

Repetición condicional + recorridos

Introducción a la programación

Tecnicatura en programación informática,
Licenciatura en Desarrollo de Software,
Universidad Nacional de Quilmes

Lo que veremos hoy

- 1 Repaso
 - Generalidades
 - Estructuras de control
 - Funciones
- 2 Repetición condicional
- 3 Recorridos

Repaso: generalidades

- Elementos de Gobstones
- Programa, comandos, propósito, efectos, etc
- Procedimientos: división en subtareas y abstracción
- Parámetro vs. argumento
- Especificación: contratos y precondition

Repaso: expresiones, valores, tipos

- Valores
 - Cada valor es **igual** a sí mismo, y **diferente** del resto de los valores
 - **Rojo, Azul, Norte, Sur, -1, 0, 1, True, False**, etc

Repaso: expresiones, valores, tipos

- Valores
 - Cada valor es **igual** a sí mismo, y **diferente** del resto de los valores
 - **Rojo, Azul, Norte, Sur, -1, 0, 1, True, False**, etc
- Expresiones
 - Formas de **denotar** valores: **evaluación**
 - Expresiones distintas pueden denotar el mismo valor
 - En distintos instantes, una expresión puede denotar valores distintos

Repaso: expresiones, valores, tipos

- Valores
 - Cada valor es **igual** a sí mismo, y **diferente** del resto de los valores
 - **Rojo, Azul, Norte, Sur, -1, 0, 1, True, False**, etc
- Expresiones
 - Formas de **denotar** valores: **evaluación**
 - Expresiones distintas pueden denotar el mismo valor
 - En distintos instantes, una expresión puede denotar valores distintos
- Tipos
 - Conjunto de valores a los que se les pueden aplicar las mismas operaciones
 - Colores, Direcciones, Booleanos, Números
 - Tipo de una expresión: tipo del valor que denota

Repaso: `repeat`

Repetición simple (`repeat`)

```
repeat(<numero>)  
  <bloque a repetir>
```

- `<numero>` debe ser una expresión numérica
- `<bloque a repetir>` se repite tantas veces como el valor denotado por `<numero>`

Repaso: `foreach`

Repetición indexada (`foreach`)

```
foreach <indice>in [<primero>..<ultimo>]  
<bloque a repetir>
```

- Se repite *<bloque a repetir>* tantas veces como elementos tenga el rango
- *<primero>* y *<ultimo>* DEBEN ser expresiones del mismo tipo
- En la primer repetición, *<indice>* denota *<primero>*
- En la segunda repetición, *<indice>* denota **siguiente**(*<primero>*)
- ...
- En la última repetición, *<indice>* denota *<ultimo>*

Repaso: `foreach`

Repetición indexada (`foreach`)

```
foreach <indice>in [ <primero>..<ultimo> ]  
<bloque a repetir>
```

- Se repite *<bloque a repetir>* tantas veces como elementos tenga el rango
- *<primero>* y *<ultimo>* DEBEN ser expresiones del mismo tipo
- En la primer repetición, *<indice>* denota *<primero>*
- En la segunda repetición, *<indice>* denota **siguiente**(*<primero>*)
- ...
- En la última repetición, *<indice>* denota *<ultimo>*
- ¿De qué tipo es *<indice>*?

Repaso: if-else

Alternativa condicional if-else

```
if <condicion>  
  <bloque caso verdadero>  
else  
  <bloque caso falso>
```

- *<condicion>* DEBE ser una expresión booleana.

