UNIVERSIDAD NACIONAL DE HURLINGHAM

Introducción a la Programación - Práctica 7

Funciones Simples

CONSEJOS:

- Leer el enunciado en su totalidad y pensar en la forma de resolverlo **ANTES** de empezar a escribir código
- Si un ejercicio no sale, se puede dejar para después y continuar con los ejercicios que siguen
- Los ejercicios están pensados para ser hechos después de haber mirado la teórica correspondiente
- Algunos de los ejercicios están tomados de las guías prácticas utilizadas en la materia de Introducción a la Programación de la Universidad Nacional de Quilmes por Pablo Ernesto “Fidel” Martínez López y su equipo. También Federico Aloi y Miguel Miloro, a su vez basada en las guías Ejercicios de Introducción a la Programación del CIU General Belgrano, elaboradas por Carlos Lombardi y Alfredo Sanzo, y Fundamentos de la Programación del Proyecto Mumuki. Agradecemos a todos los que nos ayudaron con su inspiración.
- **Realizar en papel los ejercicios que así lo indiquen.**
- **Sí un ejercicio indica BIBLIOTECA significa que podrá ser utilizado en el parcial sin definirlo. Es útil mantener registro de dichos procedimientos en su carpeta.**

FUNCIONES SIMPLES:

1. Definiendo mis primeras funciones

Definir funciones totales que describen cada uno de las siguientes casos:
Atención: Es conveniente utilizar funciones para expresar subtareas, de forma que las expresiones utilizadas resulten fáciles de entender. Recordar además escribir los contratos.

a. La cantidad total de bolitas de la celda actual.
b. Sí hay más de 5 bolitas en total en la celda actual.
c. Sí hay exactamente 5 bolitas en la celda actual.
d. Sí hay al menos 5 bolitas en la celda actual.
e. Sí hay bolitas de todos los colores en la celda actual.
f. Si la celda actual está vacía.
g. Sí a la celda actual le faltan bolitas de alguno de los colores y no está vacía.

2. Algunas funciones útiles

BIBLIOTECA Escribir las siguientes funciones, para agregarlas a la biblioteca.

a. **`esCeldaVacía()`**, que indica si la celda actual se encuentra vacía.
b. **`tieneUnaDeCada()`**, que indica si la celda actual tiene al menos una bolita de cada color.
c. **`esCeldaConBolitas()`**, que indica si la celda actual tiene al menos una bolita, de cualquier color.

3. ¡A la batalla!, parte 2

Escribir las siguientes funciones para el juego ¡A la batalla! de la práctica anterior, donde en las celdas del tablero se representan soldados (los aliados con una bolita de color Negro y los enemigos con una bolita de color Rojo por cada soldado).

a. **`cantidadDeSoldadosDe_(colorDelEjército)`**, que describe la cantidad de soldados de la celda actual del ejército dado.
b. Vuelva a escribir **`EnviarAliadosParaDuplicarEnemigos()`**, y **`PelearLaBatalla()`**, que realizó en la práctica anterior, ahora haciendo uso de la función hecha en el punto a, y luego conteste: ¿Realizó procedimientos auxiliares para resolver el problema? ¿Cuál de los códigos (entre este y el de la práctica anterior) comunica mejor la solución al problema? ¿Por qué?
c. **`esCeldaIndefensa()`** que indica si no hay soldados en la celda actual.
d. **`estadoDeEmergencia()`** que indica si existen más de 100 soldados enemigos, y además la celda está indefensa.
e. **`hayAlMenos_AliadosPorCada_Atacantes(cantidadDefensa, cantidadAtaque)`** que indica si hay por lo menos ***`cantidadDefensa`*** soldados aliados por cada **`cantidadAtaque`** soldados enemigos en la celda actual. Por ejemplos si en la celda actual hubiera 10 soldados aliados y 5 enemigos, la función invocada como **`hayAlMenos_AliadosPorCada_Atacantes(2, 1)`**, describiría Verdadero, pues hay al menos dos aliados por cada atacante. Si se invocara con esos mismos argumentos en una celda donde hay 7 aliados y 5 enemigos, describiría Falso.
f. **`aliadosNecesariosParaDefensaEficazContra_(cantidadDeSoldadosEnemigosAdicionales)`** que describe el número de soldados aliados que faltan para defender la celda actual si a ella se suman la cantidad de soldados enemigos dada. Tener en cuenta que en la celda actual puede ser que haya soldados, pero que es precondición de esta función que no hay suficientes aliados. Recordemos que 2 soldados enemigos pelean contra 3 soldados aliados y todos mueren.

4. ¡Mira mami! ¡sin bolitas!

