UNIVERSIDAD NACIONAL DE HURLINGHAM

Instituto de Tecnología e Ingeniería / Programación Estructurada

REPETICIÓN CONDICIONAL

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.*

REPETICIÓN CONDICIONAL

1. Un nuevo IrAlBorde

Definir el procedimiento **IrAlBorde_(dirección)**, que lleva al cabezal al borde dado por el parámetro dirección. ¡Atención! Debe realizar el ejercicio sin utilizar el comando primitivo IrAlBorde. Dado que el único otro comando primitivo que permite mover el cabezal es Mover, debe repetirse su uso mientras nohasta que se haya cumplido el propósito.

Reflexionamos ¿Cuál es la condición que indica que el propósito se cumplió?

2. Otra forma de sacar todas

Volver a definir el procedimiento **SacarTodasLasDeColor_(color)**, que quita todas las bolitas del color dado por el parámetro color de la celda actual, pero esta vez SIN utilizar la expresión primitiva nroBolitas (directa o indirectamente).

RECORRIDOS POR CELDAS DE LA FILA O COLUMNA

3. Vaciando una fila

Considerar el procedimiento **VaciarFilaDe_(color)**, que debe quitar todas las bolitas del color dado por el pa rámetro color de cada una de las celdas de la fila actual . El cabezal puede empezar en cualquier celda de la fila, y también puede terminar en cualquier celda de la fila (ya sea celda inicial o cualquier otra).

a) Definir el procedimiento, como siempre, comenzando por establecer el contrato, y luego recién el código.
b) ¿La solución dada funciona si el cabezal se encuentra en medio de una fila? Si no es así, corregir el programa para que funcione en este caso también.
c) Al recorrer la fila, ¿en qué dirección se movió el cabezal? ¿Podría haberse movido en la dirección opuesta?
d) A partir de la respuesta anterior, ¿de cuántas formas posibles se puede realizar el recorrido de una fila?
e) Volver a definir el procedimiento con direcciones distintas.
f) ¿Y si tuviéramos que vaciar la columna en lugar de la fila? ¿Qué cambia entre las distintas formas de moverse?

4. Vaciando una fila hacia...

Defina ahora el procedimiento **VaciarFilaDe_HaciaEl_(color, dirección)**, que debe quitar todas las bolitas del color dado por el parámetro color de cada una de las celdas de la fila actual, desde la celda en donde se encuentra el cabezal (incluyendo esta) hacia el final de la fila en la dirección dada por dirección. Tras definir el código y el contrato considere.

a) ¿Qué valores puede tomar el parámetro dirección para que el propósito sea consistente con su nombre?
b) ¿Qué nombre podría recibir el procedimiento para que sea correcto utilizarlo con cualquier dirección?

RECORRIDOS POR CELDAS DEL TABLERO

5. Vaciando un tablero

En cada uno de los casos siguientes, definir de la forma indicada el procedimiento **VaciarTableroDe_(color)**, que quite todas las bolitas del color dado por el parámetro color de cada una de las celdas del tablero. El cabezal debe poder comenzar en cualquier celda del tablero, y no es relevante para el problema donde finaliza, basta con que lo declare en su propósito.

Estructurar el procedimiento como un recorrido sobre las filas . ¿Qué subtareas van a precisarse en este caso? ¿Es necesario volver a definirlas o se pueden encontrar en esta práctica?

Estructurar el procedimiento como un recorrido sobre las celdas del tablero. Las subtareas necesarias serán diferentes, y puede ser que sea necesario definir alguna que aún no está disponible en esta práctica.

En este última opción, ¿Cuántas formas distintas hay de recorrer el las celdas del tablero? ¿Se podría elegir valores distintos para los movimientos?

Implemente ahora nuevamente el procedimiento de 2 formas distintas.

