Pontificia Universidad Católica de Chile
Escuela de Ingeniería
Departamento de Ciencia de la Computación
Primer Semestre de 2015

Examen de Introducción a la Programación - Primer Semestre 2015

Tiempo: 180 minutos – Con apuntes (1 Hoja)

Pregunta 1 (25 %)

Una palabra se dice *abecedaria* si sus letras, leídas de izquierda a derecha, están ordenadas alfabéticamente. Por ejemplo, las siguientes palabras cumplen con ser *abecedarias*:

```
amor, ello, celo, filo, chintz, luz, dinos
```

... mientras que las siguientes palabras **no lo son**:

```
gato, agua, arena, pasto, roedor
```

Se te pide lo siguiente:

a) (3 puntos) Escribir una función que permita determinar si una palabra (string) es *abecedaria*. La función debe recibir un string como parámetro (conformado exclusivamente por letras en minúscula, sin eñes ni tildes) y retornar verdadero (`True`) en caso que la palabra sea *abecedaria*, y falso (`False`) en caso contrario.

b) (3 puntos) Pedir un string al usuario e imprimir la cantidad de palabras que son *abecedarias* en él. Por ejemplo: Si el string es `"hola abel, dinos si hay luz en casa. saludos, nos vemos en intro"`, tu programa debería imprimir el valor `6`, porque `abel`, `dinos`, `luz`, `en`, `nos` y `en` son palabras *abecedarias*, pues debes ignorar los signos de puntuación (`","`, `";"`, `"."`, `"¡"`, `"!"`, `"¿"`, `"?"`) para decidir si una palabra es abecedaria o no. También puedes considerar que:

- El usuario siempre ingresará un texto que no contiene mayúsculas, eñes, ni tildes.
- Cada palabra del texto estará separada exactamente por un espacio de la siguiente.

Pregunta 2 (20 %)

Varios amigos coleccionan el álbum de la Copa América (que tiene 348 láminas o *stickers*, enumeradas del 1 al 348). Cada uno guarda en una lista ordenada (según número) las láminas que tiene pegadas en su álbum, y en una lista desordenada las láminas que tiene repetidas (es decir, láminas adicionales a las que ya tiene pegadas en el álbum).
Por ejemplo, Daniel tiene las siguientes listas:

```
album_daniel = [3, 4, 7, 14, 28, 54, 122, 123, 124, 324] #en el album
reps_daniel = [14, 3, 3, 3, 324, 3, 14] #repetidas
```

Se te pide lo siguiente:

a) (2 puntos) Escribe la función `faltan(album)` que recibe la lista de láminas que tiene una persona en su álbum `album` y entrega una lista con las láminas que le faltan.
Ejemplo: Si `album=[1,2]`, `faltan(album)` debe entregar una lista con todos los números entre 3 y 348.

b) (4 puntos) Dos personas quieren hacer un intercambio de láminas que sea equitativo, es decir, que cada uno entregue y reciba el mismo número de láminas. Para eso, escribe una función `numero(album1,album2,reps1,reps2)` que reciba las láminas que tienen ambos (`album1` y `album2`), y las láminas repetidas de ambos (`reps1` y `reps2`), y retorne cuántas láminas podrán efectivamente intercambiar.
Ejemplo: Si `album1=[1, 2, 33]`, `album2 = [1, 15, 110, 200]`, `reps1=[1, 33, 1, 33]` y `reps2=[15, 110, 200, 200]`, la función debe retornar 1, pues la persona 1 solamente le puede entregar una lámina (33) a la persona 2, aunque la persona 2 le podría entregar 3 (15, 110 y 200).

