1.

```
typedef struct NODE *pNODE;
struct NODE {
    pNODE leftmostchild, rightsibling;
};

int number_of_nodes(pNODE n)
{
    if (n == NULL) return 0;
    else return 1 + number_of_nodes(n->leftmostchild) + number_of_nodes(n->rightsibling);
}
```

2. We assume that there is no worst-case scenario where n is `NULL`. Therefore we must handle as the base case where n is a leaf. We assume that the initial input to `max_tree` is not `NULL`.

```
typedef struct NODE *pNODE;
struct NODE {
    int label;
    pNODE leftmostchild, rightsibling;
};

int max_tree(pNODE n)
{
    if (isleaf(n)) return n->label;
    else if (has_child_and_sibling(n))
        return max(n->label, max(max_tree(n->leftmostchild), max_tree(n->rightsibling)));
    else if (isparent(n)) return max(max_tree(n->leftmostchild), n->label);
    else return max(max_tree(n->rightsibling), n->label);
}
```

3.

```
int eval(pNODE n)
{
    int val1, val2;

    if ((n->op) == 'i')  return n->value;
    else {
        val1 = eval(n->leftmostChild);
        if (n->leftmostChild->rightSibling == NULL) {
            switch (n->op) {
            case '+': return val1; // unary plus
            case '-': return -val1; // unary minus
            }
        }
        else {
```

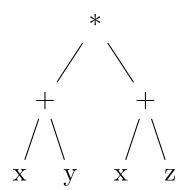
```
            val2 = eval(n->leftmostChild->rightSibling);
            switch (n->op) {
            case '+': return val1 + val2;
            case '-': return val1 - val2;
            case '*': return val1 * val2;
            case '/': return val1 / val2;
            }
        }
    }
}
```

5.

a) 1, 2, 4, 8, 9, 3, 5, 10, 13, 14, 15, 6, 7, 11, 12.

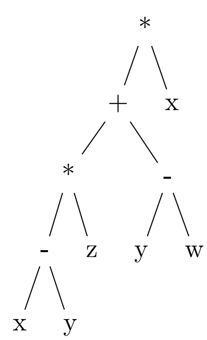
b) 8, 9, 4, 2, 13, 15, 14, 10, 5, 6, 11, 12, 7, 3, 1.

6.



$* + xy + xz.$

$xy + xz + *.$



$* + * - xyz - ywx.$

$xyw - xy - z * + * .$