Repaso + Variables

Introducción a la programación

Universidad Nacional de Quilmes

Introducción a la programación

Lo que veremos hoy

1. Repaso
   - Generalidades
   - Repetición y alternativa
   - Funciones
   - Recorridos
2. Variables

Repaso: generalidades

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

Repaso: expresiones, valores, tipos

- Valores
  - Cada valor es igual a sí mismo, y diferente del resto de los valores
  - ●, ●, ↑, ↓, **-1**, **0**, **1**, VERDADERO, FALSO, etc

Repaso: expresiones, valores, tipos

- Valores
  - Cada valor es igual a sí mismo, y diferente del resto de los valores
  - ●, ●, ↑, ↓, **-1**, **0**, **1**, VERDADERO, FALSO, etc
- Expresiones
  - Formas de denotar valores: evaluación
  - Expresiones distintas pueden denotar el mismo valor
  - En distintos instantes, una expresión puede denotar valores distintos
  - Los argumentos son expresiones y las expresiones se usan como argumentos

Repaso: expresiones, valores, tipos

- Valores
  - Cada valor es igual a sí mismo, y diferente del resto de los valores
  - ●, ●, ↑, ↓, **-1**, **0**, **1**, VERDADERO, FALSO, etc
- Expresiones
  - Formas de denotar valores: evaluación
  - Expresiones distintas pueden denotar el mismo valor
  - En distintos instantes, una expresión puede denotar valores distintos
  - Los argumentos son expresiones y las expresiones se usan como argumentos
- 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: `while`

Repetición condicional `while`

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

- *<condicion>* debe ser una expresión booleana
- *<bloque a repetir>* se repite mientras *<condicion>* denote Verdadero

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 }
```

Esquema de recorrido de filas (columnas)

- Sirve para recorrer todas las celdas de una fila o columna
- 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  }
```

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      IrAlInicioT(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

Funciones paramétricas de recorridos

```
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 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

Variables: motivación

- Como podríamos:
  - Escribir una función `colorDominante(ca, cb)` que, dados dos colores `ca`, `cb`, denote el color del que más bolitas haya.

Variables: motivación

- Como podríamos:
  - Escribir una función `colorDominante(ca, cb)` que, dados dos colores `ca`, `cb`, denote el color del que más bolitas haya.
  - Escribir una función `direccionDominante(c)` que, dado un color `c`, denote la dirección lindante en la que hay más bolitas de color `c`.

Variables: motivación

- Como podríamos:
  - Escribir una función `colorDominante(ca, cb)` que, dados dos colores `ca`, `cb`, denote el color del que más bolitas haya.
  - Escribir una función `direccionDominante(c)` que, dado un color `c`, denote la dirección lindante en la que hay más bolitas de color `c`.

```
1  procedure colorDominante(ca, cb) {
2    elDominante := ca
3    if(nroBolitas(ca) < nroBolitas(cb)) {
4      elDominante := cb
5    }
6    return(elDominante)
7  }
```

Variables: definición

- Las variables son expresiones no literales:
  - Denotan un valor y tienen un tipo
  - Pueden denotar distintos valores en distintos instantes
  - Se usan como argumentos y sólo como argumentos

Variables: definición

- Las variables son expresiones no literales:
  - Denotan un valor y tienen un tipo
  - Pueden denotar distintos valores en distintos instantes
  - Se usan como argumentos y sólo como argumentos
- Se escriben usando un nombre en minúscula: identificador
  - Como lo hacemos con los parámetros e índices
  - e.g.: `bolitasRojas`; `deboSeguir`; `colorRequerido`; `direRetorno`; etc.

Variables: definición

- Las variables son expresiones no literales:
  - Denotan un valor y tienen un tipo
  - Pueden denotar distintos valores en distintos instantes
  - Se usan como argumentos y sólo como argumentos
- Se escriben usando un nombre en minúscula: identificador
  - Como lo hacemos con los parámetros e índices
  - e.g.: `bolitasRojas`; `deboSeguir`; `colorRequerido`; `direRetorno`; etc.
- Cualquier identificador que no es un parámetro o un índice es una variable

Variables: definición

- Las variables son expresiones no literales:
  - Denotan un valor y tienen un tipo
  - Pueden denotar distintos valores en distintos instantes
  - Se usan como argumentos y sólo como argumentos
- Se escriben usando un nombre en minúscula: identificador
  - Como lo hacemos con los parámetros e índices
  - e.g.: `bolitasRojas`; `deboSeguir`; `colorRequerido`; `direRetorno`; etc.
- Cualquier identificador que no es un parámetro o un índice es una variable
- Lo que hace especial a las variables es que se pueden usar para recordar valores.

Asignación de variables I: definición

Comando de asignación :=

variable := *expresion* es un comando.

- *variable* debe ser una variable
- *expresion* es cualquier expresión
- Efecto: *variable* denota el valor de evaluar *expresion*.

Asignación de variables I: definición

Comando de asignación :=

variable := *expresion* es un comando.

- *variable* debe ser una variable
- *expresion* es cualquier expresión
- Efecto: *variable* denota el valor de evaluar *expresion*.

