Email Manager Practice
Report Document

Practice Group Code: AN NIAs of group members: 217447 217223 194985
Teacher of Practice Class: Sergio Ivan Giraldo

The submission must include the code but also a report addressing the questions and issues indicated in this section. You can answer the questions directly here and deliver this same document (modifying the filename for submission and grading management) or build a different report document containing the requested information.

Remember to put the group identifier in all files submitted both inside the file and in the filenames.

Part 1: Design of the linked lists

The emails are implemented as a linked list, and the references of emails in the folders too.

1.1 Decide and JUSTIFY what abstract data type (stack, queue, or general list) you need to define for the list of emails and the list of email references in the folders. If possible, try to match functional requirements to have the same implementation for both. Justify all your reasoning.

Es una lista general. Aunque añadimos elementos al final de la lista, cuando tenemos que eliminar un elemento la recorremos entera en busca de coincidencias. La lista de referencias de las folders funciona de igual manera que la lista de mails, y por lo tanto también es una lista general. La única diferencia es que cuando diseñamos una nueva linked list con elementos ya existentes, python por defecto nos pondrá punteros de los objetos. Sería interesante desde un punto de vista funcional poder insertar elementos a cualquier altura, sobretodo desde un punto de vista de eficencia en tiempo a la hora de recorrerla, pero para los requerimientos de la práctica estimamos que era más practico que añadir elementos se comportara como una pila.

1.2 What header do you need in the selected ADT for the lists above? Be precise what information you include in the header and what type it is and WHY it is needed.

Utilizamos dos atributos. El primero es el head. Esto nos sirve para controlar el inicio de la lista, para saber por donde empezar a recorrerla. Luego tenemos el size. Este atributo sirve para controlar tres casos críticos. El primero es el de añadir un elemento a una lista vacía, puesto que tendremos que declarar como head al elemento entrante. De la misma forma, tenemos que impedir que se intente eliminar un elemento de una lista vacía por razones obvias. Por último, el size se encarga de controlar que al eliminar un elemento de una lista con sólo un elemento el head se reinicia a None, puesto que si no nunca podríamos quedarnos con una lista vacía.

Otros atributo que pensamos en implementar es el back, para controlar el final de la lista. Decidimos que era mejor prescindir de él por la sencilla razón de que implementamos una lista simple. Aunque las dos cualidades no estén estrictamente relacionadas, encontramos más coherente desde un punto funcional implementar la lista de esta forma. La razón es la siguiente. La estructura de linked list debía ser lo más general posible. Entonces es posible que en un futuro se le de un uso diferente que al de una lista de mail (una lista de contactos, de direcciones de correo…). Y quizás en esos escenarios nos interesa recorrer la lista en busca de que no hayan coincidencias desde dentro de la misma clase. Entonces, en el caso de nuestra función append (que recorre la lista entera desde el head hasta que no encuentra node.next) solo tendría que añadir una línea para efectuar el cotejo en el mismo código. No es así en el caso de trabajar con back. En el caso de una función append que añada elementos directamente al back, tendríamos que rediseñar por completo la función si decidimos que hace falta recorrer la lista, y de ahí deriva nuestra decisión

The linked list needs to be single-linked list or double-linked list? Justify why.

No hay una respuesta clara para esta pregunta. Las dos opciones ofrecen alternativas cuanto menos plausibles. Al final, nos decantamos por la single list por el simple motivo de que no encontramos razones de peso suficientes para implementar una lista doble si el objetivo era ajustarnos a los requerimientos que se pedían desde el handout. El echo de implementar una lista doble cargaba de punteros la lista, y complicaba con más líneas de código el algoritmo que se encarga de añadir y eliminar elementos. Así pues, decidimos primar la eficacia y la simplicidad.

1.3 Indicate the exact functions you need to implement this linked list ADT designed above. Justify the need for each of them. Be careful to use the appropriate name according to the ADT theory chapter and to justify the functionality, arguments and return values for each of the functions you plan to implement.

Empezaremos haciendo referencia a las funciones que nos devuelven atributos. Accedimos a ellas tanto desde dentro de la clase como desde fuera, y sirven para obtener algun dato que nos haga falto sin acceder a la propiedad directamente.

