



UNIVERSIDAD  
NACIONAL DE  
HURLINGHAM

## Introducción a la Programación - Práctica 1

Programas y contratos

### 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. Agradecemos a todos los que nos ayudaron con su inspiración.
- **Realizar en papel los ejercicios que así lo indiquen.**

**PRIMEROS PROGRAMAS:**

**ATENCIÓN:** No es necesario que escriba contratos en estos ejercicios. El objetivo es ir entendiendo una idea de los mismos. Se le indicará a partir de qué punto es necesario que comience a escribirlos.

## 1. Reemplazando bolitas

Escribir un programa que reemplace<sup>1</sup> una bolita de color Roja con otra de color Verde en la celda actual. Pruebe su programa en la computadora, modificando el tablero inicial de forma que el programa funcione satisfactoriamente.

## 2. Moviendo bolitas

Escribir un programa que mueva<sup>2</sup> una bolita de color Negro de la celda actual a la celda vecina al Este, dejando el cabezal en la celda lindante al Este.

## 3. Poniendo en vecinas

Escribir un programa que ponga una bolita de color Azul en la celda vecina al Norte de la actual.

## 4. ¿Qué pide el enunciado?

**EN PAPEL** Los siguientes programas resuelven incorrectamente el ejercicio anterior de esta práctica. Para cada uno de ellos, explicar por qué no son correctos. Comparar con la solución dada para el ejercicio anterior.

a.

```
program {  
  Mover(Norte)  
  Poner(Azul)  
}
```

b.

```
program {  
  Poner(Azul)  
  Mover(Este)  
  Mover(Oeste)  
}
```

c.

```
program {  
  Mover(Norte)  
  Poner(Verde)  
  Mover(Sur)  
}
```

d.

```
program {  
  Poner(Azul)  
  Mover(Norte)  
  Mover(Sur)  
}
```

<sup>1</sup> Por reemplazo nos referimos al efecto observado, aunque no exista ningún comando para hacer reemplazos directamente

<sup>2</sup> Nuevamente nos referimos al efecto observado.

## 5. Analizando propósitos

**EN PAPEL** Dado el siguiente programa:

```
program {
  Poner(Verde)
  Sacar(Verde)
  Poner(Azul)
  Poner(Rojo)
}
```

Diversos estudiantes realizaron propuestas para redactar su propósito, y también un profesor realizó explicaciones sobre cada una de estas propuesta. Asociar cada propuesta de propósito para el mismo (indicadas con las letras A, B, etc.) con la explicación que resulta correcta para dicha propuesta (indicadas con los números 1, 2, etc.)

(A) Poner una bolita azul y luego una roja en la celda actual.	(1) No es un propósito, sino una descripción del funcionamiento. Para ser un propósito no debe preocuparse de los estados intermedios, solamente de la transformación final.
(B) Agregar una bolita azul y una roja.	(2) Es un propósito incompleto ya que no establece los colores de las bolitas que se agregan. El propósito debe establecer con precisión la transformación esperada.
(C) Pone una bolita verde y luego la saca, para a continuación poner una bolita azul y una roja.	(3) Es una enunciación correcta del propósito. El orden en que se agregan las bolitas es irrelevante, siempre que la celda actual finalice con una más de cada uno de los colores indicados.
(D) Agregar una bolita roja y una bolita azul en la celda actual.	(4) Es un propósito incompleto, ya que no establece dónde se agregan las bolitas en cuestión. El propósito debe establecer con precisión la transformación esperada.
(E) Agregar dos bolitas en la celda actual.	(5) Es una forma incorrecta de indicar la transformación esperada. Utiliza un lenguaje que sugiere un pensamiento operacional (o sea, centrado en las acciones individuales antes que en la transformación esperada).

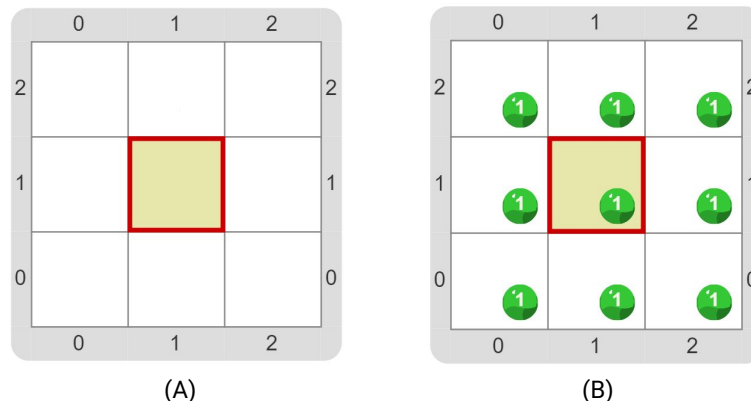
## PROGRAMAS QUE PUEDEN FALLAR:

**ATENCIÓN:** A partir de este momento se pide que escriba los propósitos de su código, aunque no así las precondiciones de los mismos, cosa que iremos practicando en los siguientes ejercicios.

### 6. Cuadrados verdes

Escribir los siguientes programas:

- uno que ponga un cuadrado<sup>3</sup> de tamaño 3 con bolitas de color verde, con centro en la celda inicial (dejando el cabezal en dicha celda al finalizar)
  - ¿Qué ocurre si el tablero ya tenía bolitas verdes? ¿Es esto un problema, o el efecto obtenido es acorde al propósito?
  - ¿Qué ocurriría si desde la celda inicial no hay espacio para colocar las bolitas necesarias para que se cumpla el propósito?
- uno que saque un cuadrado de tamaño 3 con bolitas de color verde (saca una bolita de cada celda), siendo la celda inicial el centro del cuadrado, dejando el cabezal en dicha celda al finalizar.
  - ¿Qué sucede si no hay al menos una bolita verde en cada una de las celdas necesarias? ¿Y si el tablero no tiene el tamaño adecuado?



Efectos de poner y sacar un cuadrado verde. Empezando con el Tablero (A), el programa a) termina en el Tablero (B), mientras que empezando con el Tablero (B) se obtiene el Tablero (A) al ejecutar el programa b).

### 7. ¿Y cuándo funciona?

**EN PAPEL** Discutir en clase cuáles serían las precondiciones para programas cuyo propósito sea:

- Poner 1000 bolitas color Azul en la celda actual.
- Poner una bolita color Rojo en la celda lindante al Este de la celda actual y sacar una bolita Azul de la celda lindante al Oeste de la celda actual.
- Poner un rectángulo de bolitas Negras cuyo tamaño sea 3 filas y 5 columnas, centrado en la celda actual.

**Importante:** no hay que escribir ningún programa, simplemente basta con discutir cuál es la precondición que esperarían de tales programas.

### 8. Analizando precondiciones

**EN PAPEL** La solución propuesta por diversos estudiantes para el punto c. del ejercicio anterior fue corregida por los docentes. Sin embargo, las correcciones se mezclaron, y hay que juntar cada corrección con su precondición correspondiente.

<sup>3</sup> Un cuadrado consiste en una secuencia de celdas que tienen al menos una bolita de un determinado color, y que se extienden de forma vertical la misma cantidad de celdas que en horizontal.

(A) Debe haber al menos una celda al Norte y otra al Sur, y dos celdas al Este y otras dos al Oeste de la celda actual.	(1) Es una precondition incorrecta, pues al no establecer dónde debe ubicarse el cabezal hay tableros de ese tamaño que no sirven.
(B) El rectángulo va a representar a una ventana en el dibujo de una casa.	(2) Es una precondition correcta, pues establece las condiciones mínimas necesarias para que el programa funcione, y define con precisión el conjunto de tableros iniciales que sirven.
(C) El tablero debe tener 3 o más filas, y 5 o más columnas, y el cabezal debe estar al menos a una celda de distancia de los bordes Norte y Sur, y a dos celdas de distancia de los bordes Este y Oeste.	(3) Es otra forma de enunciar correctamente la precondition: todos los tableros del conjunto que la satisfacen sirven, y todos los que no la satisfacen no sirven. Sin embargo, especificar las dimensiones del tablero es innecesario, pues son las distancias a los bordes lo relevante para que el programa no falle.
(D) Debe haber lugar suficiente en las direcciones adecuadas.	(4) Es una precondition correcta pero insuficiente, pues todos los tableros que la satisfacen sirven, pero hay muchos tableros que sirven que no la satisfacen.
(E) Se van a colocar 35 bolitas de color Negro.	(5) Es una precondition insuficiente, pues es demasiado vaga y no permite determinar cuáles son los tableros que sirven.
(F) El tablero debe tener 3 filas y 5 columnas y el cabezal estar ubicado en el centro del mismo.	(6) No es una precondition, sino una aclaración. Una precondition debe hablar sobre los tableros iniciales y no sobre el dominio del problema.
(G) El tablero debe ser de 3 filas por 5 columnas.	(7) No es una precondition, sino una explicación de la tarea. Una precondition debe hablar sobre los tableros iniciales y no sobre las características de lo que se va a realizar.

### 9. Parece pero no es

A continuación hay una serie de precondiciones que un alumno escribió en un examen. Las mismas son todas incorrectas por el mismo motivo. ¿Puede adivinar por qué son incorrectas? Justifique claramente su respuesta

- Se está escribiendo código Gobstones.
- El cabezal se encuentra en la celda actual.
- Existe una celda en el tablero.
- Existe un tablero.
- El cabezal existe y está en la celda actual.

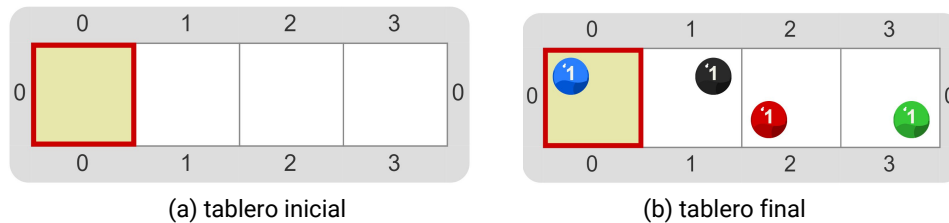
### PROGRAMAS UN POCO MÁS COMPLEJOS:

**ATENCIÓN:** A partir de este momento se pide que escriba los contratos completos, es decir, con propósito y precondiciones (y observaciones si las hubiera).

#### 10. Arcoiris

**EN PAPEL** Escribir un programa que ponga un “arco iris”, poniendo una bolita Azul en la celda actual, una Negra en la celda siguiente al Este, una Roja en la siguiente al Este, y una Verde en la siguiente al Este, dejando el cabezal en la celda inicial.

Podemos ver un ejemplo en la figura, partiendo del tablero (a) se obtendría el (b):



Luego contestar las siguientes preguntas

- ¿Qué sucedería en otros tableros iniciales?
- ¿Cuál es la precondición de este programa?
- ¿Puede darse otro programa diferente que cumpla el mismo contrato?

**PISTA:** en lugar de poner las bolitas en la ida, considere si no lo puede hacer a la vuelta.

- Pruebe ahora sus programas en la máquina para verificar que funcionan, detectar y errores y corregir.

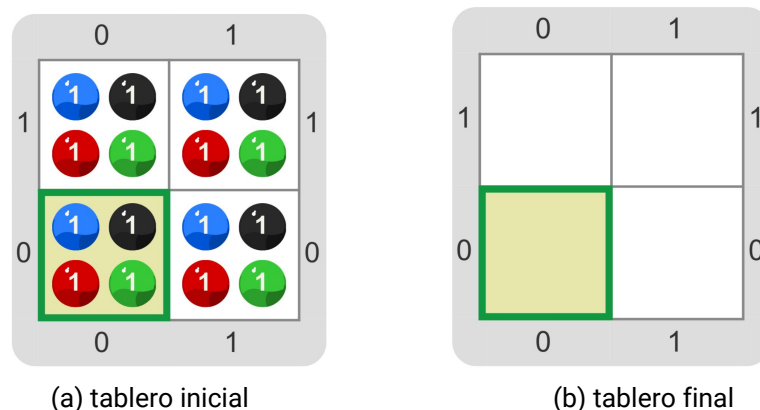
#### 11. Sacando un cuadrado

Escriba un programa que saque del tablero un cuadrado multicolor de dos celdas de lado, donde la celda actual representa el vértice inferior izquierdo del mismo.

Recuerde escribir primero el contrato del programa, y luego el código.

Considere las siguientes preguntas como guía para escribir su programa:

- ¿Que hace el programa? (Determina el propósito del programa)
- ¿Cuándo funciona tal cual se espera? (Determina la precondición del programa)
- ¿Cómo lo hace? (Determina el código del programa)





UNIVERSIDAD  
NACIONAL DE  
HURLINGHAM

## Introducción a la Programación - Práctica 2

Procedimientos y estrategia de solución

### 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. 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.**

## PROCEDIMIENTOS Y ESTRATEGIA DE SOLUCIÓN:

## 1. Iniciales

Escribir un programa que ponga sus iniciales en el tablero usando bolitas color Verde para los nombres y color Rojo para el apellido, dejando el cabezal en la celda inicial. Para escribir sus iniciales, se debe definir cómo se verán las letras en el tablero.