Reflexionamos ¿Qué diferencias hay entre los recorridos anteriores del punto a y del b? ¿Qué subtareas son más complejas en cada caso? ¿Podrían considerarse otros recorridos que no fueran sobre celdas o filas?

6. Las subtareas más útiles de la historia

BIBLIOTECA. Escribir los procedimientos y las funciones necesarias para generalizar la noción de recorrido por celdas de un tablero, para que las direcciones de recorrido no estén fijas. En particular, definir (como siempre, comenzando por los contratos):

a) IrAPrimeraCeldaEnUnRecorridoAl_Y_(dirPrincipal, dirSecundaria)
b) haySiguienteCeldaEnUnRecorridoAl_Y_(dirPrincipal, dirSecundaria)
c) IrASiguienteCeldaEnUnRecorridoAl_Y_(dirPrincipal, dirSecundaria)

Que hacen precisamente lo que sugiere su nombre, permitiendo utilizarlas en un recorrido por celdas. Puede probarlas intentando colocar una bolita en cada celda del tablero, o volviendo a implementar el ejercicio anterior, ahora de forma parametrizada. Al escribir las precondiciones, tener en cuenta que las direcciones no pueden ser cualesquiera, sino que deben estar relacionadas. ¿Cuál es esa relación? ¿Cómo expresarla?

7. Y ahora más cosas sobre el tablero

Escribir ahora los siguientes procedimientos, teniendo en cuenta que para todos, el cabezal puede comenzar en cualquier lugar del tablero, y terminar en dónde usted crea conveniente.

a) **PintarTableroDe_(color)** que coloca exactamente una bolita del color dado en cada celda del tablero.

b) **UnaDeCadaEnTodoElTablero()** que coloca una bolita de cada color en cada celda del tablero.

c) **RellenarCon_EnAusenciaDe_EnElTablero(colorAPoner, colorAMirar)** que coloca una bolita de color colorAPoner en cada celda del tablero en la que no haya al menos una bolita de color colorAMirar.

d) **CompletarHasta_De_EnElTablero(cantidad, color)** que deja en cada celda del tablero exactamente tantas bolitas del color dado como la cantidad indicada por el parámetro cantidad. Note que puede que ya existan bolitas del color dado en algunas de las celdas, en cuyo caso. Realice el procedimiento sin hacer uso del comando Sacar ni ninguno de los procedimientos que implican Sacar.

RECORRIDOS DE BÚSQUEDA

8. Buscando la bolita roja en la fila/columna

Escribir un procedimiento **IrHastaLaBolitaRojaHacia_(direcciónABuscar)** que deja el cabezal posicionado en la celda más próxima a la actual en la dirección dada que posea una bolita de color Rojo Cuidado, sí hay una bolita de color Rojo en la celda actual, el cabezal debe moverse a la más cercana, no permanecer en la actual. ¿Cuál es la precondición de este procedimiento?

9. Buscando la celda vacía

Escribir un procedimiento **IrALaSiguienteVacíaHacia_(dirección)** que posiciona el cabezal en la próxima celda vacía en la fila o columna, desde la celda en donde se encuentra el cabezal (sin incluirla) hacia el borde en la dirección dada, dejando el cabezal en el borde en caso de no haber ninguna celda vacía en dicha dirección.

10. Buscando en todo el tablero

Definir un procedimiento **IrHastaLaQueTengaUnaDeCada()** que posiciona el cabezal en cualquier celda que contenga una bolita de cada color, y que hace BOOM si no hubiera en el tablero alguna celda que cumpla con dicha característica.

11. ¿Y esto qué hace?

