Práctica 7
Introducción a Registros

Introducción a la Programación
1er Semestre de 2015

Los ejercicios que corresponden a los **contenidos mínimos** recomendados se encuentran marcados con el simbolo ⊛.

1. Manipulación básica de registros

Ejercicio 1

Declarar el registro `Persona`, que contenga el número de DNI, el domicilio (un número) y un booleano indicando si la persona es de sexo masculino. Implementar las siguientes funciones:

a. `convivientes` que dadas dos personas indique si comparten domicilio.

b. `nacimiento` que, dada una persona `madre` y un booleano indicando el sexo, cree una nueva persona que conviva con la `madre`, sin DNI asignado y el sexo indicado. Escriba la función `sinAsignar` que denote el valor cero y utilícela de modo que `dni <- sinAsignar()`.

c. `registrarNacimiento` que dada una persona y un DNI, retorna la persona con el DNI. ***Precondición:*** *la persona no debe tener un DNI asignado.*

d. `cambioDomicilio` que recibe una persona y un nuevo domicilio y denota la persona con el domicilio cambiado.

e. `cambioSexo` que recibe una persona y retorna la persona con el sexo cambiado.

Ejercicio 2

Declarar el registro `Alumno`, que contenga los datos básicos del alumno (modelado con el registro `Persona`), su número de legajo, la cantidad de materias que rindió y la cantidad que aprobó. Implementar las siguientes funciones:

a. `alumnoIngresante`, recibe una persona y un número de legajo y crea un alumno que no curso ninguna materia.

b. `materiasDesaprobadas`, que retorna el número de materias desaprobadas por un alumno.

c. `cambioLegajo`, que dado un alumno y un número de legajo retorna un nuevo alumno que resulta de aquel pasado como argumento modificando su legajo.

d. `cambioDomicilioAlumno`, recibe un alumno y una nuevo domicilio y retorna un nuevo alumno resultante de aquel pasado como argumento pero con el nuevo domicilio.

Ejercicio 3

Declarar el registro `Empleado` que contenga el nombre (representado por un número), el DNI, el puesto dentro de la empresa (representado por un número según indica la tabla) y un sueldo. Implementar las siguientes funciones:

Puesto	Codificación
Gestor de Proyecto	4
Lider de Proyecto	3
Desarrollador	2
Operador	1

a. `gestorDeProyecto`, `liderDeProyecto`, `desarrollador` y `operador` que denoten la codificación para cada puesto.

b. `actualizarSueldo` que dado un empleado y nuevo sueldo, cree un nuevo empleado con dicho sueldo.

c. `recategorizar` que dado un empleado y un nuevo puesto, cree un nuevo empleado con ese puesto.

d. `tieneMayorCargo` que dado dos empleados indique cual es el empleado de mayor categoría.

e. `tienenIgualsueldo` que dado dos personas indique si ambas cobran lo mismo.

f. `bonoPorFinalizacionDeProyecto` que dado un empleado y un porcentaje de aumento calcule el nuevo valor a cobrar. El porcentaje de aumento es un número del 0 al 100. El monto en el que se incrementa el sueldo del empleado está dado por $sueldo \times porcentajeDeAumento \div 100$.

Ejercicio 4

Declarar el registro `Cuenta` que contenga el numero de cuenta, el tipo (un numero según la tabla de codificaciones), nombre del cliente (representado por un número), un tipo de moneda (representado por un color según la tabla) y un saldo. Implementar las siguientes funciones:

Tipo de cuenta	Codificación
Cuenta corriente	1
Caja de ahorro	2
Cuenta sueldo	3

Moneda	Codificación
Peso	Negro
Dólar	Verde
Euro	Azul

a. `actualizarSaldo` que dada una cuenta y un nuevo saldo cree una nueva cuenta el nuevo saldo.

b. `cambioDeTipo` que dada una cuenta y un nuevo tipo de cuenta bancaria crea una nueva cuenta con dicho tipo.

c. Por disposición bancaria todas las cuentas bancarias de tipo Sueldo cuyo saldo supera los $25.000 deben cambiar su tipo de cuenta bancaria a Caja de Ahorro. Implemente la operación.

d. `cambioDeTipoAutomatico` que dada una cuenta bancaria retorna una nueva cuenta con nuevo tipo si es necesario.

e. `saldoEnPesos` que dada una cuenta retorna el saldo en pesos. Asuma los siguientes valores cambiarios: un dólar equivale a ocho pesos, un euro equivale a once pesos.

f. `mayorSaldo` que dadas dos cuentas bancarias retorna el número de cuenta de la cuenta bancaria que tiene mayor saldo.

g. `extraccionDeCajero` que dada una cuenta y un monto a extraer actualice el saldo de la cuenta. ¿Es necesaria una precondición?

h. `puedeExtraerDeCajero` que dada una cuenta y un monto a extraer indique si la extracción resultará en un saldo negativo.

i. `depositoEnCajero` que dada una cuenta y un monto a depositar actualice el saldo de la cuenta.

