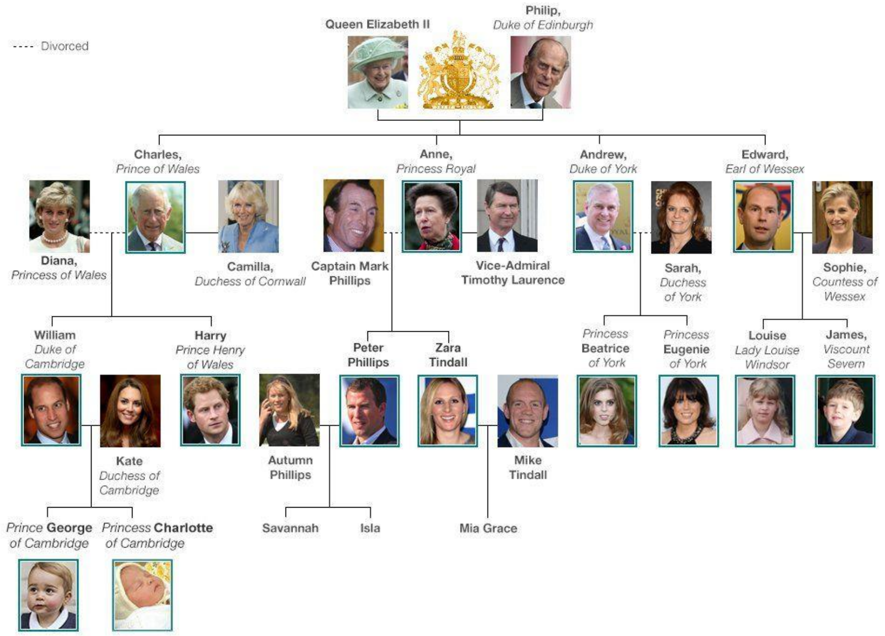
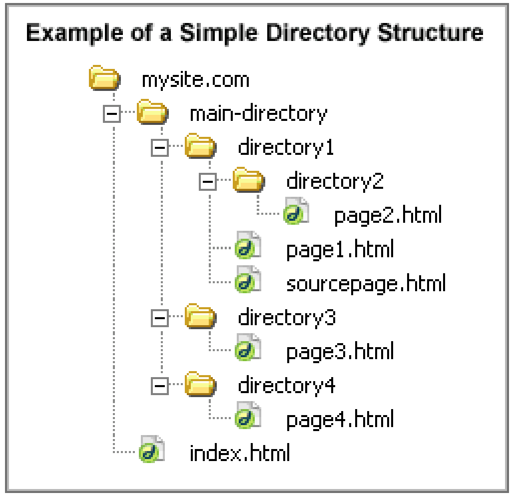
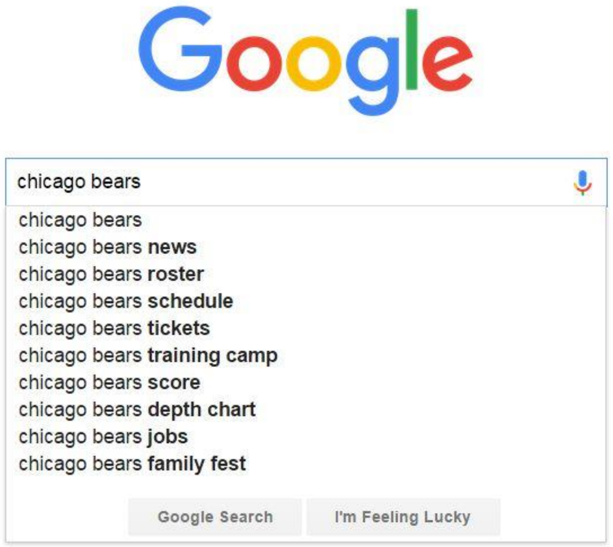
- **Sequential** data structure
  - Array
  - Linked list
  - Stack
- What are some limitations of sequential data structure?
- Fast or slow?
  - Insertion
  - Deletion
  - Searches

How can we organize hierarchical data?