EN PAPEL. Nova escribió un procedimiento cuyos contratos son inexistentes, y donde los nombres de los procedimientos y de las funciones son muy poco descriptivos.

```
procedure Pos() {
    while (not a() && b() && not estoyElFinalDeUnRecorrido__(Norte, Este)) {
        PasarASiguienteCelda__(Norte, Este)
    }
}

function a() {
    return (hayBolitas(Azul))
}

function b() {
    return (nroBolitas(Negro) < 42)
}
```

INTEGRANDO

12. Comiendo la pieza

EN PAPEL. Dadas las siguientes primitivas que modelan partes de un juego de ajedrez:

```
function hayUnaPiezaNegra()
  /*
    PROPÓSITO: Indica si hay una pieza negra en la
        celda actual.
    PRECONDICIONES:
      * Ninguna
  */
```

```
procedure ComerPiezaNegra()
  /*
    PROPÓSITO: Come la pieza negra en la celda actual.
    PRECONDICIONES:
      * Hay una pieza negra en la celda actual
  */
```

```
procedure MoverTorreBlancaHacia_(direcciónAMover)
  /*
    PROPÓSITO: Mueve la torre blanca una celda en la
        dirección dada.
    PARÁMETROS:
      * direcciónAMover: Dirección - La dirección
        hacia la cual mover la pieza.
    PRECONDICIONES:
      * Hay una celda en la dirección dada.
  */
```

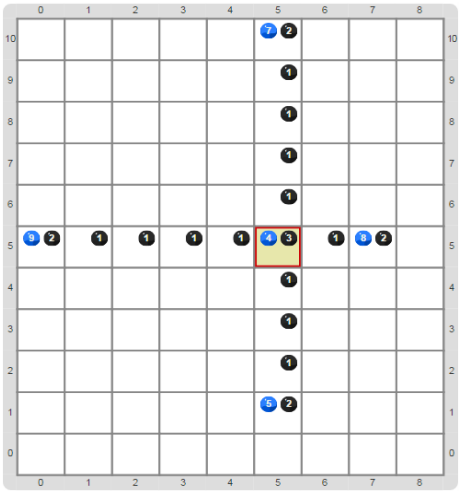
Se pide que escriba el procedimiento **ComerPiezaNegraConTorreHacia_(direcciónAComer)** que asumiendo que se está sobre una torre blanca, come la pieza negra más próxima a la celda actual hacia la dirección dada, dejando la torre en dicha celda.

13. Distribución de mercadería / ☠

Se desea modelar el movimiento de mercadería en una sencilla red de depósitos, que tiene un depósito central, más un depósito local para cada punto cardinal. Para esto, se va a representar en el tablero un mapa muy simplificado.

- Tres bolitas negras marcan el depósito central,
- Dos bolitas negras marcan un depósito local,
- Una bolita negra marca el camino de central a local,
- Cada bolita azul marca una unidad de mercadería.

Los depósitos locales forman una cruz, donde el centro es el depósito central. No se sabe a qué distancia están los depósitos locales del depósito central. Este es un ejemplo de modelo:



Definir:

a) **esDepósitoCentral()** y **esDepósitoLocal()** que indican si el cabezal está, respectivamente, en el depósito central o en un depósito local.

b) **IrDeCentralAlLocal_(dirección),** que mueve el cabezal del depósito central al depósito local que está en la dirección dada, suponiendo que el cabezal comience en el depósito central.

c) **IrDelLocal_ACentral(dirección),** que mueve el cabezal al depósito central, suponiendo que el cabezal está en el depósito local que está en la dirección dada.

Aclaración: si se pide **IrDelLocal_ACentral(Sur),** quiere decir que el cabezal está en el depósito Sur, por lo tanto, debe moverse hacia el Norte.

d) **Llevar_MercaderíasAlLocal_(cantidad, dirección)**, que lleva la cantidad de mercadería indicada del depósito central al depósito local que está en la dirección indicada. Si en el depósito central no hay suficiente cantidad de mercadería, no se hace nada. Se puede suponer que el cabezal está en el depósito central, y debe dejarse en el mismo lugar.

e) **Traer_MercaderíasDelLocal_(cantidad, dirección**), que lleva la cantidad de mercadería indicada del depósito local en la dirección indicada, al depósito central. Si en el depósito local indicado no hay suficiente cantidad de mercadería, no se hace nada. Se puede suponer que el cabezal está en el depósito central, y debe dejarse en el mismo lugar.

f) **Mover_MercaderíasDelLocal_AlLocal_(cantidad, origen, destino)**, que mueve la cantidad indicada de mercadería del depósito local que está en dirección origen al que está en dirección destino. Si en el depósito origen no hay la cantidad de mercadería necesaria, no se hace nada. Nuevamente se puede suponer que el cabezal se encuentra en el depósito central.