2. Representación del tablero mediante registros

Ejercicio 5

Declarar el registro `Coord` que representa una coordenada del tablero con los campos `fila` y `columna`, donde la coordenada del origen es `Coord(fila ← 0, columna ← 0)`. Implementar las siguientes operaciones:

a. `coord` que dado los números `x` e `y` retorne una nueva coordenada con `fila <- x` y `columna <- y`.

b. `coordOrigen` que denote la coordenada del origen, i.e., `Coord(fila <- 0, columna <- 0)`.

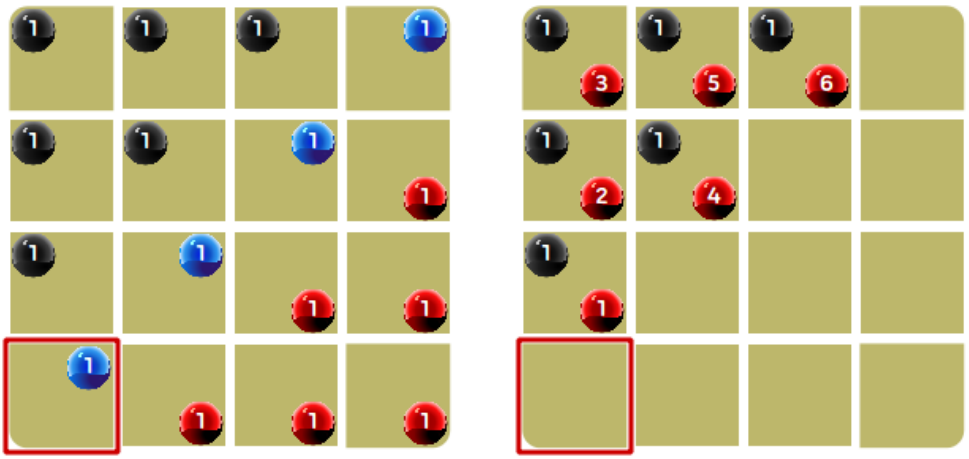
c. `esMayorCoord` que, dadas dos coordenadas, determina si la primer coordenada aparece después de la segunda en un recorrido que va primero hacia el norte y luego hacia el este. Es decir, `esMayorCoord(c1, c2)` denota `True` cuando `columna(c1) > columna(c2)` o cuando `columna(c1) == columna(c2)` pero `fila(c1) > fila(c2)`.

d. `coordActual` que retorne la coordenada en la que se encuentra el cabezal.

e. `ultimaCoord` que denote la ultima coordenada del tablero en un recorrido hacia el noreste.

f. `IrACoord` que, dada una coordenada, posicione el cabezal en la celda correspondiente del tablero.

g. `siguienteCoord` que, dada una coordenada, denote la siguiente coordenada en un recorrido que va primero hacia el norte y luego hacia el este.



(a) Las celdas con bolitas azules se encuentran en la diagonal del tablero, las celdas con bolitas negras se encuentran sobre la diagonal y las celdas con bolitas rojas se encuentran debajo la diagonal.

(b) La cantidad de bolitas rojas en las celdas indica la progresión de un recorrido (Norte, Este) sobre la diagonal.

Figura 1

h. `reflejo` que, dada una coordenada, denote la coordenada que se obtiene de intercambiar las filas y la columna. Es decir, `reflejo(c)` es la coordenada `Coord(fila ← columna(c), columna ← fila(c))`.

i. `sobreLaDiagonal` que, dada una coordenada, indique si la misma se encuentra sobre la diagonal que va del origen hasta la ultima fila. En otras palabras, denota `True` cuando la fila de la coordenada es superior a la columna (ver figura 1a).

j. `siguienteCoordSobreLaDiagonal` que, dada una coordenada, denote la siguiente coordenada que se encuentre sobre la diagonal. Por ejemplo, en un tablero de 8×8,
`siguienteCoordSobreLaDiagonal(Coord(fila ← 7, columna ← 3))`
denota `Coord(fila ← 5, columna ← 4)`. La figura 1b muestra un posible recorrido sobre la diagonal. ¿Cuál es la precondición de la función?

