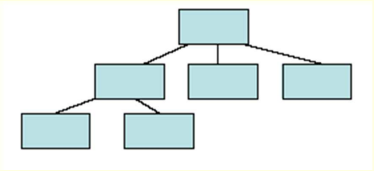
Lecture 11.1

Tree data structure

Trees

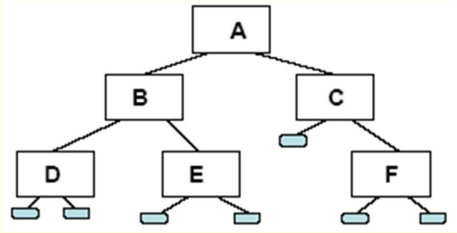
- Trees are structures that represent data in a hierarchical manner
- The top node is called the root and it may have one or several leaves or children



Binary Trees

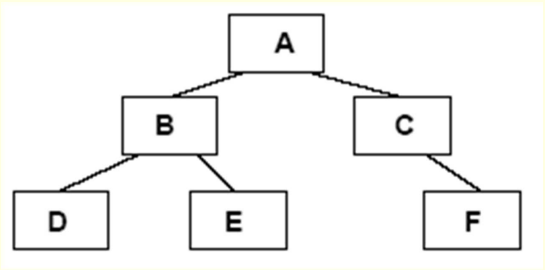
- Binary trees have at most two children (left and right)
- A binary tree is either:
  - A leaf with no branches; or
  - consist of a root and two children, left and right, each of which are themselves binary trees.
- This is a recursive definition!

Binary Tree representation



- A node without branches is a leaf (Here D, E and F are leaves) or
- A binary tree with a left and a right node, each of which is a binary tree.

Binary Tree representation



- A is a root
- B and C are left and right branches respectively
- D, E and F are leaves

Node implementation

- Each node can be implemented with the following code:

```
struct TreeNode{
   int data;
   TreeNode* leftTreeNode;
   TreeNode* rightTreeNode;
};
```

OR

Binary class template

```
template <class dataType>
class BinaryTreeNode {
    public:
        BinaryTreeNode( );
        bool isLeaf ( );
        dataType getData ( );
        void setData( const dataType & d );
        BinaryTreeNode * getLeft ( );
        BinaryTreeNode * getRight ( );
        void setLeft ( BinaryTreeNode * T1 );
        void setRight ( BinaryTreeNode *T1 );
    private:
        dataType treeNodeData;
        BinaryTreeNode * leftTreeNode;
        BinaryTreeNode * rightTreeNode;
};
```


Constructor

```
template <class dataType> //constructor
BinaryTreeNode<dataType>::BinaryTreeNode
   ( )
{
   leftTreeNode = 0;
   rightTreeNode = 0;
}
```

Binary tree implementation

```
template <class dataType>
bool BinaryTreeNode<dataType>::isLeaf()
{
   return ((this->leftTreeNode == NULL) && (this-
   >rightTreeNode == NULL));
}
template <class dataType>
dataType BinaryTreeNode<dataType>::getData( )
{
   return treeNodeData;
}
```


Binary tree implementation (cont.)

```
template <class dataType>
void BinaryTreeNode <dataType>::setData ( const
   dataType & d )
{
   treeNodeData = d;
}
```

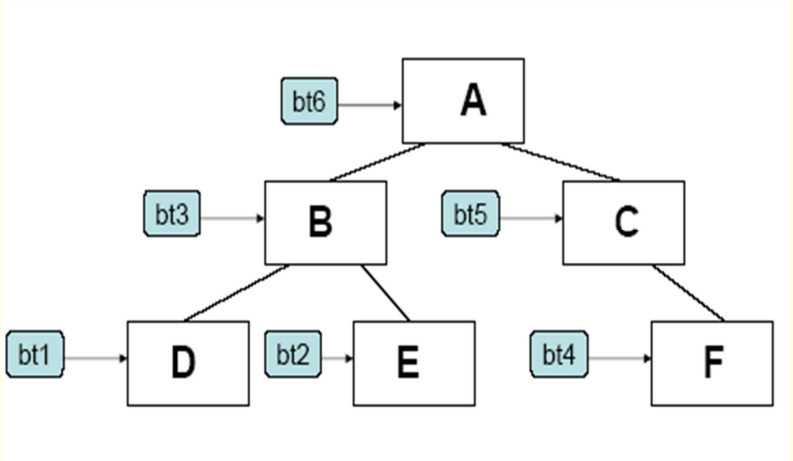

Binary tree implementation (cont.)

```
template <class dataType>
BinaryTreeNode <dataType> *
   BinaryTreeNode<dataType>:: getLeft( )
{
   return leftTreeNode;
}
template <class dataType>
BinaryTreeNode <dataType> *
   BinaryTreeNode<dataType>:: getRight( )
{
   return rightTreeNode;
}
```


Binary tree implementation (cont.)

```
template <class dataType>
void BinaryTreeNode <dataType> :: setLeft (
   BinaryTreeNode * T1 )
{
   leftTreeNode = T1;
}
template <class dataType>
void BinaryTreeNode<dataType> :: setRight (
   BinaryTreeNode * T1 )
{
   rightTreeNode = T1;
}
```

Example



Example

```
int main()
{
    typedef BinaryTreeNode <char> charTree;
    typedef charTree * charTreePtr;

    //Create left subtree ( rooted at B )
    //Create B's left subtree
    charTreePtr bt1 = new charTree;
    bt1->insert ( 'D' );

    //Create B's right subtree
    charTreePtr bt2 = new charTree;
    bt2->insert ( 'E' );
```


Example (cont.)

```
//Create node containing B, and link
//up to subtrees
charTreePtr bt3 = new charTree;
bt3->insert ( 'B' );
bt3->makeLeft ( bt1 );
bt3->makeRight ( bt2 );
//** done creating left subtree
```


Example (cont.)

```
//Create right subtree
//Create C's right subtree
charTreePtr bt4 = new charTree;
bt4->insert ( 'F' );

//Create node containing C and link
//up its right subtree;
charTreePtr bt5 = new charTree;
bt5->insert ( 'C' );
bt5->makeRight ( bt4 );
//** done creating right subtree
```

Example (cont.)

```
//Create the root of the tree and link together
charTreePtr bt6 = new charTree;
bt6->insert('A');
bt6->makeLeft ( bt3 );
bt6->makeRight ( bt5 );

//print out the root
cout<< "Root containts: " << bt6 ->getData( ) <<endl;

//print out the left subtree
cout <<"Left subtree root: " <<bt6 ->left( ) ->getData( )
   <<endl;
```


Example (cont.)

```
//print out the right subtree
cout <<"Right subtree root: "
<< bt6->right( )->getData( ) <<endl;

//print out left most child in tree
cout <<"Left most child is: "<<
bt6->left( )->left( )->getData( ) <<endl;

//print out right most child in tree
cout <<"Left most child is: "<<
bt6->right( )->right( )->getData( ) <<endl;
```