**Reflexionamos:** ¿Cuál es el propósito y las precondiciones del programa? ¿Cómo lo puede ayudar la definición de procedimientos? ¿Qué características tienen que tener los procedimientos para favorecer su adecuada combinación? ¿Cuál es el propósito y las precondiciones de los procedimientos realizados?

## 2. Por arriba

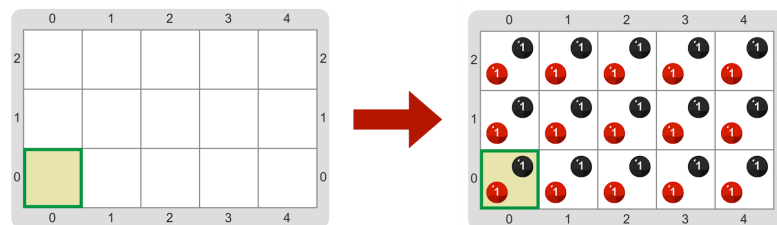
Realizar cada uno de los siguientes puntos, en el orden dado (siguiendo una metodología top-down).

- Escribir un procedimiento **DibujarRectánguloRojoYNegroDe5x3()** cuyo contrato es el siguiente:

```

procedure DibujarRectánguloRojoYNegroDe5x3()
  /*
    PROPÓSITO: Poner un rectángulo sólido de 5 celdas de ancho y 3
    celdas de alto, en la que cada celda tenga una bolita de color
    Rojo y una de color Negro. El cabezal comienza y termina en el
    vértice inferior izquierda del mismo.
    PRECONDICIONES:
    * Hay al menos 4 celdas al Este y 2 celdas al Norte de la celda
    actual.
  */
  
```

**Importante** La metodología a seguir que se debe aplicar es la siguiente. En primer lugar, pensar una estrategia; se sugiere pensar una estrategia que involucre poner líneas de 5 celdas de ancho. Luego se debe definir primero el contrato de un procedimiento llamado **DibujarLíneaRojaYNegraDeTamaño5()** que exprese la subtaska sugerida. Para completar el ejercicio, se debe utilizar este procedimiento auxiliar en la codificación del procedimiento pedido, **DibujarRectánguloRojoYNegroDe5x3()**. El código del procedimiento auxiliar no es parte de este inciso, sino del siguiente.



El rectángulo es sólido. Es decir, la transformación esperada es similar a la que se muestra en la imagen.

- Escribir el procedimiento **DibujarLíneaRojaYNegraDeTamaño5()** que quedó pendiente del ítem anterior.

**Importante** Para ello, se debe seguir la misma metodología que en el caso anterior: en primer lugar pensar una estrategia, luego definir los contratos de los procedimientos que expresan las subtaskas, y por último codificar el procedimiento pedido usando dichas subtaskas (sin su código). Se sugiere que la estrategia involucre a subtaskas llamadas **PonerUnaNegraYUnaRoja()** y **Mover4VecesAlOeste()**.

- Escribir los procedimientos **PonerUnaNegraYUnaRoja()** y **Mover4VecesAlOeste()** que quedaron pendientes del ejercicio anterior. En este caso, los procedimientos se pueden



expresar fácilmente utilizando comandos primitivos, por lo que no es necesario comenzar pensando en posibles subtareas.

### 3. Por abajo

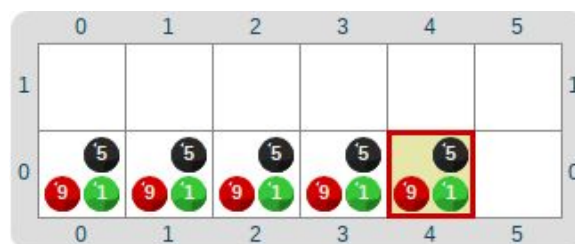
En el ejercicio anterior se trabajó con una metodología conocida como *top-down*: se comienza desde el problema, y se lo va descomponiendo en tareas cada vez más pequeñas, hasta llegar al uso de comandos primitivos. En este ejercicio vamos a proponer la estrategia inversa, *bottom-up*: primero se escriben los procedimientos elementales, y luego se van armando procedimientos cada vez más complejos, hasta construir uno que resuelva el problema original.

Escribir los siguientes procedimientos en el orden dado. Recuerde escribir los contratos correspondientes.

- BIBLIOTECA** `PonerUnaDeCada()` que ponga una bolita de cada color en la celda actual.
- Mover3VecesAlOeste() que mueva el cabezal tres celdas hacia el Oeste.**
- DibujarLíneaMulticolorDeLargo4() que ponga una línea de cuatro celdas hacia el Este en la que cada celda tenga una bolita de cada color. El cabezal debe quedar en la celda inicial. Para ello, debe reutilizar los procedimientos `PonerUnaDeCada()` y `Mover3VecesAlOeste()` (realizados en ejercicios anteriores).**
- DibujarCuadradoMulticolorDeLado4() que ponga un cuadrado sólido de  $4 \times 4$  celdas en la que cada celda tenga una bolita de cada color. El cabezal debe quedar en la celda inicial. Para ello, debe reutilizar el procedimiento `DibujarLíneaMulticolorDeLargo4()`.**
- Reflexionar sobre las diferencias entre ambas metodologías. ¿Cuál de las dos metodologías es más sencilla si no tenemos ninguna pista de cómo resolver el problema? ¿Qué se requiere para poder utilizar una estrategia *bottom-up*? ¿Cuál metodología es más utilizada y por qué?

### 4. Guarda de azulejos

Construir un procedimiento `PonerGuardaDe5Azulejos()`, que arme una "guarda" de 5 azulejos (como las que decoran las paredes). Cada azulejo está conformado por 1 bolita verde, 5 negras y 9 rojas.



Recordar seguir una metodología adecuada para la construcción del código: **COMENZAR** por escribir el contrato completo del procedimiento, luego pensar las subtareas necesarias y darles un nombre adecuado, escribir el contrato de las subtareas, **LUEGO** el código del procedimiento pedido en términos de las subtareas, y **FINALMENTE**, realizar el código de las subtareas siguiendo la misma metodología (que denominamos metodología de construcción de programas *top-down*).

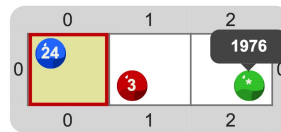
### 5. Día de la Memoria

Utilizando bolitas pueden representarse diversos elementos; un ejemplo de esto es la posibilidad de representar una fecha. Una fecha que los argentinos deberíamos recordar, para no repetirla jamás, es el 24 de marzo de 1976, hoy constituido como Día de la Memoria por la Verdad y la Justicia en Argentina.

Hacer un procedimiento `RegistrarElDíaDeLaMemoria()` que

- en la celda actual, ponga 24 bolitas Azules, que representan el día,

- en la celda lindante al Este de la actual, ponga 3 bolitas Rojas, que representan el mes, y
- en la celda lindante al Este de la anterior, ponga 1976 bolitas Verdes, representando el año.



Recordar que la solución completa incluye la documentación: propósito, precondition y parámetros, los cuales deben ser escritos **ANTES** de escribir el código.

**Importante:** La solución se puede realizar íntegramente utilizando los conceptos vistos en la materia hasta el momento, y no se requieren herramientas adicionales. No debe utilizar conceptos aún no vistos en la teoría para solucionar el problema.

## USANDO Y REUTILIZANDO:

### 6. Guarda de azulejos en L

Hacer un procedimiento **PonerGuardaEnL()**, que arme una guarda en L como muestra la figura, dejando el cabezal en la posición inicial.



¿Pensaste en reutilizar el procedimiento definido antes, o empezaste a escribirlo de nuevo? Dado que ahora se cuenta con la subtarea definida en el ejercicio anterior, la metodología *top-down* se puede enriquecer con la reutilización de procedimientos ya realizados; esto puede hacer que ciertas divisiones en subtareas, que permiten esa reutilización, sean más sencillas que otras.

### 7. Las pirámides de Gobstones

**EN PAPEL** Escribir un programa para cumplir con los propósitos de cada uno de los ítems que se indican más adelante. Para ello, deben utilizarse los procedimientos indicados a continuación.

```

procedure DibujarBase()
  /*
    PROPÓSITO: Dibuja una base de pirámide de 5 celdas de lado
    PRECONDICIONES:
      * La celda actual debe estar vacía
      * Debe haber cuatro celdas vacías al Este del cabezal
  */
procedure DibujarMedio()
  /* PROPÓSITO: Dibuja un sector del medio de pirámide
    de 3 celdas de lado
    PRECONDICIONES:
      * La celda actual debe estar vacía
      * Debe haber dos celdas vacías al Este del cabezal
  */
procedure DibujarPunta()
  /*
    PROPÓSITO: Dibuja una punta de pirámide
    PRECONDICIONES:
      * La celda actual debe estar vacía
  */

```

\*/

**¡Recordar!** Para cada ítem que sigue debe comenzarse redactando el contrato correspondiente, y de ser necesario, se deben definir subtarefas para construir su solución, expresándolas mediante procedimientos (y en ese caso se deben escribir primero los contratos de los mismos y utilizarlos **ANTES** de dar su código).

- Una pirámide.
- Una pirámide invertida.
- Una pirámide estirada a lo alto (con dos segmentos de cada uno).
- Una pirámide gigante (con 11 bloques de base y 6 bloques de altura, donde cada segmento es dos celdas más chico que el que tiene debajo). Para esto deben reutilizarse los procedimientos implementados en los puntos a. y b.

**PISTA:** una pirámide gigante se puede conseguir con 4 pirámides comunes, si se las ubica con cuidado.

## MODELANDO DOMINIOS:

### 8. El bosque, parte 1

En este ejercicio, se usará el tablero para **representar un bosque**. Cada celda representa a una **parcela**. Cada **bolita verde representa un árbol**. Cada **bolita roja representa una semilla**. Una **bolita negra representa una bomba**. Una **bolita azul representa una unidad de nutrientes**.

Escribir los siguientes procedimientos de representación, que hacen lo que su nombre indica. Todos trabajan siempre sobre la celda actual.

a.

```
PonerUnaSemilla()
```

b.

```
PonerUnÁrbol()
```

c.

```
PonerUnaBomba()
```

d.

```
PonerUnNutriente()
```

e.

```
SacarUnaSemilla()
```

f.

```
SacarUnÁrbol()
```

g.

```
SacarUnaBomba()
```

h.

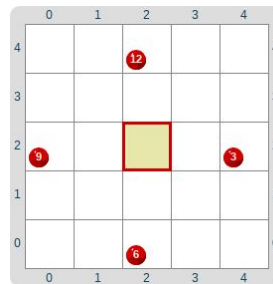
```
SacarUnNutriente()
```

### 9. Reloj Analógico

Dibujar un reloj analógico de agujas en un tablero cuadriculado puede ser un desafío. Una simplificación posible sería representar solamente algunos de los números que aparecen en el mismo: el 12 arriba, el 3 a la derecha, el 9 a la izquierda y el 6 abajo.

Construir un procedimiento **DibujarRelojAnalógicoSimplificado()**, que ponga los números del reloj tal como se indicó, alrededor del casillero actual. El tamaño del reloj será de 2 celdas de "radio" (suponiendo que miramos al reloj como un círculo).

Utilizar el comando **DibujarRelojAnalógicoSimplificado()** en un tablero inicial vacío de 5x5 con la celda inicial en el centro del mismo, es el siguiente:



**¡Recordar!** Escribir el contrato en primer lugar, y en caso de utilizar división en subtareas, en seguir la metodología *top-down* para las mismas (primero su nombre y su contrato, usarlas, y recién luego definir las con la misma metodología).

## 10. La llegada de Nova

**EN PAPEL** En el equipo de programación empezó a trabajar un programador que no tiene formación profesional, y al que todos apodaron Nova<sup>1</sup>. En el código que escribió Nova se encontró un procedimiento que no aplica la técnica de división en subtareas, y tampoco tiene contrato. Para poder integrar el código con el resto, en este equipo se requiere que los programas sigan buenas prácticas de programación. Por eso, se pide corregir este código para que cumpla con las mismas.

```
procedure ConstruirEscaleraAzulDe4Escalones() {
  Poner(Azul) Mover(Este) Poner(Azul) Mover(Norte)
  Poner(Azul) Mover(Este) Poner(Azul) Mover(Norte)
  Poner(Azul) Mover(Este) Poner(Azul) Mover(Norte)
  Poner(Azul) Mover(Este) Poner(Azul)
}
```

Comenzar por redactar el contrato y la precondition correspondientes.