1. __len__: Devuelve el atributo size de la lista. Sirve para vertebrar la función is_empty

2. Is_empty: Para saber si la linked list está vacía o no. Sirve para controlar casos clave cuando añadimos o eliminamos elementos, como hemos explicado en el apartado 1.2

3. Get_head: Devuelve el head de la lista. Sirve para empezar a recorrerla, ya sea desde dentro de la linked list (para hacer una append) o desde fuera (para encontrar un match con get_email de la database).

Luego tenemos las funciones que se encargan de añadir o quitar elementos a la linked list. Son necesarias para poder modificarla cuando añadimos emails o los quitamos, ya sea la general de mails o la de folders.

4. Append: Añade elementos al final de la lista. Para añadir elementos correos tanto cuando lees del EMConfig como cuando los creas manualmente desde el main

5. Remove: Eliminaa elemementos de la linked list. Para eliminar correos tanto de folders como de la lista principal.

6. Clear: Reseteaa la lista. Para dejarla en blanco cuando eliminamos el último elemento

Por último, hemos implementado la función string(__str__). No se usa en la versión final, pero se usó durante la implementación de la linked list para comprobar que funcionaba correctamente, y por eso la hemos dejado escrita.

Cabe decir que nosotros no usamos algunas de las funciones que se nos dan en el marco porque no las estimamos necesarias. Concretamente pop, insert e index. Esto es debido a que debido a nuestra estructura de datos, nunca debemos insertar elementos a mitad de la linked list o eliminar elementos en base a su posición en la lista.

Part 2: Design of the submitted code (graphical representations)

You can include the figures in the document or have separate files with the figures. If they are separate files indicate the name of the files included and what each one contains.

2.1. Provide a graphical representation of the data structure used by the program. Make sure the drawing adds additional graphical representation than a mere list of items. So that there are boxes and arrows that show the content and relations between them to provide a conceptual representation of the information stored. Make sure you distinguish with different color (or by other means) what is included in the skeleton and what it is designed (added/deleted/modified) by you.

Como imagen en la carpeta. Todas las clases que usamos estaban incluidas en el esqueleto y por lo tanto no hay modificaciones respecto al original.

2.2. Provide a conceptual explanation of the above data graphical representation. Clearly identify and justify what is relevant and important in this data structure design. Note: Do not use an exhaustive list of items giving the comments in the code as this does not complement and add anything to the code.

Tenemos clases. La clase database config se encarga de almacenar los metadatos sobre la base de datos, como el directorio o la extensión de los archivos. Database por su lado se encarga de Todas las operaciones relacionadas con la modificación de archivos virtualmente. Tanto de carpetas como de las linked list de correos. Además, inicializa el diccionario de carpetas y la linked list de correos, y contiene la db_config como parámetro. Linked list se encarga de almacenar los datos de forma seqüencial, dentro de nodos. Por último, folders tan solo se encarga de guardar las linkeds lists, y el nombre de la misma.

2.3. Provide a graphical representation of the structure of the program you submit. The diagram has to have the functions/methods (boxes) and their interactions (arrows). A useful representation has functions connecting to several others (instead of having several copies of the same function to have a linear drawing). This diagram has to be useful to conceptually explain everything that the program does and it has to be visible graphically. Make sure you distinguish with different color (or by other means) what is included in the skeleton and what it is designed (added/deleted/modified) by you.

Como archivo de imagen en la carpeta de la entrega. No hemos contemplado todas las relaciones entre funciones para no entorpecer la vista del diagrama (que ya de por sí es complejo), sino que tan solo hemos recogido las interacciones más importantes y el tipo de información que intercambian,

2.4. Provide a conceptual explanation of the above graphical representation of the program structure. Clearly identify and justify what is relevant and important in the design. Note: Do not use an exhaustive list of function with the text of the doc-strings in the functions as this does not complement and add anything to the code. The explanation in here has to be on functionality independently in what programming language is implemented (hence independently of the actual lines of code)

Lo primero que hacemos es cargar la Database config. Una vez eso está echo, crearemos la database y la inicializaremos mediante la función de útils load database. A partir de ahí, desde el menú del usuario vamos accediendo a las diferentes opciones que nos proporciona el menú, y si vamos siguiendo las flechas vamos viendo como las funciones se comunican entre sí para ir hilando el programa.

2.5. Explained how you worked as a team and how you distributed the work to individually advance in parallel and what was the order things were worked out. Explain how efficient you think you were in this team approach.

