EndSem Lab Exam 2

Course code: EndSem_Lab_Exam_2

Assessments code: set2

To download the asmts, run: autolab download EndSem_Lab_Exam_2:set2

Max Marks: 180

Duration: 3 hours, NO extension.

Problem 1: Parentheses Generator Machine (50 points)

Write a program that generates parentheses strings of a given length in lexicographic order satisfying a particular condition based on input parameters. Each type of condition has a specific weightage, as described below.
Parentheses (()):

Parentheses are a pair of symbols:
(is the opening parenthesis.
) is the closing parenthesis.

Lexicographic Order:

Lexicographic order is the dictionary order based on ASCII values. For parentheses:
(has an ASCII value of 40.
) has an ASCII value of 41.

Strings are sorted such that (comes before) at any position.
Example of Lexicographic Order:
For strings of length 2(n=1):

```
((
()
)(
))
```

Input Format:

n: Number of pairs of parentheses.
flag: An integer (1, 2, or 3) that specifies the type of output:
1: Generate all possible parentheses strings of length 2n in lexicographic order. Weightage: 20%.
2: Generate only valid parentheses strings (correctly nested strings) in lexicographic order. Weightage: 40%.
3: Generate only invalid parentheses strings (incorrectly nested strings) in lexicographic order. Weightage: 40%.

Output Format:

For flag = 1: Output all possible strings of length 2n in lexicographic order.
For flag = 2: Output only valid strings where parentheses are balanced and properly nested.
For flag = 3: Output only invalid strings where parentheses are unbalanced or improperly nested.

Definitions

What is Valid?

A parentheses string is valid if:
Every opening parenthesis (has a matching closing parenthesis).
At no point does the number of) exceed the number of (when reading left to right.
Examples of Valid Strings:

```
()
(()())
((()))
```

What is Invalid?

A parentheses string is invalid if:
There are unmatched parentheses (too many (or)).
Closing parentheses) appear before a matching opening parenthesis (.
Examples of Invalid Strings:

```
())
)(
(((
```

Example:

Input

```
2 1
```

Expected Output:

```
(())
()()
())(
)(()
)()(
))((
```

Input

```
2 2
```

Expected Output:

```
(())
()()
```

Input

```
2 3
```

Expected Output:

```
())(
)(()
)()(
))((
```

Constraints:

- n can be upto 10
- flag can be 1, 2, or 3

Problem 2: Professor X's Gradebook Manager (60 points)

Professor X has been managing the grades for all his students for years. But even with his amazing abilities, he finds himself overwhelmed with the task of keeping track of student grades in an ever-growing class. Unfortunately, his telepathic powers don't extend to organizing his gradebook, and that's where you come in! The professor has delegated the responsibility to you to manage the grades for his students.
Your task is to design a system to add, modify, and delete student grades based on commands. You'll need to manipulate a file that contains student records, ensuring that only valid operations are carried out on the gradebook.

You are given a file containing records of students with their corresponding grades. Each student is identified by a unique student ID. The operations you need to support include adding a new record, modifying an existing record, and deleting a record. At the end of the operations, you will also need to compute and display the maximum, minimum, and average score for the class.

Also Sort the records based on student_id and store it in the same text file. 50 marks for correct answer and 50 marks for storing correctly, an incorrect answer will be a direct zero even if file is correct.

Input Format:

- A file called `students.txt` that contains records of student IDs and their grades. The format of each line in the file is:

  ```
  student_id marks
  ```

- Where `student_id` is a unique identifier for each student, and `marks` is the mark for the student.
- On the terminal you will be given an integer `n` representing the number of queries and then n lines of queries.
- Each Query is of the form `op_id student_id` and may also have a third argument `marks` here `op_id` represents one of the following operations
  - `1 student_id marks` - Adds a new student record if the student ID doesn't already exist.
  - `2 student_id marks` - Modifies the grades of an existing student, identified by their student ID.
  - `3 student_id` - Deletes the record of the student identified by the student ID.