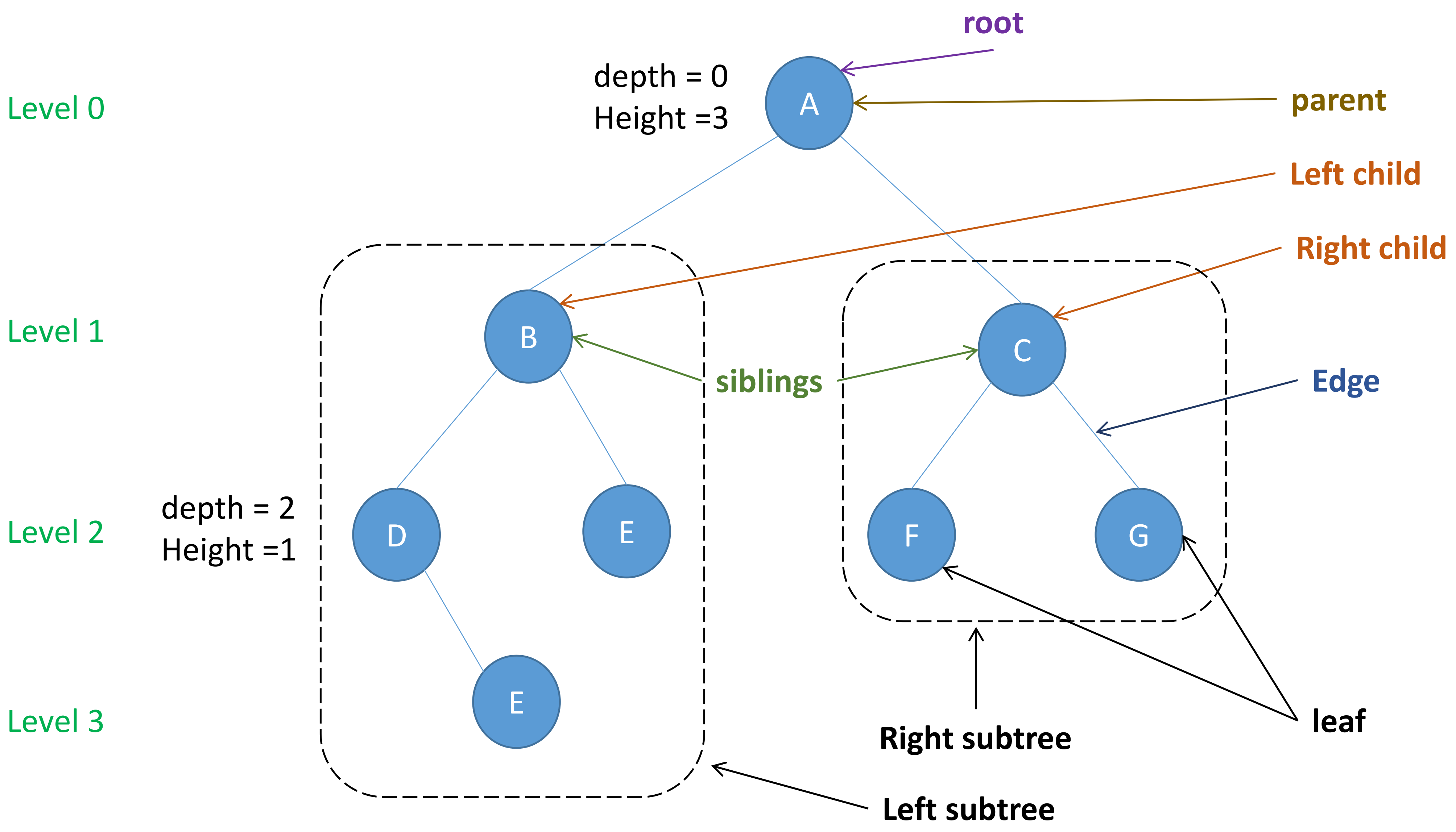
Tree data structure

- Productivity experts say that breakthroughs come by thinking “**nonlinearly**.”
- "Sequentiality is an illusion" *Kevin Skadron*
  http://www.cs.virginia.edu/~skadron//

