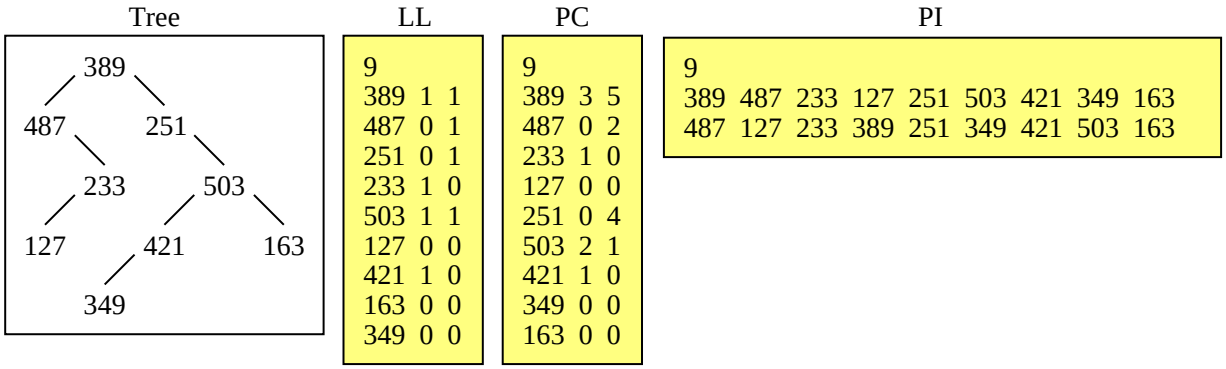
CS69001 Computing Laboratory – I

EVEN PC | **Lab Test 1** | **Date: 10–September–2018**

Part 1: C Program

Let T be a binary tree. Each node of T stores the following four items: an integer-valued **`key`**, two child pointers **`left`** and **`right`**, and an integer-valued **`size`** intended to store the number of nodes in the subtree rooted at that node (counting the node itself). The following figure shows three formats of storing T. For simplicity, assume that the keys stored in the nodes of T are distinct from one another.



LL This is the level-by-level and left-to-right-in-each-level format introduced in the assignments.

PC In this format, we store the keys in the preorder traversal sequence. Against each key, we additionally store the sizes of the left and the right subtrees of the node storing the key.

PI This format stores the preorder and the inorder sequences of the keys in the tree. One can uniquely reconstruct the tree (see below) from these two sequences.

Write a C program to perform the following tasks.

- Construct the binary tree T from a file **`C0.txt`** storing the tree in the PI format. Let P and I be the preorder and inorder sequences of keys in T. Let k be the key stored at the root of T. Finally, let P_L, I_L and P_R, I_R be the preorder and inorder sequences of the left and the right subtrees L and R. We have $P = k : P_L : P_R$ and $I = I_L : k : I_R$, where colon (:) denotes concatenation. Identify k from the first element of P. Locate k in I (if the keys of T are distinct, k can be found at a unique location). This gives the above decomposition of the input sequences. Create a root to store k. Recursively build the two subtrees from the subsequences. P_L, I_L and P_R, I_R. Write a function **`buildtree`** to solve this part.
- During the construction of T, keep the **`size`** fields uninitialized at all the nodes. After the construction is done, populate the **`size`** fields of all the nodes by a recursive traversal of the tree. Write a function **`populatesizes`** for this part.
- Write a function **`printPC`** to print the tree in the PC format.

Sample

The input and output formats are specified in the above figure. The input file **`C0.txt`** has three lines. The first line stores the number n of nodes in the tree T. The second line stores the preorder listing of the n keys in T. Finally, the third line stores the inorder listing of the n keys in T. Assume that the file stores valid listings of the keys in a binary tree, that is, your program does not have to check for the validity of the input.

Print the output to screen (not to a file). In the first line, print n. This is followed by n lines, each printing a triple consisting of a key and the sizes of the left and the right subtrees. The keys should appear in the preorder sequence.