  Here 1 , 2 , 3 are the op_id

Operations to Perform:

- **ADD:** If the student ID does not exist in the gradebook, the record should be added. If the student ID already exists, do nothing.
- **MODIFY:** If the student ID exists, update the grades for the student. If the student ID does not exist, do nothing.
- **DELETE:** If the student ID exists, remove the student record. If the student ID does not exist, do nothing.

Output Format:

After all operations have been executed, print the maximum, minimum, and average score for each student in the format:

```
max_score_for_class min_score_for_class average_score_for_class
```

Example:

Initial File: `students.txt`

```
1001 85
1002 92
1003 76
```

Operations:

```
3
1 1004 88
2 1002 95
3 1003
```

Expected Output:

```
95 85 89.33
```

Final File: `students.txt`

```
1001 85
1002 95
1004 88
```

Average should be capped off at 2 decimal points.

Constraints:

- 1 ≤ `n` ≤ 100 (Number of operations)
- 0 ≤ `marks` ≤ 100 (Each grade is a positive integer)
- 1 ≤ `no_students` ≤ 500 (Number of students)

Important Notes:

- The student IDs in the `students.txt` file are unique, but may be modified during the operations.
- After performing all the operations, display the final results showing the maximum, minimum, and average scores for each student still in the file.
- If an operation is invalid (e.g., trying to delete a student who doesn't exist), the system should simply ignore that operation.
- Ensure the results are printed with a precision of two decimal places for the average score.
- Save all changes made to students.txt in the same format as input this will also be checked.

Problem 3: Contest System Management (70 points)

You are tasked with managing the users and their ratings for an online contest system. The system must support the following operations: adding a new user, updating an existing user's rating, deleting a user, viewing a user's information, and printing the top-rated user.

Each user is identified by a unique username, and their performance is represented by a rating. You will need to read data from a log file containing operations on users and perform the corresponding actions.

Input Format:

- A file called `file.txt` that contains logs of user operations. Each log line corresponds to a specific operation in the following format:

  ```
  action [username] [rating]
  ```

  [username] and [rating] represents they may or may not be present.

- `action` can be one of the following:
  - `Add` – Creates a new user with the username and a default rating of 800 if the username does not already exist.
  - `Update` – Modifies the rating of an existing user by adding the given rating value to their current rating.
  - `Delete` – Deletes the user with the given username.
  - `View` – Displays the username and current rating of the specified user.
  - `TopCoder` – Prints the name of the user with the highest rating. If multiple users have the same rating, the user who was created first is printed.

Operations to Perform:

- **ADD:** Adds a new user with the given username and default rating of 800.
- **UPDATE:** Updates the rating of an existing user by adding the given rating.
- **DELETE:** Deletes the user with the specified username.
- **VIEW:** Prints the username and rating of the specified user.
- **TOPCODER:** Prints the username of the highest-rated user.

Important Notes:

- Usernames are case-sensitive, so `Priet` and `priet` are different.
- If the log contains a `TopCoder` operation and there are no users, print `-1`.
- If the log contains a `Delete` operation on a user that doesn't exist, print `-1`..
- If the log contains a `View` operation on a user that doesn't exist, print `-1`.
- If the log contains an `Update` operation on a user that doesn't exist, print `-1`.
- Refer to the given examples for more clarity.

Additional Notes:

- The program must print the highest-rated user based on the total rating and should print the first added user if there are ties.
- Print all output results as specified (with user ratings where applicable, and top-rated user).

Constraints:

- $0 \leq n \leq 1000$ (Number of operations)
- $1 \leq username \leq 50$
  (Length of the Username)
- $0 \leq users \leq 1000$ (Number of users in the system)

Example 1:

Initial File: `file.txt`

```
Add Priet
Update Priet -12
View Priet
Update Priet 100
View Priet
Delete Priet
View Priet
TopCoder
```

Expected Output:

```
Priet 788
Priet 888
-1
-1
```

Example 2:

Initial File: `file.txt`

```
Add Priet
Update Priet -12
Add CproTA
Update Priet 100
Update CproTA 88
View Priet
TopCoder
Delete Priet
Update priet 1000
View Priet
TopCoder
```

Expected Output:

```
Priet 888
Priet
-1
-1
CproTA
```

Submission Guidelines

Do not rename any files given in the handout. Only write the code in the specified C files in the respective directories.