Tree Terminology

- **Node** is a structure which may contain a value or condition, or represent a separate data structure. Each node of the tree is represented as a **circle.** Each node in a tree has zero or more child nodes.
- **Tree:** is a data structure made up of nodes or vertices and edges **without having any cycle** and stores elements hierarchically.
- **Parent:** A node that has a child is called the child's parent node. Each node has one parent with the exception of the root.
- **Child:** a node extending from another node.
- **Root:** The topmost node in a tree.
- **Sibling:** Two nodes are siblings if they have same parent.
- **Leaf:** a node with no children (external node).
- **Internal node**: a node with at least one child.
- **Ancestor:** a node reachable by repeated proceeding from child to parent. (Nodes on the path from the node to the root).

- **Descendant:** a node reachable by repeated proceeding from parent to child.
- **Edge:** a pair of nodes (u,v) such that u is directly connected to v. Children are connected to the parent by an arrow from the parent to the child. An arrow is usually called a **directed edge** or a directed branch.
- **Path**: sequence of nodes. There is a **unique** path from the root to every node in the binary tree.
- **Length of a path:** # of edges in path.
- **Level**: The level of a node is defined by the number of connections between the node and the root.
- **Height of a node:** Number of edges on the longest path from the node to a **leaf.** Height of leaf nodes = 0.
- **Height of a binary tree:** Number of edges on the longest path from the root to a leaf (height of the root or maximum depth of any node). Height of an empty tree is -1.
- **Depth of a node** is the number of edges from the node to the **root** (number of ancestors). Depth of root = 0.


root
depth = 0
Height =3
A
parent
Level 0
Left child
Right child
Level 1
B
siblings
C
Edge
depth = 2
Height =1
Level 2
D
E
F
G
E
Level 3
Right subtree
leaf
Left subtree

SECTIONS SEARCH **Chicago Tribune** SUBSCRIBE

TUESDAY SEP. 29, 2015 TRENDING SPORTS SUBURBS BREAKING BUSINESS OPINION BEARS ADVERTISING

Habla espanol? Local employers say it's increasingly important

Employers are increasingly looking to hire applicants who can communicate effectively in more than one language. (Jos
Pelaez Inc / Getty Images/Blend Images)

Alexia Elejalde-Ruiz · Contact Reporter
Chicago Tribune

In a survey conducted in Summer 2015 by Northern Illinois University's Center for Government Studies, a **third** of employers said it is important to hire a recent college graduate who can communicate effectively in more than one language, and half said it will be important **five years** from now.

http://www.chicagotribune.com/business/ct-bilingual-employees-niu-0929-biz-20150928-story.html

Recursion

- A function that calls itself
- The function actually knows how to solve only the simplest case(s) (base case(s)).