- Luego aplicar la técnica de división en subtareas expresadas con procedimientos. Dado que no se trata de hacer el código de nuevo, se propone identificar partes de código que se repitan, darles un buen nombre y extraer esa parte de la estrategia en procedimientos auxiliares. Por supuesto, cada procedimiento auxiliar debe tener un nombre adecuado, y debe documentarse su propósito y precondition.
- ¿Puede apreciar las ventajas de contar con el contrato y de expresar la estrategia como parte del código en forma de procedimientos? Hay más de una forma de dividir en subtareas. Comparar la solución propuesta con la de otros compañeros, y discutir las ventajas y desventajas de la dada con las que resulten diferentes.

<sup>1</sup> Todos creían que era porque era un Novato, pero en realidad es porque su código suele tener eventos cataclísmicos inesperados, tal como el fenómeno astronómico del mismo nombre...



UNIVERSIDAD  
NACIONAL DE  
HURLINGHAM

## Introducción a la Programación - Práctica 3

### Repeticiones 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. 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 SIMPLE BÁSICA:**

## 1. Moviendo tres veces al Norte

Escribir un procedimiento **Mover3VecesAlNorte()** que mueva el cabezal tres posiciones al Norte de la actual.

## 2. Moviendo tres veces al Este

Escribir un procedimiento **Mover3VecesAlEste()** que mueva el cabezal tres posiciones al Este de la actual.

- ¿Qué similitudes encuentra entre este procedimiento y el anterior?
- ¿Qué cambiaría si quisiera hacer un procedimiento que mueva tres veces al Oeste, o tres al Sur?

## 3. Y Ahora para algo completamente distinto

Escribir un procedimiento **Poner6DeColorNegro()** que ponga 6 bolitas de color Negro en la celda actual.

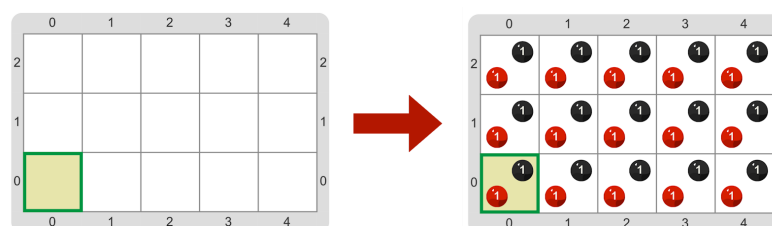
## 4. Y 6 Verdes

Escribir un procedimiento **Poner6DeColorVerde()** que ponga 6 bolitas de color Verde en la celda actual.

**REPETICIÓN SIMPLE EN SUBTAREAS:**

## 5. Dibujando un rectángulo con repeticiones

Escribir un procedimiento **DibujarRectánguloRojoYNegroDe5x3()** que dibuje un rectángulo sólido de 5 celdas de largo por 3 de alto, similar al realizado en **"P2. 2. Por Arriba"**, pero esta vez, utilice repetición para solucionar el problema.



El rectángulo es sólido. Es decir, la transformación esperada es similar a la que se muestra en la imagen

## 6. Pintando el tablero

Escribir un procedimiento **PintarElTableroDeAzul()** que, asumiendo que el tablero tiene 10 celdas de largo y 7 celdas de alto, pinte absolutamente todo el tablero con bolitas azules, dejando exactamente una bolita azul en cada celda.

- ¿Cuál es la precondition del procedimiento?
- ¿Se le ocurre otra estrategia para resolver el problema?

**Importante** Recuerde que la estrategia de solución debe quedar clara a partir de la lectura del código. Use subtareas con nombres apropiados para dicho objetivo.



UNIVERSIDAD  
NACIONAL DE  
HURLINGHAM

## Introducción a la Programación - Práctica 4

Parámetros

### 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. 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 SIMPLE Y PARÁMETROS:**

## 1. Moviendo tres veces a donde quieras

Escribir un procedimiento **Mover3VecesAl\_(direcciónAMover)** que dada una dirección **direcciónAMover** mueva el cabezal tres posiciones en dicha dirección.

- ¿Qué hay que hacer ahora para mover el cabezal tres veces al Norte?
- ¿Qué beneficios trae el uso de parámetros?
- ¿Cuántos procedimientos puedo ahorrarme haciendo un único procedimiento con un parámetros?

**¡Recordar!** No olvidar escribir el contrato del procedimiento ANTES de realizar el código (y que los parámetros son parte del mismo); también discutir la precondition escrita con sus compañeros para verificar que la misma es adecuada y correcta.

## 2. Y 6 de lo que quieras

Escribir un procedimiento **Poner6DeColor\_(colorAPoner)** que dado un color **colorAPoner** ponga 6 bolitas del color dado.

**MÚLTIPLES PARÁMETROS Y PARÁMETROS COMO ARGUMENTOS**

## 3. Moviendo y poniendo

Escribir un procedimiento **Poner\_Al\_(colorAPoner, direcciónDondePoner)** que dado un color **colorAPoner** y una dirección **direcciónDondePoner**, ponga una bolita del color dado en la celda vecina en la dirección dada, dejando el cabezal en dicha celda.

- ¿Cuántos casos distintos habría que considerar si no se usaran parámetros en este caso?
- ¿Cómo debería invocar al procedimiento para que ponga una bolita Azul en la celda al Norte?
- ¿Y si quisiera una Azul y una Roja?

## 4. Reemplazando colores

Escribir **ReemplazarUnaDe\_Por\_(colorAReemplazar, colorPorElCualReemplazar)**, un procedimiento que dado un primer color **colorAReemplazar** y un segundo color **colorPorElCualReemplazar**, reemplaza una bolita del primer color por una del segundo color (En la celda actual).

## 5. El regreso de Nova

¿Se acuerdan de Nova? Es el nuevo compañero del equipo, que no tiene formación profesional, y su código está lleno de malas prácticas de programación. Esta vez, en su código se encontró un procedimiento que tiene el código hecho, pero no el contrato, y no está indentado.

- En primer lugar, se pide corregir los errores de Nova. No olvidar indicar en qué posición queda el cabezal, ni de establecer para qué son los parámetros. El código de Nova es el siguiente:

```
procedure Poner_ADistancia3Al_(color, dirección) {  
  Mover3VecesAl_(dirección) Poner(color) }
```

- En segundo lugar, se pide contestar la siguiente pregunta que realizó Nova: ¿Cuál es la relación entre el parámetro dirección de este procedimiento con el parámetro direcciónAMover de Mover3VecesAl\_?



## 6. Los 3 puntos de Nova

Al continuar revisando código, encontramos otro procedimiento de Nova que carece de contrato y no sigue buenas prácticas de programación.

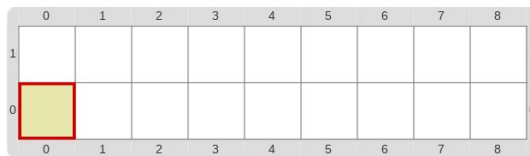
- a. Escribir el contrato faltante y corregir las malas prácticas:

```
procedure Pintar3PuntosAzules() {
  Poner(Azul)
  Poner_ADistancia3A1_(Azul, Este)
  Poner_ADistancia3A1_(Azul, Este)
  Mover3VecesA1_(Oeste) Mover3VecesA1_(Oeste)
}
```

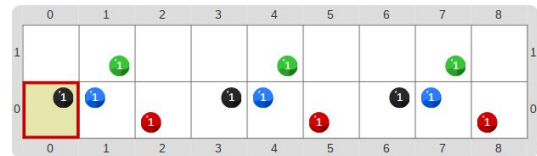
- b. Ayudar a Nova a contestar la siguiente pregunta: ¿Hay alguna relación entre los parámetros de **Poner\_ADistancia3A1\_** y **Mover3VecesA1\_**?
- c. Dado que Nova no pensó en código general, basándose en el procedimiento corregido recién, generalizarlo para que, dado un color **colorPunto**, dibuje los puntos de ese color. El nuevo procedimiento debe llamarse **Pintar3Puntos\_**.

## 7. Multi Arcoiris

Utilizando el procedimiento **Pintar3Puntos\_**, construir el procedimiento **PintarArcoIris()** que ponga el tablero de la derecha cuando el tablero inicial es el de la izquierda. ¡A no ser como Nova, y empezar escribiendo el contrato!



(a) Tablero inicial



(b) Tablero final

## 8. Y ahora, muchas Rojas

Escribir un procedimiento **Poner\_DeColorRojo(cantidadAPoner)** que dado un número **cantidadAPoner**, ponga tantas bolitas como se indica de color Rojo en la celda actual.

- a. ¿Cuántos casos posibles habría que considerar si no se usaran parámetros en este caso o el anterior?

## 9. Poner de a muchas

**BIBLIOTECA** Escribir un procedimiento **Poner\_DeColor\_(cantidadAPoner, colorAPoner)** que dado un número **cantidadAPoner** y un color **colorAPoner**, ponga tantas bolitas como se indica del color dado de la celda actual.

## 10. Moviendo tantas veces como quieras a donde quieras

**BIBLIOTECA** Escribir **Mover\_VecesA1\_(cantidadAMover, direcciónAMover)**, un procedimiento que dado un número **cantidadAMover** y una dirección **direcciónAMover** mueva el cabezal tantas veces como la dada en dicha dirección.

## 11. Sacar de a muchas

**BIBLIOTECA** Escribir un procedimiento **Sacar\_DeColor\_(cantidadASacar, colorASacar)** que dado un número **cantidadASacar** y un color **colorASacar**, saque tantas bolitas como se indica del color dado de la celda actual.

## PROCEDIMIENTOS DE DOMINIO CON PARÁMETROS

### 12. Escribiendo fechas

Construir un procedimiento **EscribirFechaConDía\_Mes\_Año\_(día, mes, año)**, que permita representar cualquier fecha dados el día, mes y año (como números). La representación debe ser la misma utilizada en el ejercicio anterior donde se registró el Día de la Memoria (Azul para el día, Rojo para el mes y Verde para el año, en tres celdas hacia el Este).

**¡Recordar!** Debe comenzarse por escribir el contrato; en este caso puede resultar útil escribir también una observación con la representación a utilizar.

### 13. Listado de fechas

Construir un programa que escriba un listado vertical con las siguientes fechas:

- Inicio de la Reforma Universitaria.
- Reglamentación del voto femenino en Argentina.
- Fecha en la que ocurrieron los hechos conmemorados en el Día Internacional de los Trabajadores.
- Creación del Ministerio de Ciencia y Tecnología argentino.
- Primera celebración del Día de la Mujer.

¿Es necesario pensar procedimientos para escribir cada una de las fechas o sirve algo de lo realizado con anterioridad?

### 14. Reloj analógico simplificado generalizado

Generalizar el ejercicio del reloj analógico simplificado de la práctica anterior para que se pueda pasar el radio como parámetro. O sea, se le pide escribir un procedimiento llamado **DibujarRelojAnalógicoSimplificadoDeRadio\_(radio)** que ponga los números del reloj como en el programa original, pero donde el radio recibido por parámetro indica la distancia al centro del reloj: mientras más grande es el radio, más alejados están los números del centro.

Por ejemplo, el programa del ejercicio anterior podría obtenerse invocando al procedimiento con el comando **DibujarRelojAnalógicoSimplificadoDeRadio\_(2)**.

### 15. Mis iniciales salen fácil

**EN PAPEL** Escribir un programa que escriba sus iniciales en el tablero utilizando la primitiva de dibujo **DibujarLíneaHaciaEl\_DeLargo\_** que se supone primitiva. También puede utilizarse procedimientos que hayan sido marcados como **BIBLIOTECA** hasta ahora.

**Importante:** ¿Hace falta que sigamos diciendo que debe primero escribirse el contrato y dividir el trabajo en subtarear? ¿Qué tal una subtarea para cada letra? ¿Será razonable hacer una excepción a la regla de nombrar a los procedimientos con verbos, para que el programa sea mucho más fácil de leer?

```

procedure DibujarLíneaHaciaEl_DeLargo_(dirección, largoDeLaLínea)
  /*
    PROPÓSITO:
      * Dibuja una línea de longitud **largoDeLaLínea** en dirección **dirección**,
        dejando el cabezal en la posición inicial.
    PARÁMETROS:
      * largoDeLaLínea: Número - La longitud de la línea que se dibuja
      * dirección : Dirección - La dirección hacia la que se dibuja la línea
    PRECONDICIONES:
      * La celda actual está vacía y debe haber al menos **largoDeLaLínea** celdas
  en
      dirección dirección.
  */
  
```

## 16. Corrigiendo el cuadradito de Nova

**EN PAPEL** ¡Nova volvió a hacer de las suyas! Esta vez, el código que escribió no funciona y hay que corregirlo. Parece que uno de los problemas de corrección está en el alcance de los parámetros, pero no es el único. Aprovechar también para mejorar los nombres de los procedimientos, y también para generalizar el tamaño del cuadrado

```

procedure DibujarCuadradito_De2(colorDelCuadrado) {
  /*
    PROPÓSITO: Dibuja un cuadrado sólido de 2x2 de color **colorDelCuadrado**.
    PARÁMETROS:
      * colorDelCuadrado: Un color del cual se dibujará el cuadrado.
    PRECONDICIONES:
      * Existen al menos 1 celda al Norte y 1 al Este de la actual.
  */
  Línea2()
  Mover(Norte)
  Línea2()
  Mover(Sur)
}

procedure Línea2() {
  /*
    PROPÓSITO: Dibujar una línea de longitud 2 de color **colorCuadrado**
    PRECONDICIONES: existe al menos 1 celda al Este de la actual
  */
  Poner(colorCuadrado)
  Mover(Este)
  Poner(colorCuadrado)
  Mover(Oeste)
}
  
```

## 11. El bosque, parte 2

Continuaremos representando el bosque que comenzamos en la práctica anterior. Esta vez queremos ser capaces de poner o sacar múltiples elementos de una sola vez.