Repaso: funciones básicas

- Las función básicas son renombres de expresiones
- Pueden tener parámetros

```
1 function totalBolitas() {  
2     return(nroBolitas(Azul) + nroBolitas(Negro) +  
3         nroBolitas(Rojo) + nroBolitas(Verde))  
4 }  
5  
6 function interna() {  
7     return(puedeMover(Norte) && puedeMover(Este) &&  
8         puedeMover(Sur) && puedeMover(Oeste))  
9 }  
10  
11 function bisiesto(anio) {  
12     return(anio mod 400 == 0 ||  
13         (anio mod 4 == 0 && anio mod 100 /= 0))  
14 }
```

Funciones básicas como expresiones

- Las funciones se **invocan** como los procedimientos
 - Se escribe el nombre de la función seguido por paréntesis y los parámetros

Funciones básicas como expresiones

- Las funciones se **invocan** como los procedimientos
 - Se escribe el nombre de la función seguido por paréntesis y los parámetros
- Las invocaciones de funciones son **expresiones**, i.e. **denotan** valores
 - Se escriben en cualquier lugar en el que sea necesario un valor
 - Tienen un tipo, como cualquier expresión

Funciones básicas como expresiones

- Las funciones se **invocan** como los procedimientos
 - Se escribe el nombre de la función seguido por paréntesis y los parámetros
- Las invocaciones de funciones son **expresiones**, i.e. **denotan** valores
 - Se escriben en cualquier lugar en el que sea necesario un valor
 - Tienen un tipo, como cualquier expresión

```
1 function borde() {  
2     return(not interna())  
3 }  
4  
5 function azulada() {  
6     return(nroBolitas(Azul) > totalBolitas() div 2)  
7 }  
8  
9 procedure PonerSiBisiesto(color, anio) {  
10     if(esBisiesto(anio)) { Poner(color) }  
11 }
```

Funciones con comandos

- Una función puede tener comandos, invocaciones a procedimientos, estructuras de control, etc, pero

Funciones con comandos

- Una función puede tener comandos, invocaciones a procedimientos, estructuras de control, etc, pero
 - La función debe finalizar con la instrucción **return**

Funciones con comandos

- Una función puede tener comandos, invocaciones a procedimientos, estructuras de control, etc, pero
 - La función debe finalizar con la instrucción **return**
 - La instrucción return es única dentro de la función

Funciones con comandos

- Una función puede tener comandos, invocaciones a procedimientos, estructuras de control, etc, pero
 - La función debe finalizar con la instrucción **return**
 - La instrucción **return** es única dentro de la función
 - Los cambios al tablero **desaparecen** cuando la función termina

Funciones con comandos

- Una función puede tener comandos, invocaciones a procedimientos, estructuras de control, etc, pero
 - La función debe finalizar con la instrucción **return**
 - La instrucción **return** es única dentro de la función
 - Los cambios al tablero **desaparecen** cuando la función termina
- Las funciones permiten una mayor expresividad abstracta

Funciones con comandos

- Una función puede tener comandos, invocaciones a procedimientos, estructuras de control, etc, pero
 - La función debe finalizar con la instrucción **return**
 - La instrucción **return** es única dentro de la función
 - Los cambios al tablero **desaparecen** cuando la función termina
- Las funciones permiten una mayor expresividad abstracta

```
1 function nroBolitasAl(c, d) {  
2     Mover(d)  
3     return(nroBolitas(c))  
4     //no hace falta regresar el cabezal  
5 }  
6  
7 function nroBolitasVecinas(c) {  
8     return(nroBolitasAl(c, Norte) + nroBolitasAl(c, Este) +  
9           nroBolitasAl(c, Sur) + nroBolitasAl(c, Oeste))  
10    //el movimiento de nroBolitasAl es transparente.  
11 }
```

Repetición condicional: motivación

- ¿Cómo hacemos para fijarnos si en alguna celda hacia el Este de la celda actual hay una bolita Roja?

Repetición condicional: motivación

- ¿Cómo hacemos para fijarnos si en alguna celda hacia el Este de la celda actual hay una bolita Roja?
- ¿Cómo hacemos para mover el cabezal hacia la celda en el extremo Noreste?

Repetición condicional: motivación

- ¿Cómo hacemos para fijarnos si en alguna celda hacia el Este de la celda actual hay una bolita Roja?
- ¿Cómo hacemos para mover el cabezal hacia la celda en el extremo Noreste?
- ¿Cómo hacemos para eliminar todas las bolitas rojas del tablero?