La principal mejora respecto a la última práctica ha sido el uso de git para la distirbución del trabajo. Concretamente, coordinarnos con GitHub ha permitido que trabajemos de forma paralela en el código. El trabajo se distribuyó en tres grandes bloques iniciales. La modificación de archivos virtuales desde la database, la modificación de archivos físicos a través de útils y por último la implementación de la linked list. Una vez eso estuvo hecho, pasamos a una implementación del main y al control de errores de manera conjunta. El trabajo en equipo ha funcionado de manera eficiente.

Part 3: Code Implementation and Verification

3.1. Explain the necessary implementation details in your program. Just mention the particular parts/algorithms that requires to explain the way it is implemented. Do not repeat description of code that does not add anything to what it is explained in the design above.

De entre todas las funciones implementadas, solo cuatro de ellas merecen ser comentadas específicamente debido a su complejidad. Para empezar, dentro de la función create_email tenemos el siguiente algoritmo para crear el body del mensaje

```
# This one is tricky. We check at every input if it contains the secuence "end" at the end. We keep adding lines of body until we found it,
# and in that moment we add all the line but the word and we finish the loop.
while stri[(len(stri)-end_chars):] != "end": #This formula only looks for the word end at the end of the line
    stri = input("")
    if stri[(len(stri)-end_chars):] != "end":
        stri += ("\n")
        body += stri

    else:
        body += stri[:(len(stri)-end_chars)]
```

Usamos la fórmula *while stri[(len(stri)-end_chars):]* para cortar la string y quedarnos solo con los tres últimos caracteres, que serán usados para saber si seguimos escribiendo el body o no y para saber si añadimos esos mismos caracteres al body o no.

Después, la función lista dentro del mismo main.

```python
def lista (db,folder_name = None):
    """
    Show the list of email contained either in the Database or in a folder.

    :param db: An email database
    :param folder_name:  The folder where it looks. If its None, then it shows the list of the mails in the entire database
    """
    contador = 1 #For the numbering at the start of each string
    pad = 50 # For a better visualization of the list
    secondpad = pad*2 # The second padding is at the double of distance than the first
    space = " " #For adding spaces when padding

    if db.get_email_ids(folder_name) != []:
        for email_id in db.get_email_ids(folder_name):
            string = "" #The line we are going to pring
            to_print = db.get_email(email_id) #We will use some email properties that will be displayed at the list

            string += ("%s. Sender: %s" % (contador,trim(to_print.sender)))
            string += (pad - len(string)) * space
            string += ("Subject: %s" % (trim(to_print.subject)))
            string += (secondpad - len(string)) * space
            string +=  ("Date: %s" % (to_print.date))
            print(string)

            contador += 1 #To the next item!

    else: # We do not display empty lists
        print("No items in this list yet!")
```

Esta función debe su complejidad a lograr el componente estético en el display del menú. Pad y second pad sirven para controlar el espaciamiento entre las diferentes columnas en la visualización de los menús. Miramos todas las columnas en busca de strings demasiado largas que nos descuadren el menú y los acortamos mediante la función trim, que utiliza el método que hemos usado para eliminar el "end" del final del body. Luego solo falta ir calculando espacios en blanco entre columnas con el padding y ir pasando el contador para crear la numeración,

Luego, la función remove de la linked list.

```python
def remove(self, item):
    """
    Remove from the list the first occurrence of item.
    Raises ValueError if there is no such item.

    :param item: object(email) to be removed from the linked list.
    """
    #Control de entrada:
        # Esta función recibe un EMAIL QUE YA ESTÉ EN LA LINKED LIST

    #Control de errores:
        # Esta función controla la reducción del size al añadir un elemento
        # Esta función no valora si el nodo tiene formato correcto o no, solo si es de la classe Node
        # Esta función compara el mismo Email. Eso quiere decir que no podemos crear un nuevo Email y tratar de compararlo
        # aun cuando tengas los mismos atributos porque nunca encontrará coincidencia.

    current = self.get_head()

    if current.data == item: #If the item matches with the first one, we change the head of the linked list for its second element (accesseb via .next)
        self.head = current.next
        self.size -= 1

    else:
        while current: #We go through the entire linked list until we find the break
            if current.next: #We only enter if current.next exists.  In each iteration we look at the next one, so there is no problem with missing the last one element because we check it in
                #the iteration before
                if current.next.data == item: #When comparing nodes with emails, we acces to the .data atribute
                    current.next = current.next.next #We just ignore the element with this line
                    self.size -= 1
                    if self.size == 0:
                        self.clear() #If we emptied the list, we reset it (the header atm)
                    break
            current = current.next

    if not current: #If current == next it means no matches were made. Error!
        raise ValueError("Email not found...")
```

