Examen final 2021-03-22

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 una función que recibe una matriz de $N \times M$ (representada como una lista de listas), y devuelve una nueva matriz que resulta de transponer la matriz original.

$$\begin{bmatrix} a & b \\ c & d \\ e & f \end{bmatrix}^t = \begin{bmatrix} a & c & e \\ b & d & f \end{bmatrix}$$

Ejercicio 2: Un supermercado necesita llevar el control de stock. Cada uno de sus productos tiene un número identificatorio único, un nombre y una cantidad en stock.

Se pide implementar la clase `Producto`, con los atributos `id`, `nombre` y `cantidad`, y las funciones:

- `guardar(productos, ruta)`: recibe una lista de instancias de `Producto` y guarda la información en un archivo en la ruta indicada.
- `cargar(ruta)` carga la información del archivo y devuelve una lista de instancias de `Producto`.
- `verificar_ids(productos)` devuelve True si todos los productos en la lista tienen un número identificatorio único; es decir si no hay repeticiones.

Queda a libre criterio la elección de tipos de dato y el formato del archivo.

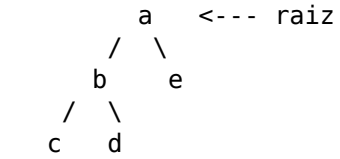
Ejercicio 3: Un árbol binario es una estructura enlazada en la que el primer nodo se llama raíz, y cada nodo contiene referencias a otros dos nodos, llamados *hijo izquierdo* y *derecho*. Si ambas referencias son nulas se dice que el nodo corresponde a una *hoja* del árbol.

Sea la clase `Nodo` que representa un nodo del árbol. Se pide implementar la función `crear_arbol(s)` que devuelve un árbol binario creado a partir de su descripción `s`.

La descripción consiste en una cadena de caracteres con las siguientes propiedades:

- el árbol vacío se representa con un punto (`.`)
- un árbol no vacío se representa con un caracter distinto de `.` (que representa el dato contenido en el nodo raíz), seguido primero de la descripción del hijo izquierdo y después de la descripción del hijo derecho.

Ejemplo: la descripción `"abc..d..e.."` corresponde al siguiente árbol:



Recomendación: pensar la función en forma recursiva.

Ejercicio 4: Sea una función matemática $f(n)$ que recibe un número entero y devuelve un número real. No sabemos exactamente qué operación matemática hace f, pero sabemos que:

- f es continua
- f es monótona creciente (si $n_1 > n_2$, entonces $f(n_1) > f(n_2)$)

Escribir la función `buscar_cero(f, n_min, n_max)`, que devuelve el valor de `n` para el cual `f(n) == 0.0` (suponiendo que `n_min <= n <= n_max`), consumiendo la menor cantidad de tiempo posible.

Ejercicio 5: Escribir en lenguaje C la función `void borrar_espacios_consecutivos(char s[])` que elimina *in-place* los espacios consecutivos en la cadena `s`. Ejemplo:

```
char s[] = "A otro    perro con   ese  hueso";
borrar_espacios_consecutivos(s);
// s contiene "A otro perro con ese hueso"
```