**Importante:** para realizar este ejercicio se espera haya realizado la parte 1 de la Práctica 2, si aún no lo hizo, se recomienda volver y realizar el mismo previo a solucionar el ejercicio actual.

a.

Poner\_Semillas(cantidadDeSemillasAPoner)

b.

Sacar\_Semillas(cantidadDeSemillasASacar)

c.

Poner\_Árboles(cantidadDeÁrbolesAPoner)

d.

Sacar\_Árboles(cantidadDeÁrbolesASacar)

e.

Poner\_Nutrientes(cantidadDeNutrientesAPoner)

f.

Sacar\_Nutrientes(cantidadDeNutrientesASacar)



UNIVERSIDAD  
NACIONAL DE  
HURLINGHAM

## Introducción a la Programación - Práctica 5

Expresiones y Tipos

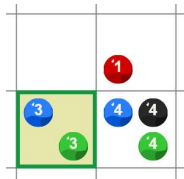
### CONSEJOS:

- 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. 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.**

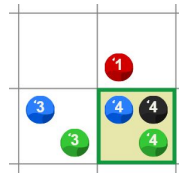
## EXPRESIONES Y TIPOS:

## 1. Mis primeras expresiones

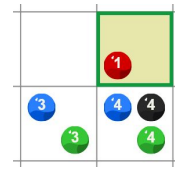
**EN PAPEL** Indicar el valor y el tipo que representan las expresiones dadas en los ítems en cada uno de los tableros A , B y C, suponiendo definido un procedimiento con el contrato dado al final.



(A)



(B)



(C)

a.

```
nroBolitas(Negro) + nroBolitas(Azul)
```

b.

```
opuesto(opuesto(Este))
```

c.

```
nroBolitas(siguiete(Azul))
```

d.

```
2 * nroBolitas(colorAImitar)
```

Para este último supondremos que la expresión aparece dentro del cuerpo del procedimiento con el siguiente contrato:

```
procedure PonerElDobleDe_QueDe_(colorAPoner, colorAImitar)
/*
  PROPÓSITO: Poner bolitas del color **colorAPoner** en
             una cantidad que sea el doble de las que hay del
             color **colorAImitar** en la celda actual.
  PARÁMETROS:
    * colorAPoner : Color - color del que se
                    pondrán bolitas.
    * colorAImitar : Color - color del que se
                    mirará cuántas bolitas hay en la
                    celda actual.
  PRECONDICIONES:
    * Ninguna
*/
```

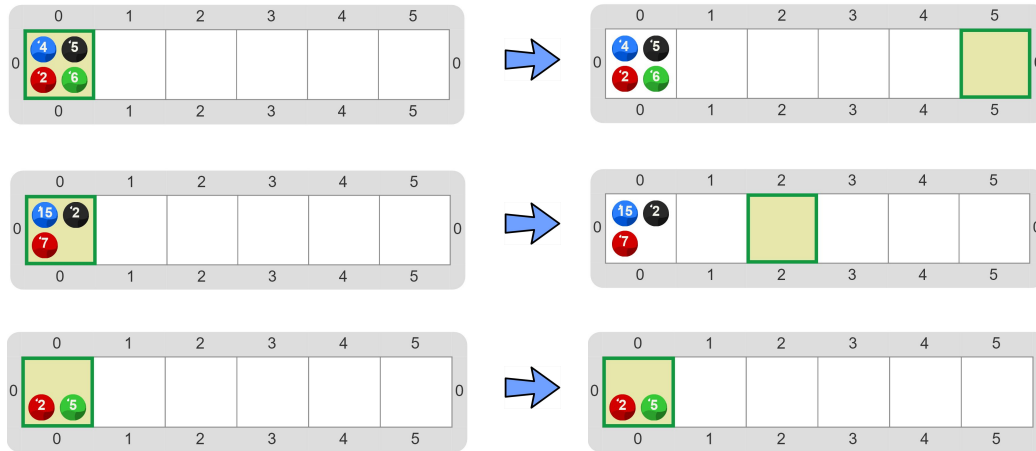
Y del cual sabemos fue invocado como:

```
PonerElDobleDe_QueDe_(Rojo, Verde)
```

**Reflexionamos** ¿Podemos saber a priori el valor de una expresión si desconocemos el estado del tablero? ¿Podemos conocer su tipo? ¿Puede variar el tipo de una expresión si varía el tablero?

## 2. Moviendo según me indican las bolitas

Escribir el procedimiento **Mover\_SegúnColor\_(dirección,color)**, que mueve el cabezal en la dirección dada tantas celdas como bolitas de color dado hay en la celda actual. Como ejemplos se ofrecen los resultados de evaluar el comando **Mover\_SegúnColor\_(Este, Negro)**, en diferentes tableros iniciales.



**Importante:** En el último caso, como la celda no tiene bolitas negras (o sea tiene 0 bolitas negras), entonces el cabezal se mueve 0 celdas hacia el Este (O sea, no se mueve). Para probar correctamente su código, pruebe pasando como argumento otras direcciones y colores.

## 3. El bosque, parte 3

En este ejercicio continuaremos con nuestro bosque, esta vez colocando semillas y árboles en la celda lindante hacia alguna dirección, y dejando el cabezal en la celda inicial.

a.

```
Poner_SemillasAl_(cantidadDeSemillas, direcciónAPoner)
// deja el cabezal en la celda inicial
```

b.

```
Sacar_ÁrbolesAl_(cantidadDeÁrboles, direcciónASacar)
// deja el cabezal en la celda inicial
```

c.

```
Sacar_SemillasEnDiagonalAl_Y_(cantidadDeSemillas, primeraDirDiagonal, segundaDirDiagonal)
// deja el cabezal en la celda inicial
```

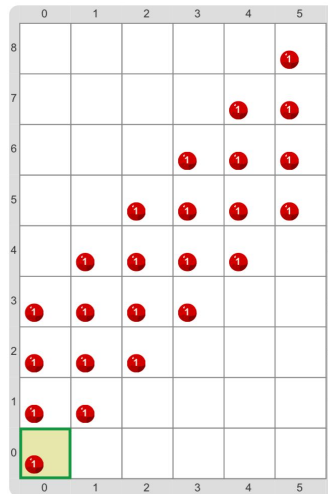
d.

```
Sacar_ÁrbolesEnDiagonalHorariaAl_(cantidadDeÁrboles, direcciónDiagonal)
// la diagonal horaria de una dirección es aquella dada por la dirección y su dirección siguiente.
// Ej. la diagonal horaria de Norte es Norte-Este, la de Sur es Sur-Oeste.
// deja el cabezal en la celda inicial
```

## LAS EXPRESIONES SE MEZCLAN EN EL CÓDIGO:

## 4. La banda de la gloriosa River Plate

Escribir **DibujarLaBandaGloriosaDeAncho\_(ancho)** que dibuja una banda diagonal de color Rojo de cuatro celdas de alto y de tantas celdas de largo como indique el parámetro ancho (dibujando hacia arriba y hacia la derecha). El procedimiento debe poder ser ejecutado en tableros en donde la banda entra justa en el tablero como se muestra a continuación:



La imagen muestra el resultado de ejecutar el procedimiento como **DibujarLaBandaGloriosaDeAncho\_(6)**, con el cabezal posicionado en la esquina Sur-Oeste del tablero al inicio.

**Importante:** Sí la banda tiene 6 celdas de largo, el argumento pasado debe ser 6, no 5. Tenga en cuenta que deberá utilizar expresiones en algún lugar de su código para solucionar el problema.

## 5. ¡A la batalla!, parte 1

Suponiendo que se está programando un juego 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), escribir los siguientes procedimientos:

- EnviarAliadosParaDuplicarEnemigos()**, que agrega soldados aliados en la celda actual en cantidad suficiente para que haya el doble de aliados que de soldados enemigos.
- PelearLaBatalla()**, que simula una batalla, suponiendo que hay suficiente cantidad de soldados aliados como para ganar la batalla. Durante una batalla, 2 soldados enemigos pelean contra 3 soldados aliados y todos mueren. Por ejemplo, si hay 6 enemigos y 10 aliados, mueren los 6 enemigos y 9 de los aliados; si hay 10 enemigos y 21 aliados, mueren los 10 enemigos y 15 soldados aliados.

**PISTA:** ¿Qué cuenta hay que hacer para saber cuántos soldados aliados morirán?

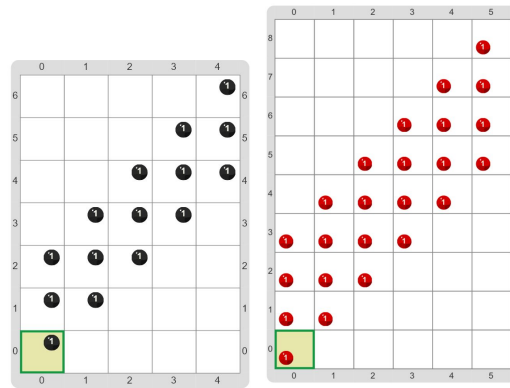
## 6. Sacando todas las de un color

**BIBLIOTECA** Escribir un procedimiento **SacarTodasLasDeColor\_(colorASacar)**, que quite de la celda actual todas las bolitas del color indicado por el parámetro.

**PISTA:** Considerar utilizar el procedimiento **Sacar\_DeColor\_**, definido en la práctica anterior. ¿Qué argumentos se le deberían pasar?

## 7. La banda ahora es para todos

Los hinchas de otros clubes se quejaron de que la banda que hicimos solo vale para River, y quieren poder hacer otras bandas. Escribir entonces **DibujarBandaDeAlto\_YAncho\_DeColor\_(alto, ancho, color)** que dibuja una banda diagonal con los parámetros dados.



La imagen de la izquierda muestra el resultado de ejecutar el procedimiento como **DibujarBandaDeAlto\_YAncho\_DeColor\_(3, 5, Negro)**, mientras que el tablero de la derecha muestra el resultado de ejecutarlo como **DibujarBandaDeAlto\_YAncho\_DeColor\_(4, 6, Rojo)**, siempre partiendo de tableros vacíos con el cabezal en el origen<sup>1</sup>.

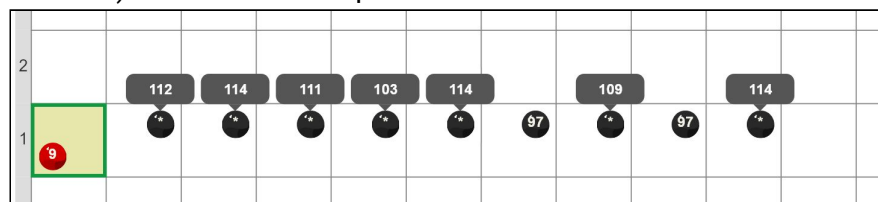
**Importante:** Nuevamente, debe seguir funcionando el código para casos de borde.

## 8. Aprendiendo a leer y escribir

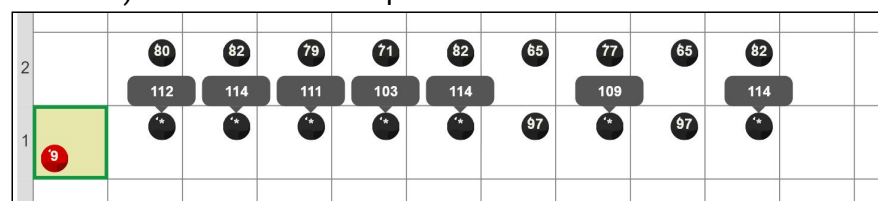
Hacer el procedimiento **PasarPalabraActualAMayúsculas()** que suponiendo que en la fila actual se codifica una palabra en minúsculas usando bolitas, ponga la misma palabra en mayúsculas en la fila al Norte.

- Cada letra se representa con una cantidad diferente de bolitas negras, según un código numérico llamado ASCII.
- En la celda más al Oeste de la fila actual se codifica la cantidad de letras de la palabra, usando bolitas rojas.
- La primera letra de la palabra está en la celda lindante al Este de la que contiene la cantidad de letras.
- En el código ASCII si las letras mayúsculas se codifican con un número N entonces la misma letra minúscula se representa con N+32 (ej. la 'a' minúsculas se representa con el número 97 y la 'A' mayúsculas, con el 65).
- El cabezal se encuentra en la celda más al Oeste de una fila donde hay una palabra representada.