Esta función diferencia dos casos. Si tenemos un match en el head de la linked list, debemos de cambiar el head al nuevo elemento. Si no, recorremos la linked list buscando coincidencias y cuando la encontramos, simplemente ignoramos el elemento cambiando el atributo .next del nodo para que sea .next.next, pasando el elemento y dejandolo sin asignación en la linked list. Si la lista pasa a tener 0 elementos, la reiniciamos con self.clear. Por último, si current acaba siendo None, quiere decir que no hemos encontrado ninguna coincidencia, y por lo tanto hacemos saltar el error.

Por último, la función más compleja del programa es la encargada de leer la base de datos, load_database (no incluímos una foto aquí por ser demasiado extensa). Esta función era relativamente más simple, pero ha ganado tamaño debido al control de errores. La parte a explicar es la siguiente: Dos booleanos controlan que estamos haciendo en todo momento. Writing_folders comprueba si estamos entrando nuevas carpetas y writing_emails comprueba si estamos entrando mensajes dentro de una carpeta. Estos valores se activan si leen las palabras claves que marcan la cabecera de los diferentes apartados, y se desactivan si leen un espacio en blanco. Si estamos escribiendo carpetas, llamaremos a la función create_folder y si estamos escribiendo mails, llamaremos a la función add_email. (cargandolo previamente como objeto de clase email mediante la otra función de útils load email). Todo lo demás es control de errores, y se explica debajo en su apartado correspondiente.

3.2. Explain your Code verification approach. Describe how you have split the program in different parts and how you have verified each of the parts. Be concrete describing what mechanisms you have used and what conditions you have tested in each part. Provide and explain how to use any material (.c .ipynb files or particular config files or msg files different than the provided ones) we need to reproduce the same tests as you. Also, and as example, if there is any means to trace the execution turning on/off prints/logs, indicate how to do it and what it produces and how to interpret the traces…

La verificación del código es lo que más tiempo nos ha consumido. Hemos tenido una aproximación muy secuencial, de función a función, y no hemos dejado que saber el input condicione el control de errores de una función o clase. Hemos seguido una máxima para todo el recorrido de verificación de código: Eficiencia, luego robustez, luego simplicidad. Eso quiere decir que hemos primado la simplicidad de código para que sea fácil de interpretar y entender menos cuando significaba que renunciábamos a robustez del programa. Y por encima de estos dos ítems, hemos intentado que el programa sea eficiente en la consecución de sus funcionalidades. A partir de esta premisa, nos disponemos para enteder la verificación de código de clase a clase.

En el main, buena parte del control de errores viene ya controlado por las funciones preestablecidas. Eso contempla controlar el rango en las entradas de enteros y que la elección de carpetas y mails venga restringida. A parte de eso, solo nos preocupaba que debido a cómo habíamos implementado las funciones dentro de útils, el uso de una palabra como “End” como nombre de carpeta pudiese hacer fallar a la funcion de load database. Por eso, creamos una función llamada restricted que se encarga de controlar que no se usen todas esas palabras que puedan ser porblemáticas como nombres de carpeta o como datos del header de los mensajes.

En la linked list, teníamos un propósito general. Que fuese lo más general posible. Eso evitó que dentro de la linked list se hiciesen comprobaciones como que si un elemento ya estaba dentro de la linked list no lo añadiese. Al final, fuimos sacando comprobaciones de la linked list hasta dejar sólo un elemento de control: no podemos eliminar un elemento que no está en ella o si la lista está vacía.

Folders es una clase puente, y por como la hemos definido no controla ningún tipo de error. Todo lo que hace y recibe ya viene controlado, y folder solo ejerce como carpeta sin funcionalidades especiales.