Repetición condicional: motivación

- ¿Cómo hacemos para fijarnos si en alguna celda hacia el Este de la celda actual hay una bolita Roja?
- ¿Cómo hacemos para mover el cabezal hacia la celda en el extremo Noreste?
- ¿Cómo hacemos para eliminar todas las bolitas rojas del tablero?

```
1 //Mueve el cabezal al este hasta encontrar una bolita roja.  
2 //Precondición: dicha celda debe existir  
3 procedure IrAlEsteHastaRoja() {  
4     while(not hayBolitas(Rojo)) {  
5         Mover(Este)  
6     }  
7 }
```


Repetición condicional: forma general

Repetición condicional (*while*)

```
while(<condicion>)  
<bloque a repetir>
```

donde,

- *<condicion>* es una expresión booleana

Repetición condicional: forma general

Repetición condicional (`while`)

```
while(<condicion>)  
<bloque a repetir>
```

donde,

- `<condicion>` es una expresion booleana
- El `while` se escribe en cualquier bloque
 - pero sin anidar repeticiones.

Repetición condicional: forma general

Repetición condicional (`while`)

```
while(<condicion>)  
<bloque a repetir>
```

donde,

- `<condicion>` es una expresión booleana
- El `while` se escribe en cualquier bloque
 - pero sin anidar repeticiones.
- Ejecuta `<bloque a repetir>` mientras `<condicion>` denota `True`
 - Luego de cada repetición se evalúa `<condicion>` nuevamente

Repetición condicional: ejemplo

- ¿Qué hace PintarHasta?
- ¿PintarHasta es total o parcial?

```
1 //Proposito: ???  
2 procedure PintarHasta(dir, color, freno)  
3     while(puedeMover(dir) && not hayBolitas(freno)) {  
4         PonerYMover(color, dir)  
5     }  
6 }
```

Repetición condicional: ejemplo

- ¿Qué hace PintarHasta?
- ¿PintarHasta es total o parcial?

```
1 //Proposito: ???  
2 procedure PintarHasta(dir, color, freno)  
3     while(puedeMover(dir) && not hayBolitas(freno)) {  
4         PonerYMover(color, dir)  
5     }  
6 }
```

- ¿Qué ocurre si no hay bolitas del color freno?

Repetición condicional: terminación I

- ¿Qué hace el siguiente procedimiento?

```
1 //Proposito: ???  
2 procedure KABOOM() {  
3     Mover(Este)  
4     while(puedeMover(Oeste)) {  
5         Mover(Este)  
6     }  
7     Kaboommmmmm()  
8 }
```

Repetición condicional: terminación I

- ¿Qué hace el siguiente procedimiento?

```
1 //Proposito: ???  
2 procedure KABOOM() {  
3     Mover(Este)  
4     while(puedeMover(Oeste)) {  
5         Mover(Este)  
6     }  
7     Kaboommmmmm()  
8 }
```

- Cada repetición debería acercarnos a la finalización del ciclo

Repetición condicional: terminación II

- ¿Qué hace el siguiente procedimiento?

```
1 //Proposito: ???  
2 procedure DontStopMeNow_ImHavingSuchAGoodTime() {  
3     while(True) {  
4         NoOnelsGonnaStopMeNow() //suponer definida y total  
5     }  
6     NuncaLlegaraAca()  
7 }
```


Repetición condicional: terminación II

- ¿Qué hace el siguiente procedimiento?

```
1 //Proposito: ???  
2 procedure DontStopMeNow_ImHavingSuchAGoodTime() {  
3     while(True) {  
4         NoOnelsGonnaStopMeNow() //suponer definida y total  
5     }  
6     NuncaLlegaraAca()  
7 }
```

- La no terminación es una forma de parcialidad
- ¿Cuál es la precondition del procedimiento?

