7 Algorithms (fms27)

Consider binary trees whose nodes have three fields: key (a single character between U and Z), left subtree, right subtree, with the two subtrees either both empty or both non-empty. Assuming suitable constructors, indicate an empty tree as $T()$ and a non-empty tree as $T(key, leftSubtree, rightSubtree)$.

(a) Define unambiguous pre-order, in-order and post-order representations for such a tree $t$, called $r_{pre}(t)$, $r_{in}(t)$, $r_{post}(t)$, consisting of strings over the \{U, V, W, X, Y, Z, (, )\} alphabet, with balanced brackets, with three characters (two brackets and a letter) for each node, starting and finishing with a bracket unless $t$ is empty. Formally describe your three representations for a generic tree $t$, then produce the corresponding strings for the following tree.  

(b) In this obfuscated pseudocode, the input $v0$ is a syntactically correct $r_{post}(t)$. Clearly explain (i) the purpose of the code; (ii) how it works; and (iii) how one should invoke it. Substitute meaningful explanatory identifiers for those $T_x$, $v_y$ and $m_z$. Identifiers $v4$ and $v6$ are worth more marks than the others.

(c) Explain in detail (i) how line 16 works, and (ii) why $v4$ will never be uninitialised when line 16 is executed. 

(d) Write clear pseudocode that takes a Tree $t$ and returns $r_{in}(t)$. 

```python
class T1:
    # data members of objects of type T1
    v2: T14 of T3
    v4: T12
    v6: T13

    def m5(v0):
        if v0 is "":
            return T3()
        else:
            for v9 in v0:
                m10(v9)
            return v2.pop()

    def m10(v11):
        if v11 is one of {"U", "V", "W", "X", "Y", "Z":
            v6 = v4 is "")"
        else if v11 is ")":"
            if v6:
                v8 = v2.pop(); v7 = v2.pop()
            else:
                v8 = null; v7 = null
            v2.push(T3(v4, v7, v8))
        v4 = v11
```