Útils por otro lado, ha pasado por un control de errores bastante extenso. Empezamos con las funciones de write y de load database. Write database no controlará errores por el simple echo que lo que hace es volcar toda la database virtual en un archivo de texto. Así pues, si controlamos los errores en la database virtual (y en el load database) no hará falta controlar los mismos errores aquí. Load database en cambio si controla que el archivo EmailDB esté configurado correctamente. Tenemos un except general que recoje errores de formato, como no respetar los espacios en blanco, y que aunque de poca información sobre lo que esté pasando si que te da pistas por el momento en el que salta. Por ejemplo si no hay un salto de línea entre dos carpetas de mensajes, intentará leer el siguiente nombre de carpeta como carpeta y no lo encontrará. Luego de manera concreta, controlemos que estén las dos carpetas importantes “Inbox” y “OutBox”, que los mensajes estén listados en el mismo orden en que aparecen las carpetas (y que estén todas las carpetas como [carpeta] Messages:). Luego tenemos la función auxiliar de slice que no controla nada por su simplicidad. Por último, tenemos las funciones que trabajan con los mails. Write_email puede fallar solo en el caso de que el directorio EmailDB desaparezca en medio de la ejecución o que la database configuration se desconfigure. Ambos escenarios son muy poco probables, pero los recojemos en un except. Delete_email puede fallar solo si el mail no se encuentra físicamente y también lo recojemos. Por último, load email tiene solo una comprobacion. Si no encuentra el mail. Porque como podemos recibir emails de otros servidores, donde no sabemos su formato, hemos decidido aceptar cualquier archivo de texto y llenar los campos que encontremos sin que salte ningún error.

Por último bailemos con la más fea. Database tiene que tener un control muy férreo de todos los elementos, puesto que es la clase principal y llama a la mayoria de las estructuras de datos. Así pues, database se tiene que asegurar que todos los ítems que mandamos a las clases son correctos. Add email, para empezar, controla si la carpeta existe o no (aunque ya lo sepamos desde el mail) y luego si el mail ya está dentro de la carpeta (o de la linked list general) para no añadirlo otra vez. Remove email, get_email_ids y remove_folder comprueban de la misma forma que exista la carpeta que le proporciona el main. Create folder comprueba que el nombre de la carpeta que estamos creando no exista ya.

Creemos que con este sistema de control exhaustivo hemos cubierto todos los posibles errores que se pueden dar en un normal funcionamiento del programa. Es posible que hayan escenarios que no hayamos comprobado con el control de errores, y es por eso que decidimos hacer un proceso de testing una vez finalizamos el programa. En ese apartado, cada uno dedicó un tiempo para interactuar con el programa para forzarlo a situaciones críticas y ver si funcionaba correctamente. A raiz de esto, fuimos corrigiendo paulatinamente diversas imperfecciones hasta acabar con el resultado final

State of the delivery: Indicate if the program does everything requested in the assignment and if your code passes all tests provided. If the submission is not finished, indicate what is completed in terms of functionalities requested in the assignment, and also indicate what

parts of the design diagram is done and which parts are not completed or not working. Provide a brief description of the non-working part indicating as much as possible the detail you know at this point of why it is not working and what you have done to try to get it work. All of this has to give a clear view of the current state of the submission and the work done to do it. (You can indicate the state of the delivery together with the code verification if you prefer to describe it together)

El programa funciona correctamente en todas sus funcionalidades.

3.3. Any additional remarks you feel useful to provide. Be concise.

Encontramos en el handout una contradicción sobre los nombres de los ficheros. En un sitio ponía que los nombres de los ficheros debían ser message + seed y en el otro que fuera seed + EDA1. Elegimos la primera opción por consistencia con los mensajes ya creados. Por otro lado encontramos un error en las funciones proporcionadas en el main. Las funciones choose email y choose folder no controloaban correctamente el input, y las modificamos un poco. Por último, queremos dejar constancia que no entendemos porqué las funciones write_email y delete_email de útils reciben como parámetro db_config=None. Db_config está como atributo dentro de db, y por lo tanto entendemos que no es necesario que reciban ese parámetro, con lo que no lo utilizamos ni cuando las llamamos ni cuando las implementamos.

Part 4: Learning process

4.1. Learning process: Think about what you have learned while doing this practice and explain your lessons learned. Remember there is also a good learning from unsuccessful decisions of any kind (technical, logistics, team work…). The evaluation of the submission will also depend on the subjective evaluation of your experience.