Part 2: Python Program

A date is specified in the US format as *Mmm dd, yyyy*. Here, *dd* is a two-digit date, *Mmm* is the three-letter abbreviation of a month with (only) the first letter in upper case, and *yyyy* is a four-digit year (assume that $1800 \leqslant yyyy \leqslant 2018$). Some examples of dates specified in this format are Aug 15, 1947 and Sep 09, 2018. You are given an input file **`P0.txt`** storing a list of dates in the US format, one in each line. The number of dates does not appear explicitly in the file.

Write a Python program to do the following tasks. Read the dates in the input file **`P0.txt`**, and sort the dates chronologically. Print the dates (to screen) in the chronologically sorted order in the US date format.

Your program does not have to check the validity of the input dates. That is, assume that the input file does not contain invalid dates like Feb 29, 2018, Shr 22, 1941, or Sep 12, 1752 (what is wrong with the last date, apart from that the year is < 1800?). Assume also that all the dates are provided in the *Mmm dd, yyyy* format, that is, ill-formatted dates like June 07, 1889, 01/21/1992, May-16-1857, or DEC 24, 2010 do not appear in the input file. The input file may contain duplicate dates. Print all the dates, that is, you must not remove duplicates.

Do *not* use *any* external module (like **`datetime`** or **`calendar`** for manipulating dates). You may, however, call the built-in sorting function of Python.

Sample

Input	Output
Nov 09, 1836	Dec 11, 1834
Apr 26, 1995	Nov 09, 1836
Jul 12, 2017	Mar 26, 1845
Jul 10, 1845	Jul 10, 1845
Apr 01, 1994	Nov 05, 1858
Dec 11, 1834	Sep 26, 1864
Nov 08, 2015	Nov 21, 1881
Sep 09, 2016	Dec 23, 1899
Jul 19, 1940	Nov 28, 1908
Nov 21, 1881	Jul 19, 1940
Oct 30, 1945	Oct 30, 1945
Aug 22, 1972	Aug 28, 1962
Nov 28, 1908	Jul 08, 1967
Jul 08, 1967	Aug 22, 1972
Dec 23, 1899	Apr 01, 1994
Nov 05, 1858	Apr 26, 1995
Mar 26, 1845	Jun 29, 1996
Sep 26, 1864	Nov 08, 2015
Aug 28, 1962	Sep 09, 2016
Jun 29, 1996	Jul 12, 2017

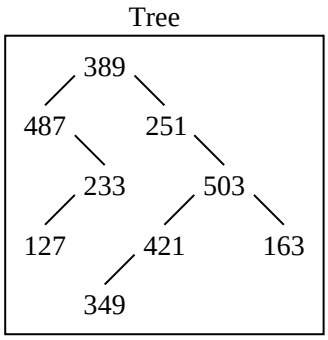
Submit one C/C++ file and one Python file. Write your name, roll no, and PC no as comments in each file.

CS69001 Computing Laboratory – I

ODD PC | **Lab Test 1** | **Date: 10–September–2018**

Part 1: C Program

Let T be a binary tree. Each node of T stores the following four items: an integer-valued **`key`**, two child pointers **`left`** and **`right`**, and an integer-valued **`size`** intended to store the number of nodes in the subtree rooted at that node (counting the node itself). The following figure shows three formats of storing T. For simplicity, assume that the keys stored in the nodes of T are distinct from one another.



LL

```
9
389  1  1
487  0  1
251  0  1
233  1  0
503  1  1
127  0  0
421  1  0
163  0  0
349  0  0
```

PC

```
9
389  3  5
487  0  2
233  1  0
127  0  0
251  0  4
503  2  1
421  1  0
349  0  0
163  0  0
```

IP

```
9
487 127 233 389 251 349 421 503 163
127 233 487 349 421 163 503 251 389
```

LL This is the level-by-level and left-to-right-in-each-level format introduced in the assignments.

PC In this format, we store the keys in the preorder traversal sequence. Against each key, we additionally store the sizes of the left and the right subtrees of the node storing the key.

