Linked lists, you can go only to the previous or next one

Tree is a different structure, at each node you have two options.

From each node there are two possible directions at most.

This is a binary tree, each node has a left and a right child.

Parent

Left child Right Child

In a binary tree pointers are the parent the left and the right child.

Properties of binary trees:

* Each node in the tree is an object, it has multiple properties, each called x.
* Each node has a key that is going to identify it e.g. for cities the name of the city would be the key.
* Each node has a parent and we will call it P
* Each parent has two children at most, can have 0 or 1 children also.
* If P equals NULL, then that node is the root of the tree.
* If x has no left child. x.leftchild == NULL
* If x has no right child x.rightchild == NULL

Binary tree structure is such that they are very efficient for searching and adding new nodes.

For each children there are two children.

Imagine a tree T, T has a root and it has a left and a right branch.

Within the tree there are sub trees of T, call the first one T’. Even T’ prime has a root. Even it has a root and a left and right branches (pointes), basically it’s a fractal structure. Continue on to the smallest unit of the tree which is just the individual node, you can think of it as just a root without children. You can build up a tree with smaller trees. You can keep going. The tree is defined by self-similarity, you can define the tree recursively, where you can keep going down and recursively define the bigger bits from the smaller ones, it also means that computationally we can use a technique called recursion to examine the tree.

printName (treeNode \*root)

//function prints the name of the root node.

Since the structure is self-similar we can give a function of a new root and traverse through the list like this.

If(root.leftchild != NULL)

printName(root.leftChild)

If(root.rightchild != NULL)

printNamr(root.rightChild)

cout<<rootName