k. `ultimaCoordSobreLaDiagonal` que denota la última coordenada sobre la diagonal. Por ejemplo, en un tablero de 8×8, denota `Coord(fila ← 7, columna ← 6)`. Como precondición, debe existir al menos una celda sobre la diagonal o, en otras palabras, el tablero debe tener al menos dos filas.

l. `triangularInferior` que denota `True` cuando las celdas sobre la diagonal están vacias (i.e., no tienen bolitas).
Sugerencia: *estructurar la solución usando un recorrido de coordenadas sobre la diagonal.*

Ejercicio 6

⊛ Declarar el registro `Celda` con campos `azul, negro, rojo` y `verde` que represente una celda del tablero contando la cantidad de bolitas de cada color que hay en la celda. ¿Es válido

el valor `Celda(azul ← 1, negro ← 3, rojo ← 7, verde ← -3)`? Escribir el invariante de representación, e implementar las siguientes operaciones

a. `celdaActual` que denote la registro correspondiente a la celda actual.

b. `celdaEnCoord` que, dada una coordenada (modelada con un registro `Coord`), denote el registro correspondiente a la celda en dicha coordenada.

c. `VolcarCelda` que, dado un registro celda, deje tantas bolitas de cada color como indica el registro.

d. `IntercambiarCeldas` que, dadas dos coordenadas, intercambie las bolitas de las celdas en dichas coordenadas.

e. `ReflejarCelda` que, dada una coordenada, intercambie el contenido de la celda actual con aquella que se encuentra en su coordenada reflejo.

f. `ReflejarTablero` que refleje todas las celdas de un tablero cuadrado (i.e., misma cantidad de filas que columnas). Para ello, conviene hacer un recorrido del tablero en que se reflejen únicamente las celdas correspondientes a posiciones sobre la diagonal.

g. `simetrico` que indique si cada celda sobre la diagonal contienen la misma cantidad de bolitas que la celda en su coordenada reflejo.

3. Ejercicios adicionales

Ejercicio 7

Se desea crear una librería para la creación de videojuegos en XGobstones. Un videojuego consiste de un conjunto de entidades que interactuan entre sí: un personaje controlado por el jugador, enemigos que buscan atacar a dicho personaje, objetos que aparecen en el nivel que pueden ser consumibles por el personaje del jugador, etc.

Declare el registro `Entidad` que contenga los campos `clase`, `vida`, `ataque` y `contador`. El campo `clase` denota un número que representa la clase de entidad ante la cual nos encontramos (enemigo, jugador, proyectil, etc.). El campo `vida` indica los puntos de vida de la entidad. El campo `ataque` denota los puntos de ataque de la entidad (el daño que produce al atacar a otra entidad). El campo `contador` representa un acumulador que puede ser utilizado para diversos fines (contar los puntos del jugador, contar la cantidad de balas restantes, etc.).

a. Definir la función `actualizarVida` que recibe una entidad `e` y un número `n` y actualiza los puntos de vida de la entidad `e` sumándole `n` puntos de vida.

b. Definir la función `actualizarContador` que recibe una entidad `e` y un número `n` y actualiza el campo `contador` de la entidad `e` sumándole `n`.

c. Definir la función `atacar` que recibe una entidad `atacante` y una entidad `atacado` y devuelve la entidad `atacado` donde sus puntos de vida se reducen en la cantidad que indica el campo `ataque` de `atacante`.

d. Escribir `PonerEntidad`, análogo a `Poner`, que recibe una entidad `e` y la coloca en la celda actual. Para ello utilice la siguiente representación: la cantidad de bolitas azules denotará la clase de la entidad, la cantidad de bolitas verdes denotará la cantidad de puntos de vida de la entidad, la cantidad de bolitas rojas denotará los puntos de ataque de la entidad y la cantidad de bolitas negras denota el valor del contador asociado a la entidad.

e. Escribir `SacarEntidad`, análogo a `Sacar`, que saca la entidad de la celda actual.
Precondición: *debe haber una entidad en la celda actual..*

f. Escribir `MoverEntidad`, análogo a `Mover`, que dada una dirección `dir`, mueve la entidad de la celda actual a la celda lindante en dicha dirección.
Precondición: *Debe haber una celda en dirección* `dir` *y la misma debe estar vacía..*

g. Escribir `hayEntidad` que retorna `True` si hay una entidad en la celda actual y `False` en caso contrario.

h. Escribir `LeerEntidad` que retorna la entidad presente en la celda actual.