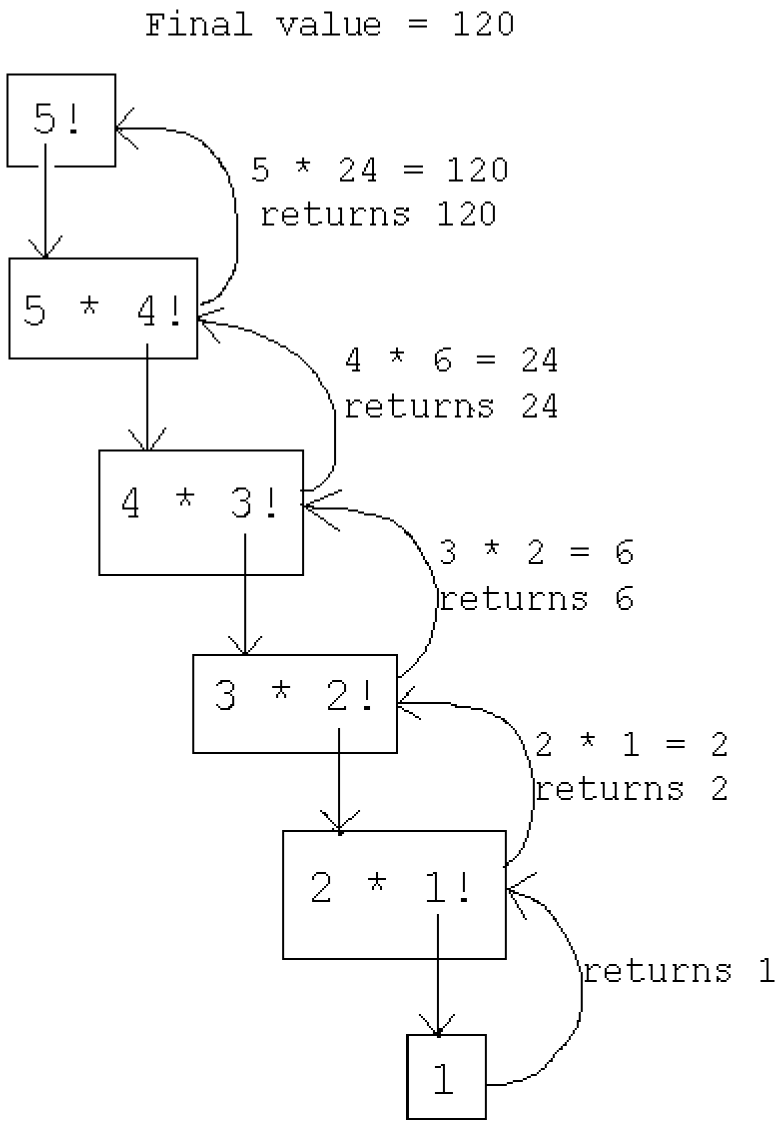
```cpp
#include <iostream>

void countDown(int count)
{
 std::cout << "push " << count;
 countDown(count - 1);
}

int main()
{
 countDown(5);
 return 0;
}
```

What is the output?

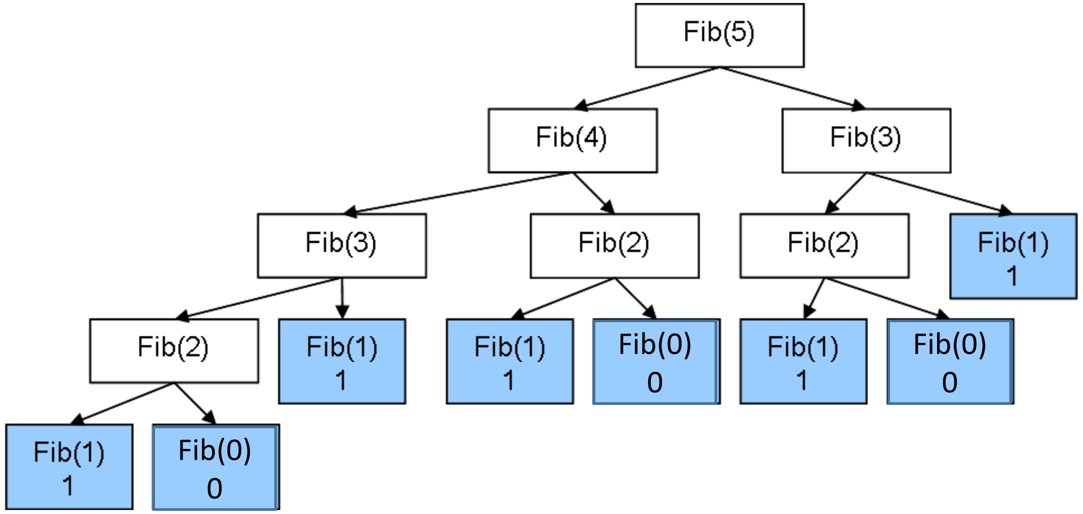
```cpp
#include <iostream>

void countDown(int count)
{
 std::cout << "push " << count << '\n';

if (count > 1) // termination condition
    countDown(count - 1);

 std::cout << "pop " << count << '\n';
}

int main()
{
 countDown(5);
 return 0;
}
```

```
push 5
push 4
push 3
push 2
push 1
pop 1
pop 2
pop 3
pop 4
pop 5
```

```
#include<iostream>
using namespace ::std;
long factorial(long number)
{
 if (number <= 1)
      return 1;
else
      return (number * factorial(number - 1));
}
int main()
{
  cout << factorial(5);
  return 0;
}
```



```
#include<iostream>
using namespace ::std;

long fibonacci(long n)
{
 if (n <2)
      return n;
 else
      return (fibonacci(n - 1) + fibonacci(n - 2));
}

int main()
{
 cout << " Fibonacci of 5 = " << fibonacci(5);
 return 0;
}
```