Repetición condicional: terminación III

- ¿Qué hace el siguiente procedimiento?

```
1 //Proposito: pinta una linea de color c en direccion d,  
2 //dejando un margen a derecha de tamanno margen.  
3 procedure PintarConMargen(c, d, margen) {  
4     while(puedoMoverN(d, margen)) { //en practica  
5         PonerYMover(c, d)  
6     }  
7 }
```

- ¿Cual es la precondition del procedimiento?

Repetición condicional: terminación III

- ¿Qué hace el siguiente procedimiento?

```
1 //Proposito: pinta una linea de color c en direccion d,  
2 //dejando un margen a derecha de tamanno margen.  
3 procedure PintarConMargen(c, d, margen) {  
4     while(puedoMoverN(d, margen)) { //en practica  
5         PonerYMover(c, d)  
6     }  
7 }
```

- ¿Cual es la precondition del procedimiento?
 - Este procedimiento hace BOOM cuando n es menor o igual a 0
 - Su precondition es, justamente, $n > 0$

Repetición condicional: terminación IV

- ¿Qué hace el siguiente procedimiento?

```
1 //Proposito: Se mueve en direccion d de acuerdo a la
2 //secuencia de bolitas de color c.
3 procedure MoverConSaltos(c, d) {
4     while(hayBolitas(c)) {
5         if(puedoMoverN(d, nroBolitas(c))) {
6             MoverN(d, nroBolitas(c))
7         }
8     }
9 }
```

Repetición condicional: terminación IV

- ¿Qué hace el siguiente procedimiento?

```
1 //Proposito: Se mueve en direccion d de acuerdo a la
2 //secuencia de bolitas de color c.
3 procedure MoverConSaltos(c, d) {
4     while(hayBolitas(c)) {
5         if(puedoMoverN(d, nroBolitas(c))) {
6             MoverN(d, nroBolitas(c))
7         }
8     }
9 }
```

- ¿Es correcto el procedimiento?

Repetición condicional: terminación IV

- ¿Qué hace el siguiente procedimiento?

```
1 //Proposito: Se mueve en direccion d de acuerdo a la
2 //secuencia de bolitas de color c.
3 procedure MoverConSaltos(c, d) {
4     while(hayBolitas(c)) {
5         if(puedoMoverN(d, nroBolitas(c))) {
6             MoverN(d, nroBolitas(c))
7         }
8     }
9 }
```

- ¿Es correcto el procedimiento?
- ¿Cómo se puede arreglar?

Repetición condicional: terminación IV

- ¿Qué hace el siguiente procedimiento?

```
1 //Proposito: Se mueve en direccion d de acuerdo a la
2 //secuencia de bolitas de color c.
3 procedure MoverConSaltos(c, d) {
4     while(hayBolitas(c) && puedoMoverN(d, nroBolitas(c))) {
5
6         MoverN(d, nroBolitas(c))
7
8     }
9 }
```

- ¿Es correcto el procedimiento?
- ¿Cómo se puede arreglar?

Repetición condicional: casos borde

- Queremos poner una bolita azul en cada celda de la fila actual
- ¿Es correcto el siguiente procedimiento?

```
1 //Proposito: pone una bolita azul en cada celda de la fila actual?  
2 procedure PintarFilaDeAzul() {  
3     IrAlExtremo(Oeste) //en practica  
4     while (puedeMover(Este)) {  
5         Poner(Azul)  
6         Mover(Este)  
7     }  
8  
9 }
```


Repetición condicional: casos borde

- Queremos poner una bolita azul en cada celda de la fila actual
- ¿Es correcto el siguiente procedimiento?

```
1 //Proposito: pone una bolita azul en cada celda de la fila actual?  
2 procedure PintarFilaDeAzul() {  
3     IrAlExtremo(Oeste) //en practica  
4     while(puedeMover(Este)) {  
5         Poner(Azul)  
6         Mover(Este)  
7     }  
8  
9 }
```

- ¿Cómo se puede arreglar el procedimiento?