i. Declarar una función `esEntidad` que recibe una clase de entidad `c` y verifica si la entidad presente en la celda actual es de la clase `c`.

j. Declarar la función `igualClase` que dado una entidad `e` y una entidad `f` retorna `True` si ambas entidades son de la misma clase, `False` caso contrario.

k. Declarar el procedimiento `SiguienteEntidad` que dado una clase de entidad `c`, una dirección interna `dirInt` y una dirección externa `dirExt` coloca el cabezal en la siguiente entidad de clase `c` que encuentra en un recorrido que va de `opuesto(dirInt)` a `dirInt` y de `opuesto(dirExt)` a `dirExt`.

l. Declarar la función `haySiguienteEntidad` que dado una clase de entidad `c`, una dirección interna `dirInt` y una dirección externa `dirExt` retorna `True` si hay una entidad de clase `c` en un recorrido que va de `opuesto(dirInt)` a `dirInt` y de `opuesto(dirExt)` a `dirExt`, retornando `False` en caso contrario.

Ejercicio 8

Utilizando la librería que definimos en el ejercicio anterior implementaremos un videojuego al que llamaremos "El pequeño juego de Sbog". Este juego consiste en un pequeño personaje llamado Sbog que recorre el nivel buscando recoger todas las monedas y evitando pisar las bombas del mapa.

a. Definir las funciones `sbog`, `moneda` y `bomba` que retornan `1`, `2` y `3` respectivamente, representando las clases de entidad del juego.

b. Definir el procedimiento `MoverSbog` que dado una dirección `dir` mueve a Sbog una celda en dicha dirección si es posible. Además, si la celda tiene una moneda, Sbog la toma sumando la cantidad de monedas que lleva, si la celda tiene una bomba, Sbog pierde tantos puntos de vida como puntos de ataque de la bomba.
Precondición: *El cabezal se encuentra sobre Sbog.*

Sugerencia: *implemente los procedimientos* `EvaluarRecolectar` *que dada una dirección* `dir` *incremente la cantidad de monedas de Sbog si hay una moneda en dirección* `dir` *y* `EvaluarAtaque` *que dada una dirección* `dir` *disminuya los puntos de vida de Sbog de acuerdo a los puntos de ataque de la bomba en dirección* `dir` *si hay una bomba en esa dirección.*

c. Definir la función `sbogVivo` que indica si Sbog sigue vivo.

d. Definir la función `quedanMonedas` que indica si quedan monedas en el mapa.

e. Definir la función `finDeJuego` que indica si el juego terminó. Recordar que el juego termina si no quedan más monedas en el mapa o si Sbog muere.

f. Escribir el procedimiento `CargarMapa` que agrega tres entidades de clase bombas con 25 puntos de ataque, cinco entidades de clase moneda y una entidad de clase Sbog al mapa, dejando el cabezal posicionado sobre esta última.
Precondición: *el tablero se encuentra vacío.*

g. Escribir el procedimiento `Victoria` que llena el mapa de monedas, indicando que el jugador ganó el juego.

h. Escribir el procedimiento `Derrota` que llena el mapa de bombas, indicando que el jugador perdió el juego.

i. Escribir el procedimiento `VerificarFinDeJuego` que, en caso que el juego haya terminado, invoca el procedimiento `Victoria` si el jugador recolectó todas las monedas o invoca el procedimiento `Derrota` si Sbog está muerto.

Para correr el programa, reemplace `program` por el siguiente fragmento de código[1] y ejecútelo. La tecla `Enter` inicializa el juego. Las teclas `W`, `A`, `S`, `D`, mueven a Sbog hacia el norte, Oeste, Sur y Este respectivamente.

```
interactive program {
  K_W     -> { MoverSbog(Norte) ; VerificarFinDeJuego() }
  K_A     -> { MoverSbog(Oeste) ; VerificarFinDeJuego() }
  K_S     -> { MoverSbog(Sur)   ; VerificarFinDeJuego() }
  K_D     -> { MoverSbog(Este)  ; VerificarFinDeJuego() }
  K_ENTER -> { VaciarTablero()  ; CargarMapa() }
  _       -> { Skip }
}
```

Puede hacer uso de la vestimenta `ElJuegoDeSbog.xml` incluida en los recursos que acompañan esta práctica (ver figura 2).

[1] El bloque `interactive program` se explica en el libro de Gobstones, "Las Bases Conceptuales de la Programación. Una nueva forma de aprender a programar.", en su sección 5.7.2., disponible en el sitio de la materia.

Figura 2: Vestimenta `ElJuegoDeSbog.xml`