Fibonacci of 5 = 5

Write a recursive function print_backwards, that receives a vector (e.g., 2,4,7) and prints it in reverse order (e.g., 7,4,2).

```
#include<iostream>
#include<vector>
using namespace std;
void print_backwards(vector<int> v)
{







}
int main()
{vector<int> v = { 2,4,7 };
print_backwards(v);
return 0;}
```

Write a recursive function print_backwards(), that receives a vector (e.g., 2,7,4) and prints it backwards on the screen (e.g., 4,7,2).

```
#include<iostream>
#include<vector>
using namespace std;
void print_backwards(vector<int> v)
{
 if (v.size() > 0)
    {
     cout << v.back();
     v.pop_back();
     print_backwards(v);
    }
}
int main()
{vector<int> v = { 2,4,7 };
print_backwards(v);
return 0;}
```

Write a recursive function to sum all the numbers in a vector

```
#include<iostream>
#include<vector>
using namespace std;
int findSum(vector<int> v)
{
 if (v.empty())
    return 0;
else
    {
     int last = v.back();
     v.pop_back();
     return last + findSum(v);
    }
}
int main()
{vector<int> v = { 3,7,9 };
cout<<findSum(v);
return 0;}
```

Data **structures**

- A group of data elements (members) grouped together under one name.
- To access the members of an object we simply insert a **dot (.)** between the object name and the member name.

```
struct product
{
   int quantity;
   double price;
};
product apple, orange;  // Many objects can be declared from a single structure type.
apple.quantity = 4;     // Access member of a structure.
```

Pointers to structures

- To access the members of a pointer to structure
  - Use the arrow operator **->** (minus sign and greater than sign with no whitespace)

    `PointerToStruct->member`
  - Which is equivalent to:

    `(*PointerToStruct).member`

```
product apple, *ptr;
ptr = &apple;
ptr->quantity = 5
// OR
// (*ptr).quantity = 5;
```

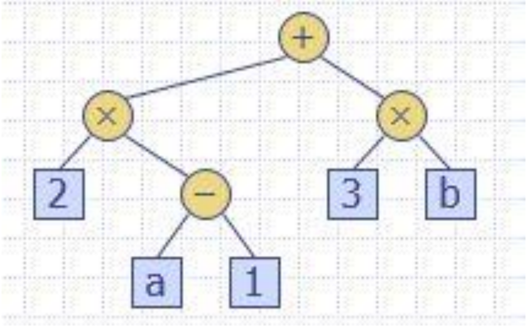
- More details http://www.cplusplus.com/doc/tutorial/structures/

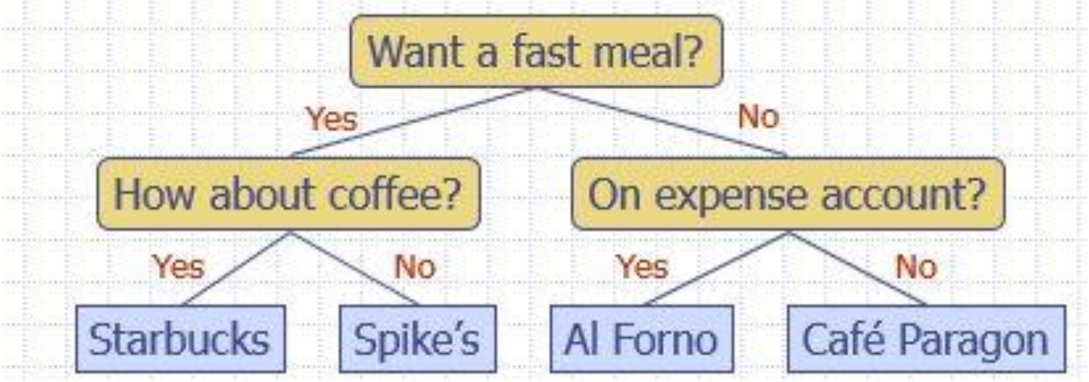
```cpp
#include <iostream>
using namespace std;
struct Time {
int hour;
int minutes;
int seconds;
};
void main()
{
 Time current, *ptr;
 current.hour = 11;
 current.minutes = 2;
 current.seconds = 30;

cout << " The time now = " <<current.hour << ":"<< current.minutes <<":"<<
current.seconds;

 ptr = &current;
 ptr->hour++;
 cout << "\n The time after 1 hour = " << current.hour << ":" << current.minutes <<
":" <<       current.seconds;
}
```

```
The time now = 11:2:30
The time after 1 hour = 12:2:30
```