Repetición condicional: casos borde

- Queremos poner una bolita azul en cada celda de la fila actual
- ¿Es correcto el siguiente procedimiento?

```
1 //Proposito: pone una bolita azul en cada celda de la fila actual?  
2 procedure PintarFilaDeAzul() {  
3     IrAlExtremo(Oeste) //en practica  
4     while (puedeMover(Este)) {  
5         Poner(Azul)  
6         Mover(Este)  
7     }  
8     Poner(Azul)  
9 }
```

- ¿Cómo se puede arreglar el procedimiento?

Repetición condicional y funciones

- Escribir una función `todasAlEsteConColor(c)` que indique si todas las celdas al este de la celda actual tienen bolitas de color `c`
- Escribir un procedimiento `PonerSiTodasAlEsteConColor(c, pone)` que ponga una bolita de color `pone` en la celda actual si todas las celdas al este de la actual tienen bolitas de color `c`

Repetición condicional y funciones

- Escribir una función `todasAlEsteConColor(c)` que indique si todas las celdas al este de la celda actual tienen bolitas de color `c`
- Escribir un procedimiento `PonerSiTodasAlEsteConColor(c, pone)` que ponga una bolita de color `pone` en la celda actual si todas las celdas al este de la actual tienen bolitas de color `c`
- Discutir los beneficios de que las funciones no produzcan efectos

Repetición condicional: resumen

- La repetición condicional permite repetir un bloque de acuerdo a una condición, sin saber a priori cuántas repeticiones se van a ejecutar

Repetición condicional: resumen

- La repetición condicional permite repetir un bloque de acuerdo a una condición, sin saber a priori cuántas repeticiones se van a ejecutar
- Puede generar parcialidad por no terminación
 - Hay que tenerlo en cuenta en la precondition

Repetición condicional: resumen

- La repetición condicional permite repetir un bloque de acuerdo a una condición, sin saber a priori cuántas repeticiones se van a ejecutar
- Puede generar parcialidad por no terminación
 - Hay que tenerlo en cuenta en la precondition
- Cada repetición del bloque nos debería acercar a la condición de terminación
 - Saber si un ciclo va a terminar es difícil

Repetición condicional: resumen

- La repetición condicional permite repetir un bloque de acuerdo a una condición, sin saber a priori cuántas repeticiones se van a ejecutar
- Puede generar parcialidad por no terminación
 - Hay que tenerlo en cuenta en la precondition
- Cada repetición del bloque nos debería acercar a la condición de terminación
 - Saber si un ciclo va a terminar es difícil
- Casos borde: asegurarse de procesar las primeras o últimas celdas

Repetición condicional: resumen

- La repetición condicional permite repetir un bloque de acuerdo a una condición, sin saber a priori cuántas repeticiones se van a ejecutar
- Puede generar parcialidad por no terminación
 - Hay que tenerlo en cuenta en la precondition
- Cada repetición del bloque nos debería acercar a la condición de terminación
 - Saber si un ciclo va a terminar es difícil
- Casos borde: asegurarse de procesar las primeras o últimas celdas
- La combinación de funciones y repeticiones condicionales es poderosa

Recorridos de filas: motivación

- Escribir un procedimiento `PonerEnFila(c)` que ponga una bolita de color `c` en cada celda de la fila actual

Recorridos de filas: motivación

- Escribir un procedimiento `PonerEnFila(c)` que ponga una bolita de color `c` en cada celda de la fila actual
- Escribir un procedimiento `SacarDeFila(c)` que saque una bolita de color `c` de cada celda de la fila actual

Recorridos de filas: motivación

- Escribir un procedimiento `PonerEnFila(c)` que ponga una bolita de color `c` en cada celda de la fila actual
- Escribir un procedimiento `SacarDeFila(c)` que saque una bolita de color `c` de cada celda de la fila actual
- Escribir un procedimiento `DuplicarFila(c)` que duplique la cantidad de bolitas de color `c` de cada celda de la fila actual

Recorridos de filas: motivación

- Escribir un procedimiento `PonerEnFila(c)` que ponga una bolita de color `c` en cada celda de la fila actual
- Escribir un procedimiento `SacarDeFila(c)` que saque una bolita de color `c` de cada celda de la fila actual
- Escribir un procedimiento `DuplicarFila(c)` que duplique la cantidad de bolitas de color `c` de cada celda de la fila actual
- ¿Nota alguna similitud entre los procedimientos?