14. El caminante

EN PAPEL. Se puede modelar el paseo de un caminante por el tablero con las siguientes consideraciones para la representación.

- El caminante está representado por entre una a cuatro bolitas azules. La dirección de su paseo es Norte si es una bolita, Este si son dos, Sur si son tres y Oeste si son cuatro.
- Las indicaciones de cambio de dirección se representan con bolitas verdes. Si el caminante llega a una celda con una de estas indicaciones, debe cambiar de dirección. La cantidad de bolitas verdes indica la nueva dirección, con la misma representación de direcciones dadas para el caminante.
- El caminante deja una huella de bolitas negras a su paso, una por cada paso.
- La meta se representa con cualquier número de bolitas rojas. El paseo del caminante termina si llega a la meta.
- La celda actual siempre se encuentra sobre el caminante.
- La única celda con bolitas azules es la del caminante.
- Todas las celdas tienen un máximo de 4 bolitas verdes.
- Las indicaciones llevan al caminante a la meta.

Como ayuda para guiar la división en subtareas, ya se realizó un análisis top-down de la estrategia, y se eligieron ciertas subtareas. Se pide, entonces, implementar los procedimientos y funciones que expresan dichas subtareas, que son los indicados a continuación. Observar que en su gran mayoría, las tareas están presentadas en forma top-down, por lo que es interesante mirarlas todas antes de empezar a implementar, y definir todos los contratos antes de proceder a escribir el código de cada una, ya que las de niveles más alto se pueden servir de las de niveles más bajos.

Además, puede tomarse la siguiente función como primitiva:

```
function direcciónDelCódigo_(código)
    /*
        PROPÓSITO: Describe la dirección correspondiente al
            código dado
        PARÁMETROS:
            * código: Número - codifica una dirección
        PRECONDICIONES: El código está entre 1 y 4
    */
```

Al escribir los contratos, no olvidar establecer las precondiciones necesarias (ya que las mismas no siempre se explicitan en los enunciados).

a) **colorCaminante()**, **colorIndicador()**, **colorHuella()** y **colorMeta()**, que describen los colores con los que se representa cada uno de los elementos nombrados.

b) **LlevarAlCaminanteALaMeta()** que, suponiendo que en el tablero está representado un escenario válido para el caminante, lleva al caminante hasta la meta.

c) **estáEnLaMeta()** que indica si el caminante está o no en la meta.

d) **DejarHuella()** que deja una huella en la celda actual.

e) **DarUnPaso()** que realiza un paso en el paseo del caminante, de acuerdo a las siguientes reglas.

   - Si hay que cambiar la dirección (está sobre una celda indicadora), lo hace.
   - Finalmente, se mueve en la dirección correspondiente.

f) **CambiarDeDirecciónSiHayIndicador()** que cambia la dirección del caminante cuando se encuentra con un indicador.

g) **MoverAlCaminanteHaciaDondeMira()** que mueve el caminante un paso en la dirección hacia la cual está mirando.

h) **hayIndicadorDeCambioDeDirección()** que indica sí en la celda actual hay un indicador de dirección.BOOM

i) **CambiarDirecciónDelCaminanteALaDelIndicador()** que cambia la dirección del caminante para que coincida con la del indicador de la celda actual.

j) **MoverAlCaminanteAl_(dirección)** que mueve al caminante un paso en la dirección dada.