Ejemplo de (fragmento de) un tablero inicial posible:



Ejemplo del (fragmento de) tablero final correspondiente al anterior:



**Importante:** ¿Cómo comenzar la resolución? En cada procedimiento, ¿qué parte debe escribirse primero?

<sup>1</sup> El origen es la denominación que utilizamos para la celda en la fila 0 y la columna 0.





UNIVERSIDAD  
NACIONAL DE  
HURLINGHAM

## Introducción a la Programación - Práctica 6

### Alternativas Condicionales

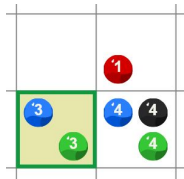
#### CONSEJOS:

- 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 Aloí 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.**

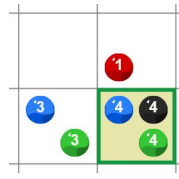
## EXPRESIONES BOOLEANAS:

## 1. Mis primeros booleanos

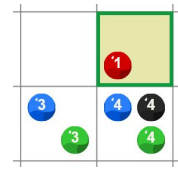
**EN PAPEL** Las siguientes expresiones son todas expresiones que representan booleanos. Indicar el valor de cada una evaluada para cada uno de los tableros A, B y C.



(A)



(B)



(C)

a.

```
not hayBolitas(Rojo)
```

b.

```
puedeMover(Sur) && puedeMover(Oeste)
```

c.

```
puedeMover(Sur) || puedeMover(Oeste)
```

d.

```
not puedeMover(Sur) && puedeMover(Oeste)
```

e.

```
nroBolitas(Negro) == nroBolitas(Azul)
&&
nroBolitas(Negro) == nroBolitas(Verde)
```

f.

```
puedeMover(opuesto(opuesto(dirección)))
```

suponiendo que, por alguna razón, esta expresión aparece dentro del cuerpo del procedimiento siguiente:

```
procedure Mover_SegúnColor_(dirección, color)
/*
  PROPÓSITO: Mover el cabezal en la dirección dada
             tantas celdas como el número de bolitas del
             color dado haya en la celda actual
  PRECONDICIONES: Hay al menos tantas celdas en la
                   dirección dada como número de bolitas del color
                   dado en la celda actual
  PARÁMETROS:
    * dirección, una dirección, hacia donde moverse
    * color, un color para saber la cantidad a moverse
*/
```

Y considerando que el mismo fue invocado como:

```
Mover_SegúnColor_(Norte, Verde)
```

- g. ¿En qué situaciones al invocar **Mover\_SegúnColor\_** la siguiente expresión sería verdadera, en caso de aparecer dentro del cuerpo del procedimiento?

```
not puedeMover(dirección)
&&
not puedeMover(opuesto(dirección))
```

## 2. Definiendo mis primeros booleanos

Definir expresiones que sean verdaderas (describen el valor de verdad Verdadero) para cada uno de los siguientes casos.

- Cuando la celda actual tiene más de 5 bolitas de color Rojo.
- Cuando la celda actual tiene al menos 9 bolitas en total entre rojas y negras.
- Cuando la celda actual es la esquina Norte-Este.
- Cuando la celda actual está vacía.
- Cuando hay una sola celda en el tablero.

## ALTERNATIVA CONDICIONAL:

### 3. Sí se puede, sí se puede...

Escribir los siguientes procedimientos, recordando no mezclar niveles de abstracción del problema, para lo cual puede ser necesario definir otros procedimientos y/o funciones.

- SacarUnaFicha\_SiSePuede(colorDeLaFicha)** que, dado el **colorDeLaFicha** que debe sacarse, saque una ficha siempre y cuando la misma esté en la celda. Si no hubiera fichas del color dado, el procedimiento no hace nada. Si hubiera varias fichas, solo debe sacar una.  
**OBSERVACIÓN:** cada ficha se representa con una bolita del color correspondiente.
- DesempatarParaElLocal\_Contra\_(colorDelLocal, colorDelVisitante)** que, dados los colores de dos jugadores, cuyos puntos se representan mediante la cantidad de bolitas del color del jugador, otorgue un punto al jugador con color **colorDelLocal** solamente en el caso en que la celda actual contiene la misma cantidad de bolitas de ambos colores.
- ExpandirBacteriaDeLaColonia()**, que siempre que en la celda actual haya un cultivo de bacterias y haya suficientes nutrientes, agregue exactamente una bacteria más y consuma nutrientes, a razón de dos nutrientes por bacteria expandida; si no hay bacterias o no hay suficientes nutrientes, no hace nada. Las bacterias se representan con bolitas Verdes y los nutrientes con bolitas Rojas.
- PonerFlecha\_AlNorteSiCorresponde(colorDeLaFlecha)**, que dado un color para representar flechas, ponga una flecha al Norte si existe espacio para moverse en esa dirección. Las flechas serán representadas con una bolita del color dado.

### 4. Hacer solo si...

La combinación de parámetros y expresiones booleanas es interesante.

- BIBLIOTECA** Escribir un procedimiento **Poner\_Si\_(color, condición)** que dado un color y un valor de verdad llamado condición, ponga en la celda actual una bolita del color dado si el valor de verdad de la condición es verdadero, y no lo ponga si no.  
**EJEMPLO:** **Poner\_Si\_(Rojo, nroBolitas(Rojo) == 2)** solamente pone una bolita roja cuando hay exactamente dos rojas en la celda actual.
- BIBLIOTECA** Escribir el procedimiento **Sacar\_Si\_(color, condición)** que actúa de forma similar al anterior, pero ahora sacando bolitas si la condición se cumple.

- c. **BIBLIOTECA** Escribir el procedimiento **Mover\_Si\_(dirección, condición)** que actúa de forma similar a los anteriores, pero ahora moviendo solo si se cumple la condición dada..
- d. ¿Que beneficios trae tener los procedimientos **Sacar\_Si\_** y **Poner\_Si\_** contra utilizar if en cada caso?

#### 5. ¡Nuevamente Nova tiene problemas!

- a. **EN PAPEL** Como Nova sigue confundido con las buenas prácticas de programación, nos consultó sobre cuál de las siguientes dos soluciones que se le ocurrieron es la correcta. El problema es sacar exactamente 8 bolitas de color Azul y la precondition, como es de esperar, es que haya al menos 8 bolitas azules en la celda actual. Explicarle a Nova cuál es la correcta, y por qué la otra no es una buena opción.

```

procedure SacarExactamente8BolitasAzulesOpciónA() {
  /*
    PROPÓSITO: Sacar exactamente 8 bolitas de color Azul
    PRECONDICIONES: Hay al menos 8 bolitas de color azul
  */
  repeat (8) {
    Sacar(Azul)
  }
}

procedure SacarExactamente8BolitasAzulesOpciónB() {
  /*
    PROPÓSITO: Sacar exactamente 8 bolitas de color Azul
    PRECONDICIONES: Hay al menos 8 bolitas de color azul
  */
  repeat (8) {
    Sacar_Si_(Azul, hayBolitas(Azul))
  }
}
  
```

- b. **EN PAPEL** Ayudar a Nova a generalizar este procedimiento, escribiendo **SacarExactamente\_BolitasDeColor\_(cantidadASacar, colorASacar)**

#### 6. ¿Vamos al banco? - Parte 1

En este ejercicio utilizaremos el tablero de Gobstones para representar cuentas bancarias. Cada celda representará a una cuenta bancaria, y en cada una de ellas puede haber dinero en distintas monedas, que representaremos con distintos colores:

- bolitas negras para pesos argentinos.
- bolitas verdes para dólares estadounidenses.
- bolitas azules para euros.
- bolitas rojas para yuanes chinos.

Se pueden hacer tres operaciones: depósitos, extracciones y conversiones a divisa extranjera. Las extracciones pueden hacerse en cualquier moneda, pero los depósitos siempre serán en pesos.

En el caso en que se quiera depositar un monto en una moneda extranjera, se aplicará automáticamente la conversión a pesos según el precio de venta dado en la siguiente tabla:

Precios de venta
------------------

1 dólar	80 pesos
1 euro	90 pesos
1 yuan	12 pesos


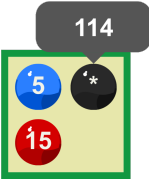

En cuanto a la conversión a divisa extranjera, el banco actualmente aplica las siguientes tarifas para la compra de divisa:

Precios de compra	
100 pesos	1 dólar
115 pesos	1 euro
17 pesos	1 yuan




Realizar los siguientes procedimientos para poder manipular la cuenta:

- a. **Depositar\_EnMoneda\_ComoPesos(cantidadADepositar, moneda)**, que dada una cantidad de dinero a depositar y un color que representa la moneda en la que está representado ese monto, agrega a la cuenta la cantidad de pesos equivalente a lo indicado para depositar. En este procedimiento hay que aplicar la conversión indicada para el precio de venta.


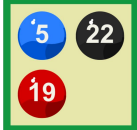
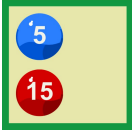
Ej.

		
Celda inicial	Depositar_EnMoneda_ComoPesos(2, Rojo)	Depositar_EnMoneda_ComoPesos(5, Verde)

- b. **ExtraerHasta\_EnMoneda\_(cantidadAExtraer, moneda)**, que dada una cantidad de dinero a extraer y un color que representa la moneda en la que se va a extraer, remueve de la cuenta la cantidad que se indica. Si no hubiera tanto dinero como el solicitado, se extrae todo lo que haya.

		
Celda inicial	ExtraerHasta_EnMoneda_(5, Rojo)	ExtraerHasta_EnMoneda_(10, Azul)

- c. **ConvertirHasta\_PesosA\_(pesosAConvertir, moneda)**, que dada una cantidad de pesos a convertir y un color que representa la moneda en la cual se quiere convertir, remueve los pesos de la cuenta y agrega la moneda solicitada. Si en la cuenta hubiera menos pesos de lo solicitado, se convierte todo lo que haya.

		
Celda inicial	ConvertirHasta_PesosA_(68, Rojo)	ConvertirHasta_PesosA_(100, Verde)

El último ejemplo es interesante: se piden convertir 100 pesos a dólares pero no hay 10 pesos en la cuenta, por lo que se va a intentar convertir el total de pesos que haya, 90. Con 90 pesos, no se llega a comprar ningún dólar, y como Gobstones solo trabaja con números enteros, no es posible tener medio dólar, por lo que queda en cero dólares.

- d. **RealizarCorridaCambiaria()**, que dado un tablero de 1 única fila y 10 columnas, donde cada celda representa una cuenta bancaria, se realiza una corrida cambiaria, donde en cada cuenta se cambia la totalidad de los pesos a dólares.



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 Aloí 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 subtarefas, de forma que las expresiones utilizadas resulten fáciles de entender. Recordar además escribir los contratos.

- La cantidad total de bolitas de la celda actual.
- Sí hay más de 5 bolitas en total en la celda actual.
- Sí hay exactamente 5 bolitas en la celda actual.
- Sí hay al menos 5 bolitas en la celda actual.
- Sí hay bolitas de todos los colores en la celda actual.
- Si la celda actual está vacía.
- 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.

- esCeldaVacía()**, que indica si la celda actual se encuentra vacía.
- tieneUnaDeCada()**, que indica si la celda actual tiene al menos una bolita de cada color.
- 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).

- cantidadDeSoldadosDe\_(colorDelEjército)**, que describe la cantidad de soldados de la celda actual del ejército dado.
- 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?
- esCeldaIndefensa()** que indica si no hay soldados en la celda actual.
- estadoDeEmergencia()** que indica si existen más de 100 soldados enemigos, y además la celda está indefensa.
- 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.
- 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 precondition 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 **distanciaAPlaneta** celdas en dirección **direcciónAPlaneta**, indica si hay un planeta a **distanciaAPlaneta** en la dirección **direcciónAPlaneta** y si el combustible es suficiente para llegar al mismo. La nave consume una única unidad de combustible 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.



UNIVERSIDAD  
NACIONAL DE  
HURLINGHAM

## Introducción a la Programación - Práctica 8

Repetición Condicional y Recorridos

### 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 Aloí 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 *hasta 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 parámetro **color** de cada una de las celdas de la fila actual<sup>1</sup>. 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).

- Definir el procedimiento, como siempre, comenzando por establecer el contrato, y luego recién el código.
- ¿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.
- Al recorrer la fila, ¿en qué dirección se movió el cabezal? ¿Podría haberse movido en la dirección opuesta?
- A partir de la respuesta anterior, ¿de cuántas formas posibles se puede realizar el recorrido de una fila?
- Volver a definir el procedimiento con direcciones distintas.
- ¿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.

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

---

<sup>1</sup> La fila actual es aquella en la que se encuentra la celda actual.

## 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*<sup>2</sup>. ¿Qué subtareas van a precisarse en este caso? ¿Es necesario volver a definir las 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 esta 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 necesarios para generalizar la noción de recorrido por celdas de un tablero, para que las direcciones de recorrido no estén fijadas. En particular, definir (como siempre, comenzando por los contratos):

- IrAPrimeraCeldaEnUnRecorridoAl\_Y\_(dirPrincipal, dirSecundaria)**
- haySiguieteCeldaEnUnRecorridoAl\_Y\_(dirPrincipal, dirSecundaria)**
- IrASiguieteCeldaEnUnRecorridoAl\_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.