Esquema de recorrido de filas (columnas)

- Sirve para recorrer todas las celdas de una fila de Oeste a Este
- Es un esquema; hay que completar las partes

```
1 esquema RecorrerFilaDeOesteAEste (<params>)
2   IrAlBorde (Oeste)
3   while (puedeMover (Este)) {
4     <ProcesarCelda> (<args>)
5     Mover (Este)
6   }
7   <ProcesarCelda> (<args>)
8 }
```

Esquema de recorrido de filas (columnas)

- Sirve para recorrer todas las celdas de una fila de Oeste a Este
- Es un esquema; hay que completar las partes

```
1 esquema RecorrerFilaDeOesteAEste (<params> )
2   IrAlBorde (Oeste)
3   while (puedeMover (Este)) {
4     <ProcesarCelda> (<args> )
5     Mover (Este)
6   }
7   <ProcesarCelda> (<args> )
8 }
```

- ¿Qué habría que cambiar si se quiere recorrer de Este a Oeste?

Esquema de recorrido de filas (columnas)

- Sirve para recorrer todas las celdas de una fila de Oeste a Este
- Es un esquema; hay que completar las partes

```
1 esquema RecorrerFilaDeOesteAEste (<params> )
2   IrAlBorde (Oeste)
3   while (puedeMover (Este)) {
4     <ProcesarCelda> (<args> )
5     Mover (Este)
6   }
7   <ProcesarCelda> (<args> )
8 }
```

- ¿Qué habría que cambiar si se quiere recorrer de Este a Oeste?
- ¿Qué habría que cambiar si se quiere recorrer una columna?

Esquema de recorrido de filas (columnas)

- Sirve para recorrer todas las celdas de una fila de Oeste a Este
- Es un esquema; hay que completar las partes

```
1  esquema RecorrerEnDir<d> (<params>)  
2      IrAlBorde (opuesto(d))  
3      while (puedeMover(d)) {  
4          <ProcesarCelda> (<args>)  
5          Mover(d)  
6      }  
7      <ProcesarCelda> (<args>)  
8  }
```

- ¿Qué habría que cambiar si se quiere recorrer de Este a Oeste?
- ¿Qué habría que cambiar si se quiere recorrer una columna?

Recorridos de tableros: motivación

- Escribir un procedimiento `PonerEnTodas(c)` que ponga una bolita de color `c` en cada celda del tablero

Recorridos de tableros: motivación

- Escribir un procedimiento `PonerEnTodas(c)` que ponga una bolita de color `c` en cada celda del tablero
- Escribir un procedimiento `SacarDeTodas(c)` que saque una bolita de color `c` de cada celda del tablero

Recorridos de tableros: motivación

- Escribir un procedimiento `PonerEnTodas(c)` que ponga una bolita de color `c` en cada celda del tablero
- Escribir un procedimiento `SacarDeTodas(c)` que saque una bolita de color `c` de cada celda del tablero
- Escribir un procedimiento `DuplicarTodas(c)` que duplique la cantidad de bolitas de color `c` de cada celda del tablero

Recorridos de tableros: motivación

- Escribir un procedimiento `PonerEnTodas(c)` que ponga una bolita de color `c` en cada celda del tablero
- Escribir un procedimiento `SacarDeTodas(c)` que saque una bolita de color `c` de cada celda del tablero
- Escribir un procedimiento `DuplicarTodas(c)` que duplique la cantidad de bolitas de color `c` de cada celda del tablero
- ¿Nota alguna similitud entre los procedimientos?