EN PAPEL A continuación se dan una serie de funciones que se consideran primitivas, es decir, que puede asumir realizadas y no debe implementarlas de ninguna forma.

```
hayUnPlanetaA_Hacia_(distancia, dirección)
/*
    PROPÓSITO: Indica si hay un planeta a **distancia** celdas hacia
**dirección**.
    PARÁMETROS:
        * distancia: Número - La cantidad de celdas a la cual se
                     desea buscar un planeta.
        * dirección: Dirección - La dirección hacia la cual mirar el planeta.
    PRECONDICIONES:
        * Hay al menos **distancia** celdas en dirección **dirección**.
        * El cabezal está sobre la nave.
    TIPO: Booleano
*/
```

```
combustibleRestante
/*
    PROPÓSITO: Indica la cantidad de combustible que le queda a la nave.
    PRECONDICIONES:
        * El cabezal está sobre la nave.
    TIPO: Número
*/
```

Utilizando dichas funciones, se pide que se definan las siguientes, sin hacer suposiciones sobre la representación.

a. **`sePuedeAterrizarA_Hacia_(distanciaAPlaneta, direcciónAPlaneta)`**, que asumiendo que el cabezal se encuentra sobre la nave y hay al menos **distanciaAPlanera** celdas en dirección ***direcciónAPlaneta***, indica si hay un planeta a ***distanciaAPlaneta*** en la dirección ***direcciónAPlaneta*** y sí el combustible es suficiente para llegar al mismo.
   La nave consume una única unidad de combustibe por cada celda que deba moverse.
b. Sabiendo que el cabezal se encuentra sobre la nave y a exactamente 3 celdas de distancia de todos los bordes, se pide que escriba la función **`hayUnPlanetaRecto()`**, que indica que existe un planeta en cualquiera de las direcciones, a cualquier distancia desde la nave.

5. El bosque, parte 4

Continuaremos utilizando el mismo dominio del bosque que venimos utilizando en las prácticas anteriores. Esta vez se pide escribir los siguientes procedimientos que modelan el bosque. Considerar la reutilización de los procedimientos hechos en las partes anteriores y la definición de nuevas funciones necesarias para no tener que depender de la representación dada.

a. **`GerminarSemilla()`**, que transforma una semilla en un árbol en la celda actual. La germinación consume tres unidades de nutrientes. Si en la celda no hay semilla, o no hay suficientes nutrientes, no se hace nada.

b. **`AlimentarÁrboles()`**, que hace que los árboles de la celda actual se alimenten, consumiendo un nutriente cada uno. El único cambio que hay que hacer es la eliminación de los nutrientes. Si hay menos nutrientes de lo que se necesita, se consumen todos los que hay.

c. **`ExplotarBomba()`**, que explota una bomba en la celda actual, eliminando árboles. Al explotar, una bomba derriba 5 árboles en la celda actual y 3 en la celda lindante al Norte. Si la celda actual está en el borde Norte, entonces solo se eliminan los árboles de la celda actual. **Atención**: cuando haya menos árboles de los que la bomba puede eliminar, entonces elimina los que haya. La bomba se consume en el proceso, o sea, hay que eliminarla.

d. **`Polinizar()`**, los árboles en la celda actual polinizan la celda lindante en la dirección Este, generando tantas semillas en esa celda como árboles haya en la celda actual, menos 3. Por ejemplo, si en la celda actual hay 5 árboles, se generan 2 semillas en la celda lindante al Este. Si en la celda actual hay menos de 3 árboles, o no tiene lindante al Este, entonces no se hace nada.

6. ¿Vamos al banco? - Parte 2

Continuaremos utilizando el mismo dominio del banco de la práctica anterior. Esta vez, vamos a realizar funciones que nos permitan abstraernos de la representación subyacente, así como simplificar cálculos en nuestras operaciones.

Se pide entonces que realice las siguientes funciones:

a. **`pesos()`** que describe el color con el que se representan los pesos en el tablero, Negro.

b. **`dólares()`** que describe el color con el que se representan los dólares en el tablero, Verde.

c. **`euros()`** que describe el color con el que se representan los euros en el tablero, Azul.

d. **`yuanes()`** que describe el color con el que se representan los yuanes en el tablero, Rojo.

e. **`ahorrosEn_(moneda)`** que dada una moneda, indica la cantidad de unidades de esa moneda en la cuenta actual.

f. **`cuantosDolaresSePuedeComprarCon_Pesos(cantidadDePesos)`** que indica la cantidad de dólares que se pueden comprar con una cantidad de pesos dada.

g. **`cuantosEurosSePuedeComprarCon_Pesos(cantidadDePesos)`** que indica la cantidad de euros que se pueden comprar con una cantidad de pesos dada.

h. **`cuantosYuanesSePuedeComprarCon_Pesos(cantidadDePesos)`** que indica la cantidad de yuanes que se pueden comprar con una cantidad de pesos dada.

i. **`cuantosPesosSiVendo_Dólares(cantidadDeMonedaExtranjera)`** que indica la cantidad de pesos a obtener si se venden (depositan) la cantidad de dólares dada.

j. **`cuantosPesosSiVendo_Euros(cantidadDeMonedaExtranjera)`** que indica la cantidad de pesos a obtener si se venden (depositan) la cantidad de euros dada.

k. **`cuantosPesosSiVendo_Yuanes(cantidadDeMonedaExtranjera)`** que indica la cantidad de pesos a obtener si se venden (depositan) la cantidad de yuanes dada.

l. Vuelva a realizar los procedimientos de la práctica anterior, ahora utilizando las funciones realizadas en los puntos anteriores.

Reflexionamos: ¿Cuánto esfuerzo conlleva cambiar la representación de Euros y Pesos, para que ahora los primeros sean representados con bolitas negras y las segundas con azules.? ¿Cuántos lugares hubo que tocar? Sí la respuesta es más de 2, puede que no haya resuelto bien los ejercicios.