1 Review of Traversals

Common data structures:

1. Arrays
2. Singly-linked lists
3. Trees

Typical traversal methods:

- Loops for arrays and lists.
- Recursion for trees.
- Recursion can also be used for traversing lists (and arrays).
- Loops can also be used for traversing trees.

2 Tree Traversals

Programs that use trees often need to process all of the nodes in a tree. This process is called a *tree traversal* (it is sometimes called *walking the tree*).

For a binary tree, there are three common ways:

1. Pre-order
2. In-order
3. Post-order

Note: Tree traversals *will be* on a future quiz and test.

Note: Tree traversals *will be* on a future quiz and test.

Review: Typical Binary Tree Representation

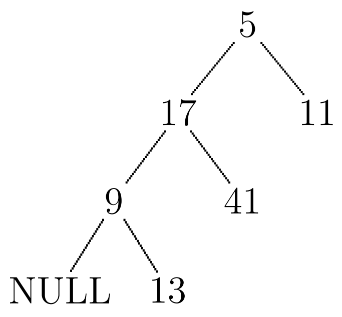
A node in a binary tree can be defined using the following:

```
struct BinaryTreeNode
{
    int  data;
    BinaryTreeNode *left;
    BinaryTreeNode *right;
};
```

As we saw with nodes in linked lists, it is convenient to declare a new data type for pointers to the nodes.

```
typedef struct BinaryTreeNode *BinaryTreeNodePtr;
```

The tree that will be used to illustrate the tree traversal methods is shown below:



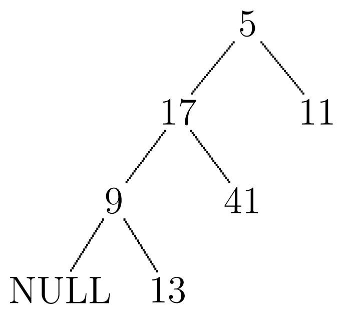
2.1 Pre-order Traversal

1. Process the root
2. Process the nodes in the left subtree (recursion)
3. Process the nodes in the right subtree (recursion)

```
void  PrintPreOrder( BinaryTreeNodePtr t )
{
    if( t != NULL )
    {
        cout << t->data << endl;

        PrintPreOrder( t->left );

        PrintPreOrder( t->right );
    }
}
```



Output:

```
5
17
9
13
41
11
```

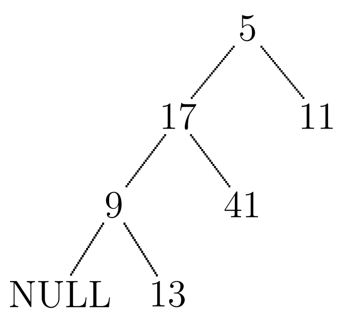
2.2 In-order Traversal

1. Process the nodes in the left subtree (recursion)
2. Process the root
3. Process the nodes in the right subtree (recursion)

```
void  PrintInOrder( BinaryTreeNodePtr t )
{
    if( t != NULL )
    {
        PrintInOrder( t->left );

        cout << t->data << endl;

        PrintInOrder( t->right );
    }
}
```



Output:

```
9
13
17
41
5
11
```

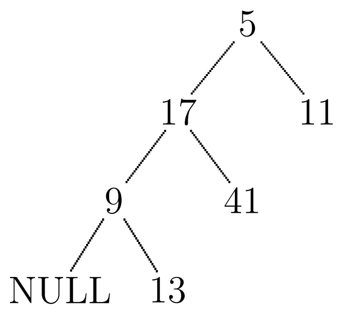
2.3 Post-order Traversal

1. Process the nodes in the left subtree (recursion)
2. Process the nodes in the right subtree (recursion)
3. Process the root

```
void  PrintPostOrder( BinaryTreeNodePtr t )
{
    if( t != NULL )
    {
        PrintPostOrder( t->left );

        PrintPostOrder( t->right );

        cout << t->data << endl;
    }
}
```



Output:

```
13
9
41
17
11
5
```

Next Lecture

Constructing trees.