Examen final 2021-09-03

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

a. Implementar una clase `Persona` que representa a una persona en el padrón electoral nacional. Una persona tiene un DNI (`int`), un nombre y una dirección (cadenas de texto).
b. Escribir la función `buscar_en_padron(dni, personas)` que recibe un arreglo de instancias de `Persona` ordenado por DNI, y un DNI a buscar, devuelve la persona correspondiente (o `None` en caso de no encontrarla) en tiempo **mejor que lineal**.

Ejercicio 2 Una lista enlazada contiene un ciclo si algún nodo referencia a algún otro nodo que se encuentra ”atrás” en la secuencia.

El algoritmo de Floyd permite detectar ciclos en una lista enlazada, y funciona mediante una *tortuga* y una *liebre*. La tortuga es una referencia a un nodo que avanza ”despacio”: en cada iteración avanza un nodo hacia adelante. La liebre es una referencia a un nodo que avanza ”rápido”: en cada iteración avanza dos nodos hacia adelante.

La tortuga y la liebre arrancan en el primer nodo. Si en algún momento se vuelven a encontrar en algún nodo significa que hay un ciclo. Si llegan al final de la lista sin volver a encontrarse es que no hay ciclos.

Implementar el método de `ListaEnlazada` `hay_ciclo` que determina si hay o no un ciclo en la lista, mediante el algoritmo de Floyd.

Ejercicio 3 Escribir en forma **recursiva** la función `intercalar(a, b)` que recibe dos cadenas `a` y `b` y devuelve el resultado de intercalarlas caracter por caracter.

Ejemplo: `intercalar("hola", "mundo!") -> "hmoulnado!"`.

Ejercicio 4 Se tiene un archivo CSV con datos de mediciones de temperatura, de la forma: `aaaa-mm-dd,lugar,temp`, no estando ordenado por ningún criterio particular. Por ejemplo:

```
2021-08-30,Buenos Aires,22
2021-08-31,Buenos Aires,23
2021-08-30,La Plata,20
2021-08-31,La Plata,19
```

Escribir una función que reciba la ruta al archivo CSV y devuelva un diccionario con el día más caluroso de cada lugar y la temperatura de ese día. Por ejemplo:

```
{
    "Buenos Aires": ("2021-08-31", 23),
    "La Plata": ("2021-08-30", 20),
}
```

Atención: La función debe ignorar cualquier línea del archivo que no cumpla con el formato indicado.

Ejercicio 5 Escribir en C la función `void ordenar_seleccion(int numeros[], int n)` que ordena el arreglo de números *in place* mediante el algoritmo de ordenamiento por selección. El pseudocódigo del algoritmo es:

```
algoritmo ordenar_seleccion(arreglo A de tamaño N):
    para i desde 0 hasta N - 2 inclusive:
        sea p := la posición del elemento mínimo entre i y N - 1 inclusive
        intercambiar A[p] con A[i]
```