- PintarTableroDe\_(color)** que coloca exactamente una bolita del color dado en cada celda del tablero.
- UnaDeCadaEnTodoElTablero()** que coloca una bolita de cada color en cada celda del tablero.
- 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**.
- 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

---

<sup>2</sup> Recordar que un recorrido considera una secuencia de elementos de a uno por vez. Un *recorrido sobre filas* es uno que considera que los elementos a recorrer son las filas del tablero...

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, si 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 precondition 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.

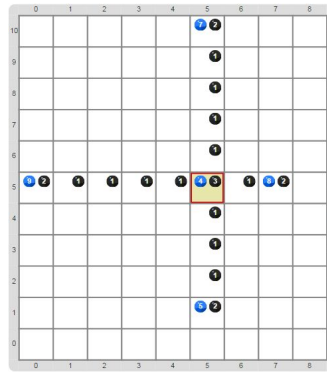
## 13. Distribución de mercadería

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:



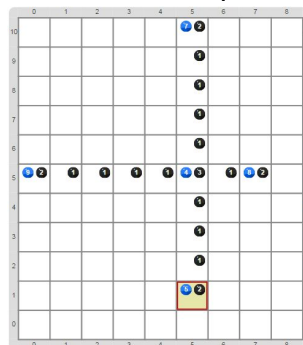


Escribir:

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

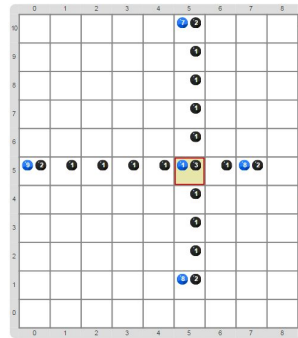
**Aclaración:** si se pide **IrDelLocal\_ACentral(Sur)**, quiere decir que el cabezal está en el depósito Sur, por lo tanto, debe moverse *hacia el Norte*.

Antes de seguir, un ejemplo de uso de estos dos procedimientos. A partir del tablero que se mostró, **IrDeCentralAlLocal\_(Sur)** deja el cabezal en el depósito local Sur, o sea:

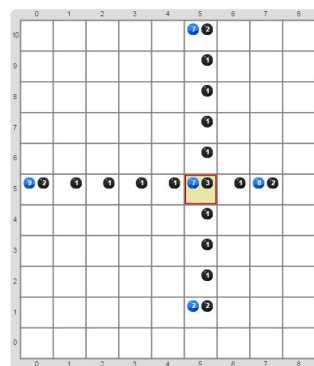


A partir de este tablero, **IrDelLocal\_ACentral(Sur)** "vuelve" al tablero inicial, o sea, el cabezal va a al depósito central.

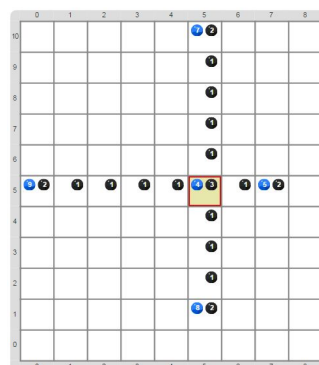
- Llevar\_MercaderíasAlLocal(cantidad, dirección)**, que lleva la cantidad de mercadería indicada del depósito central al depósito local que está en la dirección indicada. Si en el depósito central no hay suficiente cantidad de mercadería, no se hace nada. Se puede suponer que el cabezal está en el depósito central, y debe dejarse en el mismo lugar. Por ejemplo a partir del tablero inicial dado como ejemplo, **Llevar\_MercaderíasAlLocal\_(3, Sur)** tiene este efecto:



- 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. Por ejemplo, a partir del tablero inicial, **Traer\_MercaderíasDelLocal\_(3, Sur)** tiene este efecto:



- 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. Por ejemplo, a partir del tablero inicial, **Mover\_MercaderíasDelLocal\_AlLocal\_(3, Este, Sur)** tiene este efecto:



## 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 miraras 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.
  - i. Si hay que cambiar la dirección (está sobre una celda indicadora), lo hace.
  - ii. 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.
- 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.
- k. **Cambiar\_ParaImitar\_(colorACambiar, colorAImitar)** que cambia la cantidad de bolitas de **colorACambiar** según la cantidad de bolitas de **colorDeReferencia** que haya en la celda actual.
- l. **Mover\_Bolitas\_Al\_(cantidad, color, dirección)** que mueve (es decir, quita de una celda para llevar a la otra) la cantidad indicada de bolitas de color a la celda lindante en la dirección dada, y deja el cabezal en esa celda. Suponer que hay una celda lindante en esa dirección.



UNIVERSIDAD  
NACIONAL DE  
HURLINGHAM

## Introducción a la Programación - Práctica 9

Variables y Funciones con Procesamiento

### 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 Aloí 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.
- Si 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 CON PROCESAMIENTO:

### 1. Mirando la celda vecina

Escribir la función **tieneBolitas\_Al\_**, que, suponiendo que existe una celda lindante en la dirección dada, indica si la misma tiene o no bolitas del color indicado. Si no hay una celda lindante, hace BOOM.

### 2. Mirando la celda vecina, incluso si no hay vecina

**BIBLIOTECA** Escribir la función **hayBolitas\_Al\_**, que indica si hay una celda lindante en la dirección indicada y la misma tiene bolitas del color dado. Si no hay celda lindante describe Falso.

### 3. Mirando en la celda al borde

Escribir la función **hayBolitas\_EnElBorde\_**, que indica si en la celda que se encuentra en el borde dado por la dirección, hay bolitas del color indicado.

### 4. Mirando en la fila o columna

Escribir la función **hayBolitas\_Hacia\_** que indica si en alguna de las celdas hacia la dirección dada (sin incluir la celda actual) hay bolitas del color dado.

### 5. Y volviendo a mirar en la fila o columna

Escribir la función **hayCeldaVacíaHacia\_**, que indica si en alguna de las celdas hacia la dirección dada (sin incluir la celda actual) hay una que esté vacía.

### 6. Y si miramos el tablero

Escribir la función **hayAlgunaBolita\_**, que indica si en alguna de las celdas del tablero existe una bolita del color dado.

### 7. Y volvemos a mirar el tablero

Escribir la función **hayAlgunaCeldaVacía**, que indica si alguna de las celdas del tablero está vacía.

## VARIABLES:

### 8. Copiamos una celda

Escribir el procedimiento **CopiarCeldaAl\_**, que copia los contenidos de la celda actual a la celda lindante en la dirección dada. Note que se deben eliminar los contenidos originales que hubieran en la celda de destino.

### 9. Copiamos las esquinas

Escribir el procedimiento **CopiarOrigenEnEsquinas** que copia en cada esquina los contenidos que hay en la celda actual (las 4 esquinas deben terminar con exactamente las mismas bolitas de cada color que había en la celda donde estaba originalmente el cabezal en el tablero inicial. La posición final del cabezal no es relevante).

### 10. ¡Y Dale!, ¡Y dale!, ¡Y dale Nova dale!!

La revisión de código sigue mal para Nova. Esta vez se trata de unos procedimientos que además de no tener contratos ni buenos nombres, algunos no andan y otros usan las variables de formas inadecuadas. Se pide entonces, encontrar los errores en los procedimientos que escribió Nova y

justificar por qué son errores. Luego, renombrar los procedimientos, variables y parámetros (y sus usos) de manera adecuada, para que el programa resulte legible, y escribir los contratos.

```
procedure P(p)
  { p := p + 1; Poner__Veces(Azul, p) }
procedure Q(p) {
  if(p /= 3) {v := 3}
  Poner__Veces(Azul, v)
}
procedure R()
  { Poner__Veces(Azul, v) Mover__Veces(Este, v) }
procedure S(p) {
  v := Este
  Mover__Veces(v, 5)
  v := 5
  Mover__Veces(Este, v)
}
```

¡Alguien debe enseñarle pronto a Nova lo difícil que es entender y corregir el código si no se escriben buenos nombres y buenos contratos!

## ALTERNATIVA CONDICIONAL EN EXPRESIONES:

### 11. El más chico

**BIBLIOTECA** Escribir la función `mínimoEntre_Y_`, que dados dos valores describe aquel que sea más chico. Por ejemplo, `mínimoEntre_Y_(3, 7)` describe **3**, mientras que `mínimoEntre_Y_(9, 4)` describe **4**.

- ¿De qué tipo son los parámetros?
- ¿Es válida la expresión `mínimoEntre_Y_(Rojo, Azul)`? ¿Qué describe?
- ¿Qué se describe si son iguales? ¿Es relevante si es uno o el otro?

### 12. El más grande

**BIBLIOTECA** Escribir ahora la función `máximoEntre_Y_` que dados dos valores describe aquel que sea el más grande.



## 13. Mi caminante se mueve

La primitiva del ejercicio “**14. El Caminante**” de la **Práctica 8**, **direcciónDelCódigo\_(código)**, puede implementarse con alternativa condicional de expresiones. Se pide que la implemente, y que pruebe ahora su código del caminante para verificar su correcto funcionamiento.

## 14. Piedra, Papel o Tijeras

**EN PAPEL** Escribir **jugadaGanadoraDePiedraPapelOTijerasEntre\_Y\_**, que dadas dos jugadas, describe la jugada ganadora entre ambas. Para olvidarnos de cómo está codificada la jugada, tenemos las funciones **piedra()**, **papel()** y **tijeras()**, que representan a cada una de las jugadas. En piedra papel o tijeras, el jugador puede elegir una de tres opciones, y cada opción pierde contra alguna otra y le gana a alguna otra.

Jugada	Pierde contra	Gana contra
piedra()	papel()	tijeras()
papel()	tijeras()	piedra()
tijeras()	piedra()	papel()

## 15. Piedra, Papel o Tijeras... Lagarto, Spock

**EN PAPEL** La popular variante del juego piedra, papel o tijeras, lagarto, spock, [popularizada por Sheldon Cooper](#), es un juego en esencia idéntico al clásico, pero con mayor número de resultados posibles. El jugador puede elegir entre 5 posibles jugadas, y cada una pierde y/o gana ante dos jugadas, según se muestra en la siguiente tabla:

Jugada	Pierde contra	Gana contra
piedra()	papel(), spock()	tijeras(), lagarto()
papel()	tijeras(), lagarto()	piedra(), spock()
tijeras()	piedra(), spock()	papel(), lagarto()
lagarto()	tijeras(), piedra()	spock(), papel()
spock()	papel(), lagarto()	tijeras(), piedra()

Escribir **jugadaGanadoraDePiedraPapelOTijerasLagartoSpockEntre\_Y\_**, que dadas dos jugadas, describe la jugada ganadora entre ambas. Para olvidarnos de cómo está codificada la jugada, tenemos las funciones **piedra()**, **papel()**, **tijeras()**, **lagarto()** y **spock()** que representan a cada una de las jugadas.

## VARIABLES Y ACUMULACIONES:

### 16. Contando bolitas

Escribir la función **nroBolitas\_EnLaFilaActual** que describa la cantidad de bolitas del color dado en la fila actual.

- Escribir la solución con un recorrido de la fila actual que utilice una variable **cantidadDeBolitasYaVistas** cuyo propósito sea describir la cantidad de bolitas del color correspondiente que se contaron en cada momento. ¿Cuántas se vieron antes de empezar a contar? ¿Cómo estar seguro que se consideraron todas las celdas para contarlas?
- ¿En qué celda queda el cabezal al utilizar la función desde cualquier punto del programa? ¿Por qué?

### 17. Contando celdas hacia un lado

**BIBLIOTECA** Escribir la función **distanciaAlBorde\_**, que describe la cantidad de celdas que hay entre la celda actual y el borde indicado.

**Observación:** si la celda actual se encuentra en el borde, la distancia es 0.

### 18. Mis coordenadas son...

**BIBLIOTECA** Escribir las funciones **coordenadaX** y **coordenadaY** que retornen la coordenada de la columna y la coordenada de la fila de la celda actual, respectivamente. Suponer que 0 es la coordenada de la primera fila y columna.

- ¿Es necesario escribir un recorrido para estas funciones, o puede reutilizarse alguna otra función ya hecha?

### 19. Contando filas y columnas

**BIBLIOTECA** Escribir las funciones **nroFilas** y **nroColumnas** que describan la cantidad de filas y columnas del tablero respectivamente.

- ¿Se podría conocer la cantidad de filas o columnas del tablero sin que el cabezal se mueva *realmente* de la celda actual?
- ¿Qué habría que usar si hubiese que hacerlo exclusivamente con procedimientos?
- ¿Es necesario escribir un recorrido para estas funciones, o puede reutilizarse alguna otra función ya hecha?