```
1  procedure colorDominante(ca, cb) {
2      elDominante := ca
3      if(nroBolitas(ca) > nroBolitas(cb)}
4          elDominante := cb
5      }
6      return(elDominante)
7  }
```

Asignación de variables I: tipo

Comando de asignación :=

variable := *expresion* es un comando.

- *variable* debe ser una variable
- *expresion* es cualquier expresión
- Efecto: *variable* denota el valor de evaluar *expresion*.
- Tipos: *expresion* debe tener el mismo tipo que *variable*

Asignación de variables I: tipo

Comando de asignación :=

variable := *expresion* es un comando.

- *variable* debe ser una variable
- *expresion* es cualquier expresión
- Efecto: *variable* denota el valor de evaluar *expresion*.
- Tipos: *expresion* debe tener el mismo tipo que *variable*

- La primer asignacion determina el tipo de *variable*

Asignación de variables I: tipo

Comando de asignación :=

variable:= *expresion* es un comando.

- *variable* debe ser una variable
- *expresion* es cualquier expresión
- Efecto: *variable* denota el valor de evaluar *expresion*.
- Tipos: *expresion* debe tener el mismo tipo que *variable*

- La primer asignacion determina el tipo de *variable*

```
1  procedure ErrorDeTipo() {
2    //varios comandos
3    var := 1 //var tiene tipo numero
4    //varios comandos
5    var := True //error de tipo; True no es numero
6    //varios comandos
7  }
```

Variables, parámetros e índices: alcance e inicialización

- Un identificador está inicializado si denota un valor válido
- Un identificador no inicializado no puede usarse (ya que denota BOOM).

Variables, parámetros e índices: alcance e inicialización

- Un identificador está inicializado si denota un valor válido
- Un identificador no inicializado no puede usarse (ya que denota BOOM).
- El alcance de un identificador indica el bloque de código en que puede utilizarse el identificador.

Variables, parámetros e índices: alcance e inicialización

- Un identificador está inicializado si denota un valor válido
- Un identificador no inicializado no puede usarse (ya que denota BOOM).
- El alcance de un identificador indica el bloque de código en que puede utilizarse el identificador.

Identificador	Alcance	Inicialización
Parámetros	Bloque Procedimiento	Invocación
Índices	Bloque foreach	Sentencia foreach
Variables	Bloque Procedimiento	Primer asignación

- La regla del nombre y apellido vale para las variables

Variables: inicialización

- ¿Cuál es el error de la siguiente función?

```
1  //Denota si hay una celda lindante en d con bolitas de color c
2  procedure lindanteConColor(d, c) {
3    if(puedeMover(d)) {
4      Mover(d)
5      conBolitas := hayBolitas(c)
6    }
7    return(conBolitas)
8  }
```

- ¿Cómo se puede arreglar?

Variables: inicialización

- ¿Cuál es el error de la siguiente función?

```
1  //Denota si hay una celda lindante en d con bolitas de color c
2  procedure lindanteConColor(d, c) {
3    if(puedeMover(d)) {
4      Mover(d)
5      conBolitas := hayBolitas(c)
6    }
7    return(conBolitas)
8  }
```

- ¿Cómo se puede arreglar?
- Siempre acordarse de inicializar las variables
  - Sugerencia: inicilizarlas lo antes posible

Variables innecesarias I

- ¿Cómo se puede mejorar la siguiente función?

```
1  //Denota si hay una celda lindante en d con bolitas de color c
2  procedure lindanteConColor(d, c) {
3    conBolitas := False
4
5    if(puedeMover(d)) {
6      Mover(d)
7      conBolitas := hayBolitas(c)
8    }
9
10   return(conBolitas)}
11 }
```

- Pensar en expresiones.

Variables innecesarias I

- ¿Cómo se puede mejorar la siguiente función?