Pregunta 3 (30 %)

Para hacer más eficiente la administración de las redes de comunicación, los mensajes enviados por Internet de un computador a otro se envían separados en partes pequeñas llamadas `paquetes`. Cada paquete es enviado con su debida *identificación* (mensaje al que pertenece) y con un número que indica su posición en la secuencia de paquetes del mensaje. Por demoras en la red, los paquetes de un mensaje pueden llegar al receptor desordenados, por lo que el computador receptor debe reconstruir el mensaje original usando el número que indica la posición del paquete en el mensaje. Por ejemplo, si el mensaje X se divide en 4 partes [X1,X2,X3,X4] y el mensaje Y en 2 [Y1,Y2], el receptor los podría recibir como [X2,Y2,X4,X1,Y1,X3] y luego los tiene que re-armar como dos mensajes distintos: [X1,X2,X3,X4] y [Y1,Y2] para poder procesarlos.
Implementa las clases y métodos que se indican a continuación. Para hacerlo, puedes asumir que el identificador del mensaje y que la posición del paquete son enteros y que el contenido del paquete es de tipo string:

Clase	Atributos	Métodos

`Paquete`	`identificador_mensaje`
	`posicion_en_secuencia`
	`contenido_paquete`
		`__init__(self,id,posicion,contenido)`(1)

`Mensaje`	`identificador_mensaje` `lista_de_paquetes`	`__init__(self,id)`(2) `agregar_paquete(self,p)`(3) `imprime_mensaje_ordenado(self)`(4)

(1) Recibe como parámetros el identificador del mensaje (`id`), la posición del paquete en la secuencia del mensaje (`posicion`) y el contenido del paquete (`contenido`).
(2) Recibe como parámetro el identificador del mensaje (`id`).
(3) Recibe como parámetro el valor `p` que es un objeto de tipo Paquete. El paquete `p` se puede agregar al mensaje solamente si su identificador es igual al identificador del mensaje. Si el identificador del paquete no es igual al del mensaje, el paquete se debe ignorar. Puedes considerar que no te llegarán paquetes repetidos (con el mismo identificador y número de secuencia que otro paquete que ya fue agregado al mensaje).
(4) Imprime un string con el mensaje ordenado. Por ejemplo, si el mensaje tiene los paquetes Paquete(1,2,"mundo!") y Paquete(1,1,"hola "), debes imprimir "hola mundo!".

a) (1 punto) Implementa la clase `Paquete`, con el constructor indicado en la tabla anterior.

b) (2.5 puntos) Implementa la clase `Mensaje`, con el constructor indicado y sus dos métodos.

c) (2.5 puntos) Implementa la función `recibir(paquetes)`, que recibe una lista de paquetes desordenados (referentes a uno o varios mensajes), e imprime cada mensaje recibido en forma ordenada. Por ejemplo, si recibe la lista `[Paquete(1,2,"mundo!"), Paquete(2,4,"progra"), Paquete(2,3,"de "), Paquete(2,1,"Exam"), Paquete(2,2,"en "), Paquete(1,1,"hola ")]`, se deberían imprimir dos mensajes:

```
Mensaje 1: hola mundo!
Mensaje 2: Examen de progra
```

No importa si se imprime primero el mensaje 1 o el mensaje 2, lo importante es que cada mensaje se imprima ordenado.

Pregunta 4 (25 %)

Programa las siguientes funciones utilizando recursión. Una solución iterativa será evaluada con **0 puntos**. En ambos casos, si lo necesitas, puedes agregar parámetros extra a la definición de tu función.

a) (3 puntos) Implementa la función **recursiva** `mezclar(s1,s2,...)`, que recibe 2 palabras *abecedarias* (`s1` y `s2`) y retorna una nueva palabra *abecedaria* formada por la mezcla de `s1` y `s2` (i.e. que incluye todos los caracteres de `s1` y `s2`). Recordar que una palabra se dice *abecedaria* si sus caracteres están ordenados alfabéticamente de izquierda a derecha. Puedes considerar que `s1` y `s2` solo tendrán caracteres alfabéticos en minúsculas (sin eñes ni tildes).

Por ejemplo, `mezclar("amor","luz",...)` debería retornar `"almoruz"`.

b) (3 puntos) Implementa la función **recursiva** `calcular_strings(n,...)`, que retorna una lista con todos los strings formados por ceros y unos de largo `n`. Por ejemplo:

Llamada a la función	Retorno esperado
`calcular_strings(1,...)`	`["0","1"]`
`calcular_strings(2,...)`	`["00","01","10","11"]`
`calcular_strings(3,...)`	`["000","001","010","011","100","101","110","111"]`

Observación: No importa el orden de los elementos en la lista.