### 20. Contando celdas vacías

**BIBLIOTECA** Escribir una función **nroVacías** que describa la cantidad de celdas vacías del tablero. Estructurar el código como recorrido por las celdas del tablero.

### 21. Contando celdas con bolitas.

**BIBLIOTECA** Escribir la función **cantidadDeCeldasConBolitasDeColor\_** que describe la cantidad de celdas que contienen al menos una bolita del color dado.

## 22. Contando bolitas de un color

**BIBLIOTECA** Escribir una función **nroBolitasTotalDeColor\_** que describa la cantidad de bolitas del color dado que hay en total en todo el tablero. Estructurar el código como recorrido por las celdas del tablero.

## 23. Y volvemos a mirar el tablero

**EN PAPEL** Dado que en el tablero está representada una carretera, y en cada celda puede haber hasta un auto, se pide que realice la función **cantidadDeAutosEnLaCarretera**. Para realizar esto se puede hacer uso de las siguientes:

```
function hayUnAuto()  
  
    /*  
        PROPÓSITO: Indica si hay un auto en la celda actual.  
        TIPO: Booleano.  
        PRECONDICIONES: Ninguna.  
    */
```

## 24. El bosque, parte 5

- Escribir la función **cantidadTotalDeÁrbolesEnElTerreno** que describa la cantidad de árboles que hay en el bosque. Organizar el código como un recorrido genérico sobre las parcelas instanciado para el sentido Sur-Oeste.
- Escribir **cantidadTotalDeÁrbolesEnElTerrenoLuegoDeExplosiones**, una función que indica la cantidad total de árboles que quedarán en el terreno luego de explotar todas las bombas que hayan en este.

## 25. Contar se vuelve más fácil

**BIBLIOTECA** Una subtarea de mucha utilidad es aquella que describe 1 cuando se cumple una condición o cero en caso contrario. Se pide entonces escriba la función **unoSi\_ceroSino** que realiza precisamente esto.

## 26. Y volviendo a contar

Reescriba sus recorridos de acumulación anteriores para utilizar la función **unoSi\_CeroSino** en los casos en los que sea posible.

## RECORRIDOS SOBRE ENUMERATIVOS:

## 27. Otra vez una de cada

Volver a escribir el procedimiento **PonerUnaDeCadaColor** que pone una bolita de cada color, estructurando la solución como un recorrido sobre colores.

## 28. Limpiando la cruz

Escribir el procedimiento **LimpiarCruzDeColor\_** que dado un color limpia el dibujo de una cruz realizado con bolitas de dicho color, bajo la suposición de que el cabezal se encuentra en el centro de dicha cruz.

## 29. Hacia la cual hay bolitas

Escribir la función **direcciónHaciaLaCualHayBolitasDe\_** que dado un color describe la dirección hacia la cual hay bolitas de dicho color, bajo la suposición de que hay una única celda con bolitas de dicho color en la celda. La función realizada debe ser total.

## 30. Vecinas con bolitas

Escribir la función **cantidadDeVecinasConBolitas** que describe la cantidad de celdas vecinas que contienen bolitas (de cualquier color). En este caso el concepto de vecindad implica tanto las celdas ortogonales como las diagonales, es decir, las celdas hacia el N, E, S y O y también las diagonales hacia el NE, SE, SO y NO. La función realizada debe ser total.

## 31. Incrementando las cantidades

Escribir el procedimiento **Poner\_EnLineaHacia\_De\_IncrementandoDeA\_ComenzandoEn\_** que dado un número que representa una cantidad de celdas a abarcar, una dirección hacia donde dibujar la línea, un color que indica en color de bolitas a poner, un número que indica el factor de incremento, y un número inicial, pone una línea de bolitas en donde, en la primer celda pone tantas bolitas del color dado como el número inicial, en la celda siguiente hacia la dirección dada, pone tantas bolitas como el número inicial sumado en el factor de incremento, en la dos lugares hacia la dirección tantas como el número inicial sumado en el doble del factor del incremento, y así siguiendo tantos lugares como el primer argumento. Ej. sí se invoca al procedimiento de la siguiente forma **Poner\_EnLineaHacia\_De\_IncrementandoDeA\_ComenzandoEn\_(5, Norte, Rojo, 3, 3)** entonces se dibujará una línea de 5 celdas hacia el norte de bolitas de color rojo, comenzando en la celda actual, en donde se tendrán en cada celda (contando de la actual) las siguientes cantidades de bolitas: 3, 6, 9, 12, 15.

## 32. El número de Fibonacci

Escribir la función **fibonacciNro\_**, que dado un número que representa una posición en la secuencia de fibonacci (debe ser mayor o igual a cero) describe el número de fibonacci correspondiente a dicha posición.

La sucesión de fibonacci es una sucesión infinita de número en donde se comienza con el número 1 como elemento en la primera y segunda posición de la sucesión, y luego, cada elemento de la sucesión se calcula como la suma de los dos elementos anteriores. A continuación se deja una pequeña tabla de la sucesión de fibonacci para los primeros números:

Posición	0	1	2	3	4	5	6	7	8	9	10
Elemento	1	1	2	3	5	8	13	21	34	55	89
Cálculo	-	-	1+1	2+1	3+2	5+3	8+5	13+8	21+13	34+21	55+34

## RECORRIDOS DE MÁXIMO Y MÍNIMO:

### 33. La celda con más

Escribir las funciones **coordenadaXConMásBolitas** y **coordenadaYConMásBolitas** que describen las coordenadas X e Y de aquella celda que tiene más bolitas (en total) que el resto. Se garantiza por precondition que hay alguna celda que tiene más bolitas que el resto.

### 34. El borde más cercano

Escribir la función **bordeMásCercano** que describe la dirección hacia la cual se encuentra el borde que está más cerca (a menor cantidad de celdas). Si hubiera dos bordes a la misma distancia describe la dirección más chica entre ellas.

### 35. Color más chico del cual hay bolitas

Escribir la función **colorMásChicoDelCualHayBolitas** que describe el color más chico para el cual haya bolitas en la celda actual. Por ej. si en la celda hay bolitas Negras, Rojas y Verdes, el color más chico del cual hay bolitas es Negro. Si solo hay bolitas de color Rojo y Verde, el más chico es Rojo.

- ¿Qué pasa si no hay bolitas en la celda actual?
- ¿Qué tipo de recorrido se está aplicando?

### 36. Color más grande del cual hay bolitas

Escribir la función **colorMásChicoDelCualHayBolitas** que describe el color más grande del cual hay bolitas. Por ej. si en la celda hay bolitas Negras, Rojas y Verdes, el color más grande del cual hay bolitas es Verde. Si solo hay bolitas de color Rojo y Negro, el más grande es Rojo.

## EJERCICIOS INTEGRADORES:

### 37. Nova y sus variables misteriosas

¡Alguien tendría que sentarse con Nova y darle un par de lecciones! En su código aparecieron dos funciones muy mal indentadas, y donde algunos parámetros y variables tienen nombres que no ayudan a entender su propósito. ¡Y es tan difícil entender para qué puso una variable si no la nombra adecuadamente!

```
function total(x) {  
    // nro bolitas x en el tablero  
    IrAlInicioDeUnRecorrido__((Sur,  
    Oeste)  
    var := 0  
    while(not estoyElFinalDeUnRecorrido__((Norte, Este))  
    { var := var + nroBolitas(x)  
    PasarASiguienteCelda__(( Norte, Este) }  
    return(var + nroBolitas(x))  
}
```

```
function bolitasDeColor(n) {
    // nro máximo bolitas n en una celda
    IrAlInicioDeUnRecorrido__(Sur, Oeste)
    var := nroBolitas(n)
    PasarASiguienteCelda__(Norte, Este)
    while(not estoyElFinalDeUnRecorrido__(Norte, Este))
    { var := máximoEntre_Y_(var, nroBolitas(n))
      PasarASiguienteCelda__(Norte, Este)}
    return(máximoEntre_Y_(var, nroBolitas(n)))
}
```

- a. Asociar las siguientes oraciones A y B que describen propósitos de variables con sus respectivas variables en el código de Nova.
  - A. Denota la cantidad de bolitas del color dado que ya se contaron.
  - B. Denota la mayor cantidad de bolitas del color dado que tenía alguna de las celdas recorridas.
- b. Una vez asociado, renombrar las variables y parámetros de forma adecuada y reescribir la indentación para que se entiendan de manera adecuada.

## 38. Posicionandose en una celda vacía puntual

Escribir un procedimiento **IrAVacíaNúmero\_(númeroDeVacía)** que posicione el cabezal en la celda vacía número **númeroDeVacía** que se encuentra en un recorrido del tablero por celdas en las direcciones Este y Norte. Si no hay suficientes celdas vacías, deja el cabezal en la esquina NorEste. Por ejemplo, **IrAVacíaNúmero\_(1)** posiciona el cabezal en la primer celda vacía, **IrAVacíaNúmero\_(2)** posiciona el cabezal en la segunda celda vacía, etc. Organizar la solución como un recorrido de búsqueda por celdas que utilice una variable **celdasVacíasYaVistas**, cuyo propósito sea denotar la cantidad de celdas vacías que ya se recorrieron.

## 39. La torre de control

En este ejercicio las columnas del tablero representan cada una una pista de aterrizaje en la que despegan y aterrizan aviones. Se considera que una pista está libre para aterrizar si no hay avión en ninguna de las posiciones de esa pista. Los aviones se representan con tantas bolitas azules como el número de su vuelo, y con una bolita Roja en la misma celda si está aterrizando o con una bolita Verde si está despegando. El tablero representa a todo el aeropuerto. Además sabemos que en cada celda hay a lo sumo un avión, y que en una misma pista pueden haber varios aviones.

- a. La función **cantidadDePistasLibres**, que devuelva la cantidad de pistas de aterrizaje sin aviones en ella.
- b. La función **cantidadDeAvionesDespegando**, que devuelva la cantidad total de aviones que están despegando en todo el aeropuerto.

- c. El procedimiento **IrAPistaLibreParaDespegar**, que deje el cabezal en la primera pista libre más cerca al Este del aeropuerto, en la posición más al Sur de la pista.
- d. La función **cantidadDeAvionesTotales**, que devuelva la cantidad total de aviones que están despegando o aterrizando en todo el aeropuerto.
- e. El procedimiento **IrAPistaLibreParaAterrizar**, que debe dejar al cabezal en la primera pista libre más al Oeste del aeropuerto, en la celda más al Norte de la misma.
- f. La función **cantidadDePistasConColisiónInminente**, que devuelva la cantidad de pistas con posibles colisiones en todo el aeropuerto. Se considera que hay una colisión posible si en la misma pista hay un avión que está despegando debajo (más al Sur) de un avión que está aterrizando.
- g. El procedimiento **IrAPistaConColisiónInminente**, que deje el cabezal en la primera pista con una colisión inminente contando desde el Oeste, y en la celda más al Sur de la pista.



UNIVERSIDAD  
NACIONAL DE  
HURLINGHAM

## Introducción a la Programación - Práctica 11

### Tipos de Datos Personalizados

#### 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 Aloí 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.**



## VARIANTES:

### 1. Días de la semana

Declarar un tipo variante llamado **DíaDeLaSemana**, que sirva para identificar los días de la semana. Luego implementar las siguientes funciones (sin olvidar sus contratos):

- díaSiguiente\_** que dado un día de la semana, devuelve el día siguiente.
- díaPrevio\_** que dado un día de la semana, devuelve el día previo.
- esDíaDeFinDeSemana\_**, que indica si el día dado es uno del fin de semana.

**Reflexionamos:** ¿De qué tipo es el parámetro dado a cada una de estas funciones? ¿Cuál sería un buen nombre para el mismo? ¿Por qué las funciones siguiente y previo no pueden utilizarse para solucionar este problema? ¿Qué beneficios trae tener ahora díaSiguiente y díaPrevio si necesito saber algo cómo "en qué día ocurrió un suceso"?

### 2. Partidos políticos

**EN PAPEL** Se quiere modelar una serie de partidos políticos de un país, y poder realizar consultas sobre los mismos. Se cuenta entonces con la siguiente tabla de información:

Partido
Democracia por la Verdad
Unidos por la República
Liberales por la Libertad
Izquierda de los Obreros

Se quiere entonces crear un tipo de datos variante **PartidoPolítico** que modele a los partidos mencionados. Además se les brinda la siguiente función como primitiva:

```
function cantidadDeVotosDe_(unPartido)
/*
  PROPÓSITO: Indica la cantidad de votos que recibió un partido.
  PARÁMETROS
    * unPartido: PartidoPolítico - El partido político del cual saber
    su cantidad de votos.
  TIPO: Número
  PRECONDICIÓN: Ninguna
*/
```

Luego escriba las siguientes funciones:

- tieneMásVotantes\_Que\_** que dados dos partidos indique si el primero tiene más votos que el segundo.
- elQueTieneMásVotos** que describe el partido con más votantes.
- habráBallotage** que indica si en estas elecciones habrá ballotage. Esto se da cuando el partido con más votos no acumula más del 50% de los votos totales y no hay una diferencia de más del 10% sobre entre el primero y el segundo candidato.

