Shell Programming

A certain railway company in Hong Kong uses a signaling system to keep track of trains in its railway system. The system generates a log file (.txt) every day. Each line in the log file represents a train calling at a station and has the following format:

```
[time_HH:MM:SS], [train_id], [station_id], [paltform_no.]
```

Each column is separated by a comma "," in the log file.
The log files have "Trains_" as prefix in their file names, followed by a date in form of YYYY-MM-DD.
The records are sorted by time in ascending order

Field	**Description**
time_HH:MM:SS	The time has the format HH:MM:SS
train_id	Identify a train Train ID starting with "E" represents a passenger train. (e.g E201, E217) Train ID starting with a number represents a maintenance train. (e.g. 8001, 59)
station_id	Identify a station
platform_no.	The platform number in that particular station

Example:
Consider an example *Trains_2022-09-01.txt* that stores the records on 2022-09-01.

```
06:01:10,E201,SHS,1
06:06:28,E209,SHS,1
...
07:10:27,8001,MOK,2
...
```

From the log file, we know that the E201 passenger train called at SHS station platform No.1 at time 06:01:10.

Problem 1:

Create a shell script ***1.sh*** that performs the following:

1. For each "Trains_" log file, find out the three busiest stations that have the most number of **passenger trains** calling at. The output should display the count of passenger trains that called at that station and the station ID.
2. If two or more stations have the same count, we output them in alphabetical order of the station ID. Therefore "4 FAN" is put ahead of "4 HHK" under "Trains_2022-09-03.txt" in the example below.

If we run ***1.sh***, the following will output. The sample output can be found in the file ***output1.txt***.

```
Trains_2018-09-01.txt:
     25 LMC
     23 TWO
     22 FAN
Trains_2018-09-02.txt:
     30 SHS
     24 LMC
     24 SHT
Trains_2018-09-03.txt:
      4 FAN
      4 HHK
      4 KOT
Trains_2018-09-04.txt:
     17 SHS
     11 MOK
      9 FAN
```

Problem 2:

Create a shell script ***2.sh*** that performs the following:

1. The *2.sh* takes one command line input argument, which represents the train ID of a train that we would like to trace.
2. The *2.sh* generates a file *<train_id>.txt* that contains a list of stations called by the train in all "Trains_" log files, organized by the filenames and the records in the order of Date and Time.
3. If the number of input arguments is not 1, then the error message "Usage: ./Train_trace.sh <train ID>" should be output in shell prompt.
4. If there is no train record found, then the log file will be deleted (or no need to generate it). The script should outputs "No records found for train *id*" in shell prompt. (where *id* is the train ID inputted by the user).

For example, If we run:

```
 $ ./2.sh E201
```

The shell script will generate a file "E201.txt" that contains the following contents:

```
Trains_2022-09-01.txt
06:01:10,E201,SHS,1
06:19:13,E201,FAN,2
06:48:48,E201,FTR,3
…
Trains_2022-09-02.txt
06:00:50,E201,SHS,1
06:23:40,E201,HHK,3
…
23:15:24,E201,SHS,1
Trains_2022-09-03.txt
Trains_2022-09-04.txt
```

Sample Test Cases:

2_1

```
$ ./2.sh E201
E201.txt will be generated. Please refer to the files given.
```

2_2

```
$ ./2.sh E209
E209.txt will be generated. Please refer to the files given.
```

2_3

```
$ ./2.sh E213
E213.txt will be generated. Please refer to the files given.
```

2_4

```
$ ./2.sh E229
E229.txt will be generated. Please refer to the files given.
```

2_5

```
$ ./2.sh 8001
8001.txt will be generated. Please refer to the files given.
```