing */
    return;
  traverse_inorder(cursor->left);
  printf("%c", cursor->data);
  traverse_inorder(cursor->right);
}
```

```
.ORIG x3000
MAIN
  LD R5, RSTACK
  LD R6, RSTACK
  LD R0, ROOT
  STR R0, R6, #0 ; push argument onto stack
  JSR TRAVERSE_INORDER
  ADD R6, R6, #2 ; caller teardown (pop return value & a
  HALT

ROOT .FILL x6000
RSTACK .FILL x7000

TRAVERSE_INORDER

;; FILL IN ALL THE NECESSARY CODE!!

RET
.END

.ORIG x6000
;; … COPY FROM PREVIOUS SLIDE
.END
```