Data Structure Assignment 3

Task 1 : Binary Search Tree (I/O: 10/10/10 points, coding style: 5 points)

Given an input number `n` and serial integers, please build a binary search tree and show the preorder, inorder and postorder traversals. While building the Binary Search Tree, the root is the first integer of the serial integers. If the upcoming integer **equals** to an existing node, put it on its **left (left-child)**. For other cases, put the smaller ones on left (left-child), bigger ones on right (right-child).

- Input Format:
  - The first line represents the amount of the input series. (a variable `n`)
  - Starting from the second line, each line represents a serial input to build a binary search tree.
  - All inputs end with a new line "`\n`".
  - The numbers in a serial input are integers ($-2^{31} < \text{number} < 2^{31} - 1$), and they are separated by a comma ("**,**").
- Output Format:
  - For each serial input, show its preorder, inorder and postorder traversals, respectively.
  - Take notice of the upper and lower cases of "`Preorder`", "`Inorder`" and "`Postorder`".
  - There's no white space before the colon ("**:**"), but one white space after it.
  - One white space between the output numbers, and no white space but a new line "`\n`" at the end.

Example:

Input	output
3 9,5,6,7,1,8,3 22,86,-5,8,66,9 45,3,5,3,8,6,-8,-9	Preorder: 9 5 1 3 6 7 8 Inorder: 1 3 5 6 7 8 9 Postorder: 3 1 8 7 6 5 9 Preorder: 22 -5 8 9 86 66 Inorder: -5 8 9 22 66 86 Postorder: 9 8 -5 66 86 22 Preorder: 45 3 3 -8 -9 5 8 6 Inorder: -9 -8 3 3 5 6 8 45 Postorder: -9 -8 3 6 8 5 3 45

Task2 : Huffman Tree (I/O: 25/20 points, coding style: 5 points)

Given an input number `n` and a serial input sentences, please implement the Huffman algorithm then output the Huffman code of each symbol and its compression ratio.

While building the Huffman tree by bottom-up method, please follow the rules below:

First, count the frequency of each character in all input sentences ordered by **`A to Z`** and then **`a to z`** (Ignoring `' '`, `'.'`, `','`, `'~'`, and so on). Second, put the counted frequencies to a priority queue in ascending order (if more two different characters have the same frequencies, order them by `A` to `Z` and then `a` to `z`). In this priority queue, **the smaller frequencies get the higher priorities**. So, pop the highest two priorities in this queue. The **highest one** is a **left-child**, and the **second highest** is a **right-child**. Sum them up as a new node and add it to the queue. Do it repeatedly and finally, you'll get a unique Huffman tree. The values of **left** branches are **0**, and the values of **right** branches are **1**.

- Input Format:
  - The first line represents the amount of the input sentences. (a variable `n`)
  - Starting from the second line, each input sentence is smaller than `1000` characters.
  - All the inputs end with a new line "`\n`".
- Output Format:
  - show the Huffman code of each symbol and a "`\n`" at the end.
  - The output ordered by the shortest Huffman code to the longest (`A` to `Z` and then `a` to `z` if having the same length).
  - Calculate its compression ratio rounded off to the second decimal place.
  - There's no white space before the colon ("`:`"), but one white space after it.

Example:

Input	output
3 I have a pen. I have an apple. UMMMM~ APPLE PEN.	M: 001 a: 110 e: 010 E: 0001 I: 0110 P: 1110 h: 0111 l: 0000 n: 1000 p: 1111 v: 1001 A: 10100 L: 10101 N: 10110 U: 10111 Compression ratio: 53.31%

[How to calculate compression ratio?]

Total bit length after encoding: 3*(4+5+4) + 4*(2+2+3+2+1+2+3+2) + 5*(1+1+1+1) = 127
Total bit length before encoding: 8*34 = 272
Compression ratio: 1 - 127/272 = 53.31%

Note:

- You must use C as your programming language.
- According to different algorithms, you'll get various Huffman trees. So, make sure you follow our rules to get the **unique Huffman tree** like the tree in "hw3-Huffman-tree.pdf".
- In addition, even with different Huffman tree algorithms, you'll get **the same compression ratio**, too.

[Good coding styles]

- A clear main() function with few nested structures (if-else)
- Meaningful variable names and no global variables
- No fake functions or declaring similar functions doing the same thing
- No warnings after compiling

Put the files below in the **folder** (folder name: studentID), and compress this **folder** as **"studentID.zip"**

1. Two source code file. (filename: studentID_1.c, studentID_2.c)
2. One **report** with your coding environment (OS, IDE, …), problems you encountered, and references (websites, friends who taught you, …). (filename: studentID.pdf) **(10 points)**

All the file names are correct, or you'll get zero points. **(5 points)**
Deadline: 2018/12/11 23:59. **You must hand in the assignment on time, or you will get zero points**.

Warning: We encourage you to discuss assignments with each other. However, you have the responsibility to finish the assignments individually. Do not copy others' assignment, or all of you will get zero points.