**REGISTROS:**

## 3. Jugamos con cartas

Declare el tipo variante **Palo** y el tipo registro **Carta** y escriba las siguientes funciones. No se olvide de ir probando en Gobstones cada función que realiza para saber si el resultado es correcto.

- a. **anchoDeEspadas** que describe la carta 1 de Espadas.
- b. **anchoDeBastos** que describe la carta 1 de Bastos
- c. **laCarta\_de\_** que dado un número y un palo que describe la carta con dicho número y dicho palo..
- d. **esUnAncho\_** que indica si la carta dada es un 1.
- e. **esFigura\_** que dada una carta, indica si la misma es una figura (las figuras son los 10s, los 11s y los 12s).
- f. **esDeOro\_** que indica si la carta dada es de Oros.
- g. **tieneUnNúmeroMásGrande\_Que\_** que dadas dos cartas indica si la primera carta tiene un número más grande que la segunda.
- h. **sonDelMismoPalo\_Y\_** que dadas dos cartas, indica si estas tienen el mismo palo.
- i. **valorParaEnvioDe\_** que describa el número que aporta la carta dada en el canto del envío. El número se corresponde al número de la carta, si la misma no es figura, y cero, si fuera figura.  
**Atención:** Si ya sabe jugar al truco, tenga en cuenta que se está preguntando el valor que aportaría una única carta, no una jugada de multiples cartas.
- j. **mayorValorEntre\_Y\_** que describe el valor más grande entre dos cartas, según lo que aporta cada una para el envío. Por ejemplo, si las cartas son un 7 y un 12, describe 7, pues el 12 no aporta nada para el envío.
- k. **sumaParaElEnvioCon\_Y\_** que dadas dos cartas, describe el número que tienen las mismas para el envío. El envío se calcula como la suma los valores del envío de cada carta más 20, si las cartas son del mismo palo, o como el mayor valor entre ellas, cuando son de distinto palo.
- l. **sonMejores\_Y\_Que\_Y\_** que dadas 4 cartas, donde las dos primeras son las cartas del primer jugador para cantar envío, y las segundas dos las del segundo jugador, indica si el envío del primer jugador es más grande que el envío del segundo jugador.

## 4. Las viejas y queridas celdas, ahora con sabor a datos

Dado el siguiente tipo de registro:

```
type Celda is record {  
    /*  
        PROPÓSITO: Modelar una celda del tablero  
        INV.REP.: Los números son todos >=0  
    */  
    field cantidadDeAzules // tipo: Número  
    field cantidadDeNegras // tipo: Número  
    field cantidadDeRojas // tipo: Número  
    field cantidadDeVerdes // tipo: Número  
}
```

Se pide que realice las siguientes funciones y procedimientos:

- a. **celdaActual** que lee la información de la celda en donde está el cabezal y retorna un dato de tipo Celda.

- b. **EscribirEnCelda\_** que dado el dato de una calda, escribe la información de este en el tablero en la celda actual.
- c. **tienenMismaCantidadDeRojas\_Y\_** que dados dos datos del tipo celda, indica si efectivamente ambos tienen la misma cantidad de bolitas rojas.

## 5. Mis primeras personas

- a. Declarar un tipo de registros llamado **Persona**, que contenga el número de DNI y el domicilio representados mediante **Strings**, y un **Booleano** indicando si la persona es donante de órganos. Implementar las siguientes funciones:
- b. **sonConvivientes\_Y\_**, que dadas dos personas indique si comparten domicilio.
- c. **personaNacidaDe\_** que, dada una persona madre, describe a una nueva persona que haya nacido de la misma, y por lo tanto conviva con la madre, no tenga DNI asignado y en principio se indica que no es donante de órganos.  
Para registrar el DNI sin asignar, escribir la función **sinAsignar**, que denote el string vacío y utilizarla adecuadamente.
- d. **persona\_RegistradaCon\_**, que dada una persona con DNI sin asignar y un DNI de registro, describe a la persona pero con el DNI asignado al dado.
- e. **persona\_ConDomicilioNuevoEn\_**, que recibe una persona y un nuevo domicilio y describe a la persona con el domicilio cambiado.
- f. **persona\_ConSituaciónComoDonanteCambiada**, que recibe una persona y retorna la persona con su situación como donante cambiada.

**Reflexionamos:** Considere las siguientes preguntas:

- ¿Es conveniente representar el DNI y el domicilio como Strings?
- ¿Puedo validar de alguna forma si el DNI dado es válido?
- ¿Y para el domicilio?
- ¿Cómo puedo lidiar con distintos formatos de escribir un domicilio?
- ¿Hay forma de distinguir las partes de una dirección (calle, altura, etc.)?
- ¿Y si representáramos el nombre de una persona? ¿Conviene utilizar un String?
- ¿Qué diferencia hay entre la dirección y el nombre de una persona que hace que la primera pueda ser inconveniente para representar con Strings?
- Aquí representamos la situación de un donante como un booleano, ¿Hay situaciones donde esto sea inconveniente?
- En muchas aplicaciones que modelan personas es común incluir un campo para identificar el "sexo" de las personas, típicamente como "varón" o "mujer", o quizás como "masculino" o "femenino". Sin embargo, en la actualidad la condición binaria de este concepto está en discusión, así como la necesidad de que al modelar personas sea necesario identificar esta característica, ya que existen personas que no se identifican necesariamente en forma binaria con lo que históricamente se entendió por "sexo". Discutir en clase con los docentes y sus compañeros sobre cómo afecta la vida de las personas las decisiones sobre modelado de datos que se toman al escribir programas. Discutir también sobre cómo habría que modelar esta información en caso de considerarlo necesario. Como guía, se puede utilizar este video llamado "¿Qué es la diversidad sexual?" publicado por la Colectiva Antipatriarcal Floreciendo en Fuga:

<https://www.youtube.com/watch?v=OqQm8nvEhUw>.

## 6. Mis primeras personas ahora van a laburar

Declarar un tipo de registros llamado **Empleado** que contenga la identidad del empleado modelada con el registro **Persona**, el puesto dentro de la empresa, representado por un tipo variante llamado **Puesto** dado a continuación, y un sueldo modelado como un **Número** en pesos (sin centavos).

```

type Puesto is variant {
  /*
    PROPÓSITO: Modelar los diferentes puestos que hay
               dentro de una empresa de software
  */
  case GestorDeProyecto {}
  case LiderDeProyecto {}
  case Desarrollador {}
  case Operador {}
}

```

Implementar las siguientes funciones:

- empleado\_ConSueldoActualizadoA\_**, que dado un empleado y un nuevo sueldo, describa al empleado con el sueldo actualizado.
- empleado\_ConNuevoPuesto\_**, que dado un empleado y un nuevo puesto, describa al empleado con el puesto actualizado.
- categoríaDelPuesto\_**, que dado un valor de tipo puesto devuelve su categoría según la siguiente tabla:

<b>GestorDeProyecto</b>	4
<b>LíderDeProyecto</b>	3
<b>Desarrollador</b>	2
<b>Operador</b>	1

- empleadoConMayorCargoEntre\_Y\_**, que dados dos empleados describa el empleado de mayor categoría entre ellos.
- tienenIgualSueldo\_Y\_**, que dados dos empleados indique si ambas cobran lo mismo.
- empleado\_ConAumentoEn\_PorBonoDeFinalizaciónDeProyecto**, que dado un empleado y un porcentaje de aumento describa al empleado con el sueldo actualizado en ese porcentaje. El porcentaje de aumento es un número del 0 al 100. El monto en el que se incrementa depende del porcentaje (i.e. si el porcentaje es 20%, cuando el sueldo es 100, el nuevo sueldo es 120, cuando el sueldo es 200 el nuevo sueldo es 240, y cuando el sueldo es 150, el nuevo sueldo es 180).
- empleado\_ConDomicilioActualizadoA\_** que dado un empleado y un nuevo domicilio, describe el empleado en donde la identidad se ha actualizado para contener el nuevo domicilio.

## 7. Fechas, ahora en sabor datos

Declarar los tipos necesarios para representar **Fechas** (según el calendario gregoriano utilizado para identificar nuestras fechas, componiéndolas de una **día** (representado mediante un **Número**, un **mes** (representado mediante un tipo **variante Mes** que modela los meses, y un **año** representado mediante un **Número**).

- ¿Cuáles deben ser las invariantes de representación?  
**AYUDA:** separar los meses en 3 grupos, y para Febrero considerar los años que sean bisiestos y no lo sean.

- b. Escribir funciones para describir las fechas dadas en el ejercicio 5 de la práctica 2 y en el ejercicio 12 de la práctica 4 como valores del tipo **Fecha**.

## 8. Escribir fechas ahora requiere menos argumentos

Escribir el procedimiento **EscribirFecha\_**, que dada una fecha, la represente con bolitas siguiendo la representación utilizada en las prácticas 2 y 4.

**AYUDA:** considerar reutilizar el procedimiento **EscribirFechaConDía\_Mes\_Año\_**<sup>1</sup> definido en el ejercicio 12 de la práctica 4.

## 9. Y ahora puedo calcular cosas sobre las fechas

Escribir las siguientes funciones sobre fechas:

- a. **esDíaDeÑois\_**, que indica si el día dado es un 29.
- b. **esDelPrimerSemestre\_**, que indica si el día determinado por la fecha dada es uno del primer semestre (entre el 1ro de enero y el 31 de julio).  
**AYUDA:** Considere implementar una función auxiliar **númeroDelMes\_** que describa el número de un mes dado, y otra que le permita determinar si un mes es anterior a otro.
- c. **esFechaDeAñoBisiesto\_**, que dada una fecha, indica si la misma cae dentro de un año bisiesto.  
**AYUDA:** Para saber si un año es bisiesto, hay que verificar que el año sea múltiplo de 4, pero no de 100 a menos que sea múltiplo de 400. Por ejemplo, 1796, 1896 y 1996 son bisiestos (son múltiplos de 4 y no de 100), pero 1800 y 1900 NO lo son (son múltiplos de 4 y de 100, pero no de 400); en cambio 2000 ES bisiesto (es múltiplo de 4, de 100 y de 400).
- d. **primerDíaDelInviernoDe\_**, que describa el primer día del invierno del año dado (en el hemisferio sur).
- e. **últimoDíaDelInviernoDe\_**, que describa el último día del invierno del año dado (en el hemisferio sur).
- f. **esMásAntigua\_Que\_**, que dadas dos fechas indique si la primera es anterior a la segunda.  
**AYUDA:** Considere primero comparar por año, si no se pudiera determinar en base a este pues son iguales, considerar los meses, y si con los meses no se pudiera determinar, considerar el día.
- g. **esInviernoEl\_**, que dada una fecha indica si el día dado es uno del invierno.  
**AYUDA:** considerar utilizar las funciones de los 3 puntos anteriores.
- h. Escriba también **esVeranoEl\_**, **esPrimaveraEl\_** y **esOtoñoEl\_**, siguiendo la misma estrategia que la utilizada para **esInviernoEl\_**.
- i. **estaciónDe\_**, que dada una fecha retorna la estación del año en la que ocurre la fecha. Para ello, definir un tipo variante **Estación** que modela las estaciones del año.
- j. **cuántosAñosPasaronEntre\_Y\_**, que dadas dos fechas describa el número de años que hay entre las dos fechas (no tiene en cuenta períodos de menos de 1 año – o sea, si la distancia es menor a 1 año, describe 0).
- k. **siguienteFecha\_**, que dada una fecha, devuelve la fecha del día siguiente.  
**AYUDA:** usar funciones auxiliares **siguienteDíaParaDía\_YMes\_** y **siguienteMesParaDía\_YMes\_**.
- l. **previoFecha\_**, que dada una fecha, devuelve la fecha del día anterior.  
**AYUDA:** usar funciones auxiliares **previoDíaParaDía\_YMes\_** y **previoMesParaDía\_YMes\_**.

<sup>1</sup> Observar que tienen diferente cantidad de parámetros.

- m. **Escribir**`CalendarioDeAño_` que escriba en el tablero en forma vertical todas las fechas del año dado, usando la representación del ejercicio anterior, con la primera más al Norte del tablero, y la última más al Sur. No se olvide sus precondiciones.
- n. **EN PAPEL** Dada la siguiente función primitiva:

```
function hayCumpleañosEn_(unaFecha)
/*
    PROPÓSITO: Indica si en la fecha dada hay un cumpleaños de alguno
                de los creadores de Gobstones
    PARÁMETROS:
        * unaFecha : Fecha - La fecha de la cual saber si hay un cumpleaños
    TIPO: Booleano
    PRECONDICIÓN: Ninguna
*/
```

Escriba las funciones `primerCumpleañosDelAño_` y `últimoCumpleañosDelAño_` que dado un año describen la fecha en la que se da el primero de los cumpleaños del año, y la última, respectivamente.