Esquema de recorrido de tableros Norte + Este

- Sirve para recorrer todas las celdas de un tablero
 - En sentido primero Norte, después Este
 - **No usar:** conviene usar el esquema general

```
1 esquema RecorrerTableroNorteEste (<params>)
2   IrAlOrigen ()
3   while (puedeMover (Norte) || puedeMover (Este)) {
4     <ProcesarCelda> (<args>)
5     MoverNorteEste ()
6   }
7   <ProcesarCelda> (<args>)
8 }
9
10 //Precondición: puede mover al norte o este
11 procedure MoverNorteEste () {
12   if (puedeMover (Norte)) {
13     Mover (Norte)
14   } else {
15     IrAlBorde (Sur)
16     Mover (Este)
17   }
18 }
```

Parametrizando recorridos

- Objetivo: recorrer en distintos sentidos
- Primero en una dirección `dir_int` y luego en otra `dir_ext`

Parametrizando recorridos

- Objetivo: recorrer en distintos sentidos
- Primero en una dirección `dir_int` y luego en otra `dir_ext`

```
1 //Proposito: posiciona el cabezal en la primer celda del recorrido
2 procedure IrAlInicioT(dir_int , dir_ext) {
3     IrAlBorde(opuesto(dir_int)); IrAlBorde(opuesto(dir_ext))
4 }
5
6 //Proposito: denota True si hay mas celdas por recorrer
7 function puedeMoverT(dir_int , dir_ext) {
8     return(puedeMover(dir_int) || puedeMover(dir_ext))
9 }
10
11 //Proposito: Pasa a la siguiente celda del recorrido
12 //Precondicion: puedeMoverT(dir_int , dir_ext)
13 procedure MoverT(dir_int , dir_ext) {
14     if (puedeMover(dir_int)) {Mover(dir_int)}
15     else {IrAlBorde(opuesto(dir_int)); Mover(dir_ext)}
16 }
```


Esquema general de recorrido de tableros

- Sirve para recorrer todas las celdas de un tablero
 - Depende de dos parámetros que indican la dirección
 - Primero `dir_int` y luego `dir_ext`

```
1 esquema RecorrerTablero<dir_int , dir_ext> (<params> )  
2   IrAllInicioT(dir_int , dir_ext)  
3   while(puedeMoverT(dir_int , dir_ext)) {  
4       <ProcesarCelda>(<args>)  
5       MoverT(dir_int , dir_ext)  
6   }  
7   <ProcesarCelda>(<args>)  
8 }
```

- Observar que es **igual** al recorrido por filas o columnas

Recorridos de búsqueda: motivación

- Escribir un procedimiento `IrAColor(c)` que posicione el cabezal en alguna celda que tenga una bolita de color `c`; suponer que dicha celda existe

Recorridos de búsqueda: motivación

- Escribir un procedimiento `IrAColor(c)` que posicione el cabezal en alguna celda que tenga una bolita de color `c`; suponer que dicha celda existe
- Escribir una función `hayBolitasTablero(c)` que indique si el tablero tiene al menos una bolita de color `c`

Recorridos de búsqueda: motivación

- Escribir un procedimiento `IrAColor(c)` que posicione el cabezal en alguna celda que tenga una bolita de color `c`; suponer que dicha celda existe
- Escribir una función `hayBolitasTablero(c)` que indique si el tablero tiene al menos una bolita de color `c`
- Escribir una función `hayVacía()` que indique si el tablero tiene una celda vacía

Recorridos de búsqueda: motivación

- Escribir un procedimiento `IrAColor(c)` que posicione el cabezal en alguna celda que tenga una bolita de color `c`; suponer que dicha celda existe
- Escribir una función `hayBolitasTablero(c)` que indique si el tablero tiene al menos una bolita de color `c`
- Escribir una función `hayVacía()` que indique si el tablero tiene una celda vacía

- ¿Qué cambia con respecto a los recorridos anteriores?