Binary Tree

- Every node in a **binary tree** has **at most two children**.
- Applications:
  - arithmetic expressions (e.g., (2 × (a - 1) + (3 × b)))
  - decision processes (e.g., dining decision)
  - searching



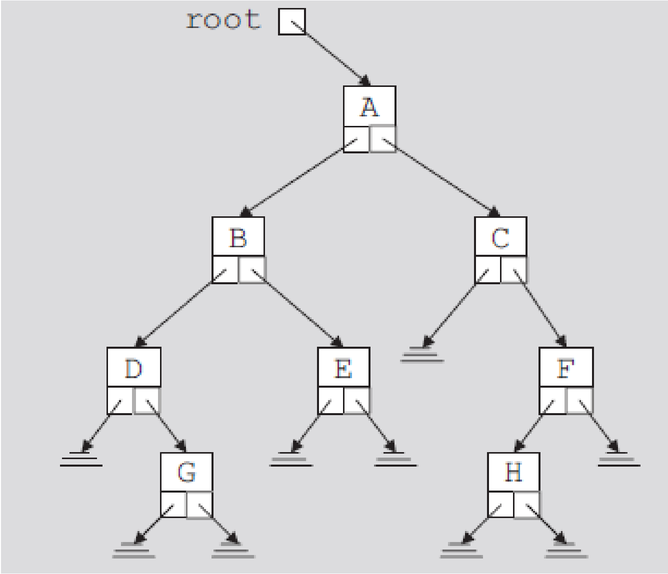


Properties of Binary Trees

- n = number of nodes
- ne = number of external nodes
- ni = number of internal nodes
- h = height of a binary tree
- Then the binary tree has the following properties:
  - $h+1 \leq n \leq 2^{h+1} - 1$
  - $1 \leq ne \leq 2^{h}$
  - $h \leq ni \leq 2^{h} - 1$
  - $\log(n+1) - 1 \leq h \leq n-1$

- For each node
  - The data stored in **`data`**
  - A pointer to the left child stored in **`left`**
  - A pointer to the right child stored in **`right`**

```
class Node
{
 int data;
 Node *left;
 Node *right;
};
```



- Pointer to **root** node is stored outside the binary tree. The root node defines an entry point into the binary tree.

Write a C++ program to find the maximum depth of a tree

```
#include<iostream>
using namespace std;
class Node
{
public:
 int data;
 Node* left;
 Node* right;
};
/* Helper function that allocates a new node with the given data and NULL left and right
pointers.*/
Node* newNode(int data)
{
  Node* n = new Node; // dynamically allocate new objects of type Node
  n->data = data;
  n->left = nullptr;
  n->right = nullptr;
  return(n);
}
```

Replace with constructor

```
int main()
{
 Node *root = newNode(1);
 root->left = newNode(2);
 root->right = newNode(3);
 root->left->left = newNode(4);
 root->left->right = newNode(5);
 cout << "The maximum depth is " << maxDepth(root);
 return 0;
}
```

Replace with insert function

```
/* Compute the "maxDepth" of a tree -- the number of edges along the longest path
from the root node down to the farthest leaf node.*/
int maxDepth(Node* n)
{
 if (n == nullptr)
     return -1; // depth of an empty tree
else
{
  // compute the depth of each subtree
  int lDepth = maxDepth(n->left);
  int rDepth = maxDepth(n->right);
  // use the larger one
  if (lDepth > rDepth)
      return(lDepth + 1);
  else
      return(rDepth + 1);
  // OR just
  // return 1 + max(maxDepth(n->left), maxDepth(n->right));
} }
```

