Examen final 2021-03-15

95.14/75.40 - Algoritmos y Programación I - Curso Essaya

Objetivo

Se dispone de los archivos `ej1.py`, `ej2.py`, `ej3.py`, `ej4.py` y `ej5.c` correspondientes a los 5 ejercicios del examen.

Cada uno tiene un lugar para escribir la implementación del ejercicio, y una función de pruebas para verificar que la solución es correcta.

El examen se aprueba con al menos 3 ejercicios correctamente resueltos. Un ejercicio se considera correctamente resuelto si:

- El programa `ej<n>` no tiene errores de sintaxis y puede ser ejecutado
- La implementación cumple con lo pedido en el enunciado

En algunos ejercicios se incluye un ejemplo de uno o dos casos de prueba y queda a cargo del alumno agregar más casos de prueba, para los que se provee sugerencias. En otros ejercicios se provee únicamente sugerencias. La implementación de las pruebas adicionales es **opcional**, pero se recomienda hacerlo ya que permite asegurar que la resolución del ejercicio es correcta.

Ejercicios en lenguaje Python

Al ejecutar cada uno de los ejercicios (`python3 ej<n>.py`), se ejecutan todas las pruebas presentes en la función `pruebas`.

Si alguna de las verificaciones falla, se imprime un mensaje de error y el programa termina su ejecución. Por ejemplo:

```
$ python3 ej1.py
Traceback (most recent call last):
File "ej1.py", line 148, in pruebas
    assert p != None
AssertionError
```

Cuando todas las pruebas pasan correctamente, se imprime `OK`:

```
$ python3 ej1.py
ej1.py: OK
```

Pruebas

Se recomienda usar la instrucción `assert` de la biblioteca estándar para verificar condiciones en las pruebas. Ejemplo de uso:

```python
# función a probar
def sumar(a, b):
    return a + b

# pruebas
```

```python
def pruebas():
    assert sumar(0, 0) == 0
    assert sumar(2, 3) == 5
    assert sumar(2, -2) == 0

    from os import path
    print(f"{path.basename(__file__)}: OK")

pruebas()
```

Nota: A veces para depurar un error en las pruebas es útil imprimir valores; se permite el uso de `print()` para ello.

Nota: A veces para implementar las pruebas es útil utilizar números aleatorios. Se permite el uso de la biblioteca `random` para ello. En ese caso, se recomienda ejecutar `random.seed(0)` al inicio del programa para asegurar que la secuencia de números aleatorios sea siempre la misma, y así facilitar la depuración.

Ejercicios en lenguaje C

Para compilar y ejecutar el ejercicio `ej5.c`:

```
$ gcc -Wall -pedantic -std=c99 ej5.c -o ej5
$ ./ej5
ej5.c: OK
```

Pruebas

Se recomienda usar la función `assert` de la biblioteca estándar para verificar condiciones en las pruebas. Ejemplo de uso:

```c
#include <stdio.h>
#include <assert.h>

// funcion a probar
int sumar(int a, int b) {
    return a + b;
}

// pruebas
int main(void) {
    assert(sumar(0, 0) == 0);
    assert(sumar(2, 3) == 5);
    assert(sumar(2, -2) == 0);

    printf("%s: OK\n", __FILE__);
    return 0;
}
```

Ejercicios

Ejercicio 1: Escribir el método de `ListaEnlazada` `esta_ordenada` que determine **en forma recursiva** si la lista está ordenada o no en forma ascendente.

Ejercicio 2: Escribir una función que recibe la ruta a un archivo `nacimientos.csv` con la forma `fecha;apellido;nombre`, donde la `fecha` tiene la forma `aaaa-mm-dd`, y devuelve un diccionario con el nombre más popular de cada año.

Es posible que alguna fila del archivo no cumpla con el formato indicado (por ejemplo que falte algún campo); en ese caso se debe ignorar esa fila.

También es posible que el campo `nombre` contenga dos o más nombres separados por espacios; en ese caso considerar cada uno de los nombres por separado para el año correspondiente.

También es posible que los nombres no sean consistentes con el uso de mayúsculas / minúsculas; se debe considerar `"nombre"`, `"NOMBRE"` o `"Nombre"` como el mismo nombre.

Ejercicio 3: Implementar la función `multi_merge` que recibe una lista de k listas ordenadas, y devuelve una lista ordenada con los elementos de todas las listas recibidas, **en tiempo lineal**.

Ayuda: Para el caso de $k = 2$ debería comportarse como la función `merge` de MergeSort.

Ejercicio 4: Sea la clase `Cola` implementada como un arreglo circular, con los siguientes atributos:

- Una lista de Python de tamaño fijo `K` (inicialmente todos sus elementos son `None`)
- La posición de inicio de la cola (`0 ≤ inicio ≤ K - 1`)
- La posición de fin de la cola (`0 ≤ fin ≤ K - 1`)

La cola está vacía si `inicio` es igual a `fin`. En caso contrario, la cola contiene todos los elementos que están en la lista desde la posición `inicio` hasta la posición `fin` (no inclusive). La posición `fin` puede ser menor a `inicio`, en cuyo caso los elementos que están en la cola son los que se encuentran hacia la derecha de `inicio` y hacia la izquierda de `fin`. La cola puede contener como máximo `K - 1` elementos.

Ejemplo, 5 elementos encolados, con `K = 10`, `inicio = 7`, `fin = 2`; el caracter `-` representa `None`:

```
[ 4 5 - - - - - 1 2 3 ]
      ▲         ▲
      fin       inicio
```

Implementar los métodos `__init__`, `encolar`, `desencolar`.

Ejercicio 5: Escribir en C la función `void intercalar(const char *s1, const char *s2, char *dest)` que guarda en `dest` el resultado de intercalar caracter a caracter las cadenas `s1` y `s2`. Asumir que `dest` tiene espacio suficiente para guardar el resultado.

Ejemplo: para las cadenas `"Hola"` y `"mundo!"` el resultado en `dest` sería `"Hmoulnado!"`