Creemos que durante la práctica hemos aprendido mucho. Por un lado, sobre la implementación de la linked list. Durante las primeras instancias de su implementación implementamos todas sus funciones. También atributos como el back, y el prev en los nodos. Teníamos la convicción que implementar todas las funciones y todos los atributos haría la linked list más fácil de usar. A partir de ahí, conforme fuimos implementando la database y el resto del programa nos dimos cuenta que muchas funcionalidades pueden ser implementadas sin necesidad de utilizar todos los atributos, y de forma natural los atributos menos usados fueron desapareciendo. De igual forma con las funciones, al ir terminando el proyecto nos dimos cuenta que había funciones que no se utilizaban o que casi no se utilizaban, y también fueron desapareciendo. Respecto a útils, también tuvimos que aprender mucho sobre la manipulación de archivos físicos. Como se ha detallado arriba, aprendimos a usar el with y a manejar la biblioteca os para conseguir pathes de archivo relativo. Sobre la datababse y el main, nuestro aprendizaje se ha basado más en el control de errores que en la implementación del código, como hemos explicado arriba.

De manera transversal, hemos aprendido a trabajar con control de versión y con branches dentro de un mismo proyecto. Esto ha permitido una comunicación efectiva entre los miembros del grupo. Además, lo que valoramos más es nuestro aprendizaje sobre el control de errores. También lo detallamos arriba, pero como nota general podemos decir que ahora nos vemos preparados para encarar un proyecto desde todas sus facetas y no solo desde la implementación.

4.2. Final Evaluation Experience: Provide a constructive evaluation of the quality of this assignment with respect to meet the learning objectives of the related theory concepts. Comment from all perspectives you feel relevant (motivation, interest, usefulness, difficulty, efficiency, detail, duration, what is missing or too much? Or any other...) to improve it.

Lo primero de todo, nos gustaria agradecer la libertad dentro de la práctica para tomar nuestras propias decisiones. Somos conscientes del aumento de la carga de trabajo de cara a las correciones individuales, y damos las gracias por la posibilidad de tener margen para poder razonar como implementar cada una de las clases. Quizás al principio no nos sentimos atraídos en un nivel motivacional, pero conforme han ido pasado las semanas y nos hemos involucrado más hemos ido cogiendo interés, hasta acabar disfrutando mientras pensabamos en todos los casos posibles que debíamos controlar. La dificultad de la práctica es un poco elevada con respecto a nuestros conocimientos al principio del curso, y eso es perfecto para que no resulte aburrida ni imposible, y que favorezca buscar por stackOverflow y similares las respuestas a las preguntas que no sabemos, lo que al final repercute en nuestro beneficio. Como puntilla final, nos gustaría decir que para ser la última de las prácticas, nos hubiera gustado tener libertad total en cuanto a cantidad de clases y como organizar toda la estructura de datos, para tomar las decisiones racionales desde la concepción hasta la ejecución. Aún así, entendemos porqué esto no se ha echo. Estamos satisfechos con como nos ha quedado esta práctica y no cambiariamos nada en ella.

Submission Instructions

1. Indicate group identifier, the nia of each group member participating in the submission and name of the professor practice class in all code files and text files submitted.
2. Name all your document files (but not the code files) with a prefix: EDA1-P2-G<code_group>.
3. Create a **directory** called EDA1-P2-<code_group> to pack inside your submission.
4. Include in this directory:
    - The report in pdf format named as follows: EDA1-P2-<code_group>-report.pdf.
    - The design diagram files if in a separate document
    - Config or email files for verification, jupyter notebooks with tests or any other files you have created to your submission.
5. Create a subdirectory **code** to put a copy of your code (structured in sub-directories). This directory should contain where needed the additional files you have created for code verification (Config, email, jupyter notebooks or any other file) if any.
6. Compact the whole directory in a zip (or .rar) file named as the main submission directory (EDA1-P2-<code_group>). Submit this file in the hanoi assignment in aula global before the deadline.

Evaluation

We will be using private tests to evaluate the correctness of this submission. However, note that this submission has a less specific skeleton and hence there is bigger weight on design decisions than in the previous submissions.

The criteria to evaluate these submissions are detailed below. The evaluation of each criterion includes the design, the implementation and the code verification designed by you.

	Criteria group	**Criteria**	**Weight**
1	File management (disk) 20%	Reading config file	5%
2		Writing config file	5%
3		Reading emails	5%
4		Writing & deleting emails	5%
5	Data Structure management (in RAM) 35%	Database Configuration management	5%
6		Folder management	10%
7		Email management	10%
8		Linked List Management	10%
9	User interaction 10%	Interactive Game	5%
10		Input/output & Visualization	5%
11	Code Verification 15%		15%
12	Report 20%		20%

Before Submission

Make sure your submission fulfills the eligibility criteria.