UNIVERSITY LOGO DEPARTMENT LOGO  
TERM INFORMATION COURSE NAME

**Tree Project**

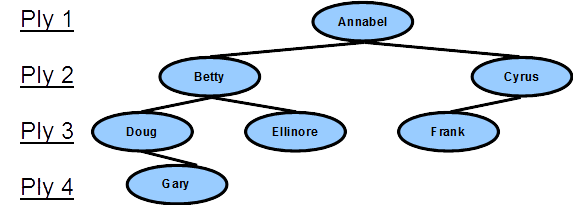
# Overview

This tree project is intended to provide students with an understanding of how tree structures are built and connected and the basic traversal algorithms associated with them. This project involves the implementation of a templated tree node and functions for traversing such nodes.

# Structure

Information is often non-linear: it cannot be easily placed “in line” because of more complex relationships that exist between pieces of data. This kind of data is usually difficult to store in arrays, lists, or tables. For example, a descendant family tree would be difficult to implement in a list fashion, because each parent may have several children. This information is more easily stored in a ***tree***. A tree is made up of a ***root*** node and its descendants. Each node in the tree may have zero or more ***child*** nodes. A node with no children is called a ***leaf***node. The ply level of node is its depth.

**A tree holding a family tree (of descendants). Betty is the left child of Annabel, and Gary is a leaf node.**



# Traversal

In this project, students will implement three traversals (ways of visiting nodes): ***depth-first pre-order***, ***depth-first post-order***, and ***breadth-first***.

#### **Pre-Order**

In a (depth-first) pre-order traversal a node's data is processed ***before*** its children's data:

def preOrderTraversal(node, function):  
 function(node)  
  
 for child in node.children:  
 depthFirstPreorder(child)

A pre-order traversal of the example tree would yield:  
Annabel, Betty, Doug, Gary, Ellinore, Cyrus, Frank

#### **Post-Order**

In a (depth-first) post-order traversal a node's data is processed ***after*** its children's data:

def postOrderTraversal(node, function):  
 for child in node.children:  
 depthFirstPostorder(child)  
  
 function(node)

A post-order traversal of the example tree would yield:  
Gary, Doug, Ellinore, Betty, Frank, Cyrus, Annabel

#### **Breadth-First**

In a breadth-first traversal a node's data is processed ***before nodes in lower levels***. The techniques involved in a breadth first-traversal are very different from those of depth-first traversals. A queue (or list) and a loop are used to store nodes rather than relying on recursive calls:

def breadthFirstTraversal (root, function):  
 queue = [ root ]  
  
 while not queue.empty():  
 node = queue.popfront()  
 function(node)  
  
 for child in node.children:  
 queue.append(child)

The output of a breadth-first search would look like this:   
Annabel, Betty, Cyrus, Doug, Ellinore, Frank, Gary

# Implementation

You will implement a generic tree node with the following methods and the following traversal functions:

TreeNode<T>

*TreeNode<T>()*

Constructor for **TreeNode**. It stores a default-constructed data value and starts with no children.

*TreeNode<T>(T element)*

Constructor for **TreeNode**. It stores element as its data and starts with no children.

*~TreeNode<T>()*

Destructor for **TreeNode**. It recursively deletes all children.

*T& getData()*

Returns a reference to the stored data.

*size\_t getChildCount()*

Returns the number of children this node has.

*TreeNode<T>\* getChild(size\_t index)*

Returns the child node at specified by index.

*void addChild(TreeNode<T>\* child)*

Add the child node to the children of this node.

*void removeChild(size\_t index)*

Remove the child node at specified by index.

*void deleteChildren()*

Recursively deletes all children.

Templated, Standalone Functions (Not Contained)

*void breadthFirstTraverse(TreeNode<T>\* root, void (\*dataFunction)(T&))*

Breadth-first traversal starting at root. Calls dataFunction on the data element to process it.

*void preOrderTraverse(TreeNode<T>\* root, void (\*dataFunction)(T&))*

Pre-order traversal starting at root. Calls dataFunction on the data element to process it.

*void postOrderTraverse(TreeNode<T>\* root, void (\*dataFunction)(T&))*

Post-order traversal starting at root. Calls dataFunction on the data element to process it.

# Submissions

Submit the following files at the end of this project:

* Tree.h

Place them in the ROOT directory of your zip file, not in a subdirectory. Do not submit any other files.