ECE 250 *Algorithms and Data Structure*
Department of Electrical and Computer Engineering
University of Waterloo
Please send any comments or criticisms to dwharder@alumni.uwaterloo.ca
with the subject ECE 250 Questions 2.4.
Assistances and comments will be acknowledged.

4.2*a* For each of the operations, determine whether it is describing a local or global property of the tree relative to the location of the node on which the query is made.

1. The degree of the node.
2. The number of descendants of this node.
3. The root of the tree.
4. The parent of the node.
5. The depth of the node.
6. The height of the sub-tree rooted at this node.

4.2*b* It is relatively easy to define an implicit linear order: $a < b$ if $b - a$ is positive, $f(n) = \mathbf{o}(g(n))$ if $\lim_{n\to\infty} \frac{f(n)}{g(n)} = 0$. Hierarchical orders, however, are almost always defined explicitly: b is the parent of a and c, d, and e are children of a. Suggest why it is so difficult to define an implicit hierarchical order.

4.2*c* Write a recursive member function that finds the depth of a node.

```
template <typename Type>
int Simple_tree<Type>::depth() const{




}
```

Write an iterative member function that finds the depth of a node. Such a function would not call itself, but would instead use, for example, a for, while, or do-while loop.

```
template <typename Type>
int Simple_tree<Type>::depth() const{




}
```

4.2*d* Write a recursive member function that returns a pointer to the root of the tree containing this node. The name of the function should be `root`. Your implementation should have the correct signature for such a member function.

4.2*e* Write a one-line member function that returns the number of siblings of a node. The root node has zero siblings.

```
template <typename Type>
int Simple_tree<Type>::sibling_count() const {
    return
}
```

4.2*f* Write a function that deletes the current node. If the current node is the root node, all children will have their parent set to `nullptr` (this will generate a forest). If the current node is not a root node, each of the children is made a child of the current node's parent. Hint: remember `erase`.

```
template <typename Type>
Simple_tree<Type> *Simple_tree<Type>::lca( Simple_tree<Type> const &node ) const {
```

4.2*g* The lowest common ancestor was defined in Question Set 4.1. Write a member function that returns a pointer to the lowest common ancestor of a node and `nullptr` if the nodes are from two different trees. Hint: you are welcome to call the member functions `depth` and `root` defined above.

```
template <typename Type>
Simple_tree<Type> *Simple_tree<Type>::lca( Simple_tree<Type> const &node ) const {
```

4.2*h* Write a non-recursive member function that does a breadth-first traversal of the tree rooted at this node by printing out the element stored in the node followed by a comma and a space. The last node should not be followed by a comma and a space.

Note: the last node not being followed by a comma and a space is equivalent to saying the first node should not be preceded by a comma and a space—that might make the programming easier. Use your `Single_list` class for the queue.

```
template <typename Type>
Simple_tree<Type> *Simple_tree<Type>::dft() const {
    Single_list< Simple_tree<Type> * > list;
```