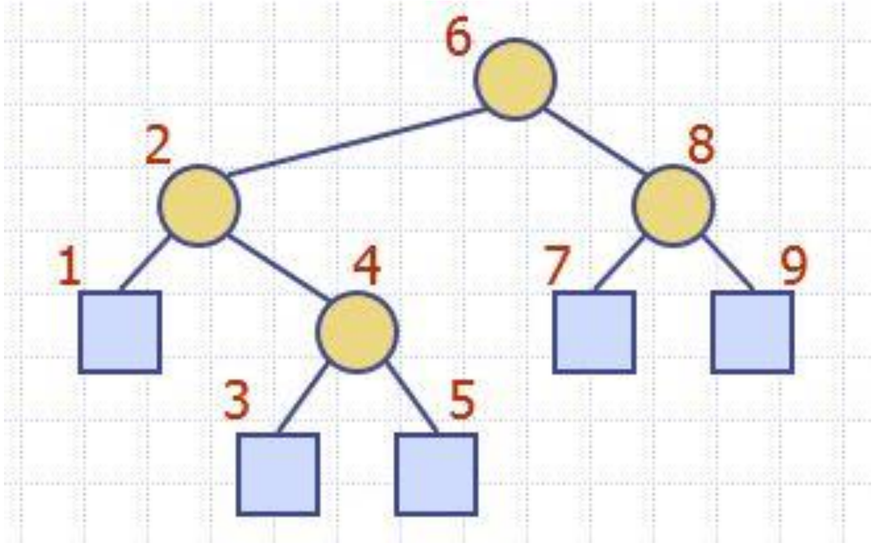
Binary Tree Traversal

- The item insertion, deletion, and lookup operations require that the binary tree be traversed or **visit** each node of the binary tree.
- Must **start** at the **root**, and then we can first visit the
  a) node *or*
  b) subtrees
- These choices lead to different recursive traversal algorithms
  1) Depth-first traversal
     1) Inorder
     2) Preorder
     3) Postorder
  2) Breadth-first traversal

1) Inorder traversal (LNR)

- Traverse the left subtree
- Visit the node (We cannot do step 2 until we have finished step 1).
- Traverse the right subtree

```
void inorder(Node *p)
{
 if (p != NULL)
    {
     inorder(p->left);
     cout << p->data;
     inorder(p->right);
    }
}
```



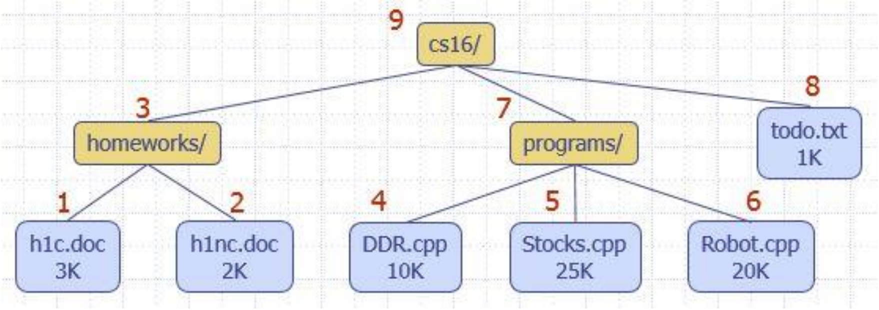
2) Preorder traversal (NLR)

- Visit the node
- Traverse the left subtree
- Traverse the right subtree

```
void preorder(Node *p)
{
 if (p != NULL)
    {
     cout << p->data;
     preorder(p->left);
     preorder(p->right);
    }
}
```

3) Postorder traversal (LRN)

- Traverse the left subtree
- Traverse the right subtree
- Visit the node



- **Application:** compute space used by files in a directory and its subdirectories

```
void postorder(Node *p)
{
 if (p != NULL)
    {
     postorder(p->left);
     postorder(p->right);
     cout << p->data;
    }
}
```