IP This format stores the inorder and the postorder sequences of the keys in the tree. One can uniquely reconstruct the tree (see below) from these two sequences.

Write a C program to perform the following tasks.

- Construct the binary tree T from a file **`C1.txt`** storing the tree in the IP format. Let I and P be the inorder and postorder sequences of keys in T. Let k be the key stored at the root of T. Finally, let I_L, P_L and I_R, P_R be the inorder and postorder sequences of the left and the right subtrees L and R. We have $I = I_L : k : I_R$ and $P = P_L : P_R : k$, where colon (:) denotes concatenation. Identify k from the last element of P. Locate k in I (if the keys of T are distinct, k can be found at a unique location). This gives the above decomposition of the input sequences. Create a root to store k. Recursively build the two subtrees from the subsequences. I_L, P_L and I_R, P_R. Write a function **`buildtree`** to solve this part.
- During the construction of T, keep the **`size`** fields uninitialized at all the nodes. After the construction is done, populate the **`size`** fields of all the nodes by a recursive traversal of the tree. Write a function **`populatesizes`** for this part.
- Write a function **`printPC`** to print the tree in the PC format.

Sample

The input and output formats are specified in the above figure. The input file **`C1.txt`** has three lines. The first line stores the number n of nodes in the tree T. The second line stores the inorder listing of the n keys in T. Finally, the third line stores the postorder listing of the n keys in T. Assume that the file stores valid listings of the keys in a binary tree, that is, your program does not have to check for the validity of the input.

Print the output to screen (not to a file). In the first line, print n. This is followed by n lines, each printing a triple consisting of a key and the sizes of the left and the right subtrees. The keys should appear in the preorder sequence.

Part 2: Python Program

A date is specified in the EU format as *dd-Mmm-yyyy*. Here, *dd* is a two-digit date, *Mmm* is the three-letter abbreviation of a month with (only) the first letter in upper case, and *yyyy* is a four-digit year (assume that $1800 \leqslant yyyy \leqslant 2018$). Some examples of dates specified in this format are 15-Aug-1947 and 09-Sep-2018. You are given an input file **`P1.txt`** storing a list of dates in the EU format, one in each line. The number of dates does not appear explicitly in the file.

Write a Python program to do the following tasks. Read the dates in the input file **`P1.txt`**, and sort the dates chronologically. Print the dates (to screen) in the chronologically sorted order in the EU date format.

Your program does not have to check the validity of the input dates. That is, assume that the input file does not contain invalid dates like 29-Feb-2018, 22-Shr-1941, or 12-Sep-1752 (what is wrong with the last date, apart from that the year is < 1800?). Assume also that all the dates are provided in the *dd-Mmm-yyyy* format, that is, ill-formatted dates like 07-June-1889, 21/01/1992, 16-05-1857, or 24-DEC-2010 do not appear in the input file. The input file may contain duplicate dates. Print all the dates, that is, you must not remove duplicates.

Do *not* use *any* external module (like **`datetime`** or **`calendar`** for manipulating dates). You may, however, call the built-in sorting function of Python.

Sample

Input	Output
04-Mar-1970	03-Jul-1827
12-Apr-1830	12-Apr-1830
27-Jun-2014	15-Mar-1841
03-Jun-1882	16-Mar-1881
03-Jul-1827	03-Jun-1882
30-Mar-2014	28-Sep-1890
07-Mar-1910	24-Aug-1899
31-Oct-1919	22-Mar-1902
22-Mar-1902	07-Mar-1910
15-Mar-1841	31-Oct-1919
30-Jan-2015	17-Feb-1922
26-Dec-1926	26-Dec-1926
24-Aug-1899	02-Apr-1931
10-Jan-1952	10-Jan-1952
17-Feb-1922	04-Mar-1970
18-Aug-1977	18-Aug-1977
16-Mar-1881	01-Sep-1978
28-Sep-1890	30-Mar-2014
02-Apr-1931	27-Jun-2014
01-Sep-1978	30-Jan-2015

Submit one C/C++ file and one Python file. Write your name, roll no, and PC no as comments in each file.