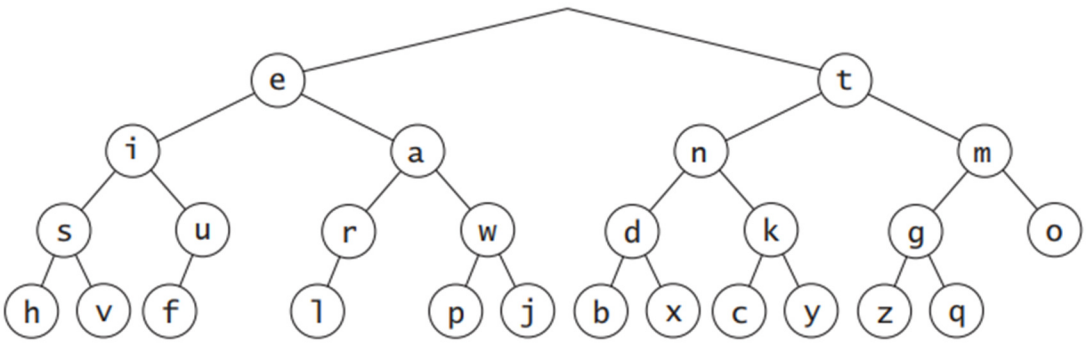
CS 700: Assignment 5

Due Date: *Nov 25 (Th) 11:59 PM* **Overdue submissions:** 25% penalty per day

Morse code (see the following table) is a common code that is used to encode messages consisting of letters and digits. Each letter consists of a series of dots and dashes; for example, the code for the letter **a** is •– and the code for the letter **b** is –•••.

a	• –	b	– • • •	c	– • – •
d	– • •	e	•	f	• • – •
g	– – •	h	• • • •	i	• •
j	• – – –	k	– • –	l	• – • •
m	– –	n	– •	o	– – –
p	• – – •	q	– – • –	r	• – •
s	• • •	t	–	u	• • –
v	• • • –	w	• – –	x	– • • –
y	– • – –	z	– – • •		

1) Store each letter of the alphabet in a node of a binary tree of depth 4 (see the following Figure). The root node is at depth 0 and stores no letter. The left node at depth 1 stores the letter **e** (code is •) and the right node stores the letter **t** (code is –). The four nodes at depth 2 store the letters with codes (••, •–, –•, – –).



To build the tree read a file in which each line consists of a letter followed by its code. The letters should be ordered by tree depth. To find the position for a letter in the tree, scan the code and branch left for a dot and branch right for a dash.

2) Encode a message by replacing each letter by its code symbol. Make sure you use a delimiter symbol between coded letters
3) Then decode the coded message using the Morse code tree.

A sample output may look like the following:

```
>
Enter a message
Hello World
Coded message
.... . .-.. .-.. ---   .-- --- .-. .-.. -..
Decoded message
hello  world
>
```

Marking Scheme:

- **Working program [50%]** A working program which satisfies all of the requirements automatically receives 50% of the total assignment mark. Each element of non-compliance will be penalized with respect to its severity.
- **Program Structure [25%]** A program which follows principles of Object-Oriented Design and structured programming rules (procedural, modular, uses parameters) to perfection automatically receives 25% of the total assignment mark. Marks are deducted depending on severity and number of occurrences of non-compliant elements.
- **Program Documentation [15%]**
    - **Internal documentation [10%]:** Documentation should be complete and in a standard format. Every non trivial part of the code should have a *clear* comment that explains it. In addition, every method or function, including the main program should have an explicative comment header. This header includes: module name, author, date of creation and purpose. A description of parameters and method output is mandatory. Marks are deducted according to the absence of these elements.
    - **External documentation [5%]:** HTML documentations generated by a program such as Doxygen should be provided.
- **Program Style [5%]** Style refers to Occam's razor principle. Code that is needlessly tricky, obscure, or difficult to read will be marked accordingly. Program text indentation is also an element of style and must be present. Significant constant, variable and structure names must be used. Marks are deducted on the basis of the frequency of these errors.
- **Version Control System [5%]** Track of changes that made to a program should be kept in a **private** Github repository using an IDE integrated version control system**.** The teaching assistant must be invited as the only collaborator to have an access to the repository. A link to the repository and the corresponding screenshots should be included in the submission.
- All files to be submitted should be placed in a single directory and zipped together into a single file for uploading to **UR Courses**.
- Your submission for each programming question includes: (a) the source code files (.CPP and .h files); (b) a Readme.txt file containing a brief explanation of each file along with a link to the

Github repository of the assignment; and (c) screenshots of testing runs and the Github repository.