Esquema de búsqueda en un tablero

- Posiciona el cabezal en alguna celda
- Podemos usarlo en una función para determinar si dicha celda existe

```
1 esquema BuscarEnTablero<dir_int , dir_ext> (<params>) {  
2     IrAlInicioT(dir_int , dir_ext)  
3     while (not esCeldaBuscada(<args>)  
4         /*&& puedeMoverT(dir_int , dir_ext)*/)  
5     {  
6         MoverT(dir_int , dir_ext)  
7     }  
8     /* return (esCeldaBuscada(it!<args>!)) */  
9 }
```

Esquema de búsqueda en un tablero

- Posiciona el cabezal en alguna celda
- Podemos usarlo en una función para determinar si dicha celda existe

```
1 esquema BuscarEnTablero<dir_int , dir_ext> (<params>) {  
2     IrAlInicioT(dir_int , dir_ext)  
3     while(not esCeldaBuscada(<args>)  
4         /*&& puedeMoverT(dir_int , dir_ext)*/)  
5     {  
6         MoverT(dir_int , dir_ext)  
7     }  
8     /* return(esCeldaBuscada(it!<args>!)) */  
9 }
```

- Tener cuidado con las precondiciones

Recorrido general I

- Los elementos a recorrer no tienen por qué ser celdas
- El esquema siguiente funciona siempre que los elementos tengan algún orden
 - e.g., recorrer las celdas desde la que tiene menor cantidad de bolitas rojas a la que más tiene

```
1 esquema RecorridoConOrden (<params>) {  
2   <IrAPrimerElemento> (<args>)  
3   while (<puedeMoverASiguienteElemento (<args>)>) {  
4     <ProcesarElemento> (<args>)  
5     <MoverASiguienteElemento> (<args>)  
6   }  
7   <ProcesarElemento> (<args>)  
8 }
```


Recorrido general I

- Los elementos a recorrer no tienen por qué ser celdas
- El esquema siguiente funciona siempre que los elementos tengan algún orden
 - e.g., recorrer las celdas desde la que tiene menor cantidad de bolitas rojas a la que más tiene

```
1 esquema RecorridoConOrden (<params>) {  
2   <IrAPrimerElemento> (<args>)  
3   while (<puedeMoverASiguienteElemento (<args>)>) {  
4     <ProcesarElemento> (<args>)  
5     <MoverASiguienteElemento> (<args>)  
6   }  
7   <ProcesarElemento> (<args>)  
8 }
```

- La dificultad está en completar el esquema!

Recorrido general II

- El siguiente esquema funciona cuando no se puede establecer un orden de los elementos
 - Requiere que los elementos puedan “sacarse” del recorrido (e.g., marcandolos)
 - e.g., recorrer todas las celdas alcanzables desde una celda prefijada, recorriendo sólo celdas con bolitas azules

```
1 esquema RecorridoDesordenado(<params>) {  
2     while(hayElementosParaProcesar(<args>)) {  
3         <BuscarUnElemento>(<args>)  
4         <ProcesarElementoYQuitarloDelRecorrido>(<args>)  
5     }  
6 }
```

- Todos los elementos se procesan dentro del `while`

Recorridos: resumen

- Esquema general para recorrer **elementos** dentro del tablero
 - Los elementos pueden ser celdas, filas, columnas, caminos, regiones, etc

Recorridos: resumen

- Esquema general para recorrer **elementos** dentro del tablero
 - Los elementos pueden ser celdas, filas, columnas, caminos, regiones, etc
- Existen muchas especializaciones de los esquemas de recorridos
 - Recorrido total, búsqueda, posicionamiento del cabezal, etc
 - En un orden predeterminado o buscando elemento a elemento

Recorridos: resumen

- Esquema general para recorrer **elementos** dentro del tablero
 - Los elementos pueden ser celdas, filas, columnas, caminos, regiones, etc
- Existen muchas especializaciones de los esquemas de recorridos
 - Recorrido total, búsqueda, posicionamiento del cabezal, etc
 - En un orden predeterminado o buscando elemento a elemento
- Conviene dividir en subtarefas el recorrido del procesamiento