```
1  //Denota si hay una celda lindante en d con bolitas de color c
2  procedure lindanteConColor(d, c) {
3    return(puedeMover(d) && hayBolitasAl(d, c))
4  }
5
6  //Denota True si hay bolita de color c en lindante en d
7  //Precondicion: puedeMover(d)
8  procedure hayBolitasAl(d, c) {
9    Mover(d)
10   return(hayBolitas(c))
11 }
```

- Pensar en expresiones.
- Conviene evitar las variables innecesarias, ya que hacen menos expresivo el código.

Acumuladores

- ¿Qué hace `nroBolitasLindantes(Azul)`?

```
1  function nroBolitasLindantes(c) {
2    bolitas := 0
3    foreach d in [minDir()..maxDir()] {
4      bolitas := bolitas + nroBolitasAl(c,d)
5    }
6    return(bolitas)
7  }
```

Acumuladores

- ¿Qué hace `nroBolitasLindantes(Azul)`?

```
1  function nroBolitasLindantes(c) {
2    bolitas := 0
3    foreach d in [minDir()..maxDir()] {
4      bolitas := bolitas + nroBolitasAl(c,d)
5    }
6    return(bolitas)
7  }
```

- `bolitas := bolitas + nroBolitasAl(c,d)`
  - Primero se evalúa `bolitas + nroBolitasAl(c,d)`
  - Luego, `bolitas` recuerda el valor obtenido

Acumuladores

- ¿Qué hace `nroBolitasLindantes(Azul)`?

```
1  function nroBolitasLindantes(c) {
2    bolitas := 0
3    foreach d in [minDir()..maxDir()] {
4      bolitas := bolitas + nroBolitasAl(c,d)
5    }
6    return(bolitas)
7  }
```

- `bolitas := bolitas + nroBolitasAl(c,d)`
  - Primero se evalúa `bolitas + nroBolitasAl(c,d)`
  - Luego, `bolitas` recuerda el valor obtenido
- De esta forma, podemos contar o acumular valores

Variables y recorridos

- Escribir una función `nroBolitasTablero(c)` que cuente la cantidad de bolitas de color `c` en el tablero.
- Organizarlo como un recorrido.

Variables y recorridos

- Escribir una función `nroBolitasTablero(c)` que cuente la cantidad de bolitas de color `c` en el tablero.
- Organizarlo como un recorrido.

```
1  function nroBolitasTablero(c) {
2    bolitas := 0
3    IrAPrimeroT(Norte, Este)
4    while(puedeMoverT(Norte, Este)) {
5      bolitas := bolitas + nroBolitas(c)
6      MoverT(Norte, Este)
7    }
8    return(bolitas + nroBolitas(c))
9  }
```

Variables y recorridos

- Escribir una función `nroBolitasTablero(c)` que cuente la cantidad de bolitas de color `c` en el tablero.
- Organizarlo como un recorrido.

```
function nroBolitasTablero(c) {
  bolitas := 0
  IrAPrimeroT(Norte, Este)
  while(puedeMoverT(Norte, Este)) {
    bolitas := bolitas + nroBolitas(c)
    MoverT(Norte, Este)
  }
  return(bolitas + nroBolitas(c))
}
```

- Las variables son esenciales para contar valores mientras nos movemos

Variables innecesarias II

- ¿Cómo se puede mejorar el siguiente procedimiento?

```
1  //Deja el cabezal en una celda con bolitas de color c
2  //hay al menos una celda con bolitas de color c
3  procedure IrAColor(c) {
4    IrAPrimeroT(Norte, Este)}
5    encontre := hayBolitas(c)
6    while(not encontre) {
7      MoverT(Norte, Este)
8      encontre := hayBolitas(c)
9    }
10 }
```

- Pensar en recorridos de busqueda.

Variables innecesarias II

- ¿Cómo se puede mejorar el siguiente procedimiento?

```
1  //Deja el cabezal en una celda con bolitas de color c
2  //hay al menos una celda con bolitas de color c
3  procedure IrAColor(c) {
4    IrAPrimeroT(Norte, Este)
5
6    while(not hayBolitas(c)) {
7      MoverT(Norte, Este)
8    }
9
10 }
```

- Pensar en recorridos de busqueda.
- Las variables se usan para recordar valores a los que no tenemos acceso de otra forma.

Variables innecesarias II

- ¿Cómo se puede mejorar el siguiente procedimiento?

```
1  //Deja el cabezal en una celda con bolitas de color c
2  //hay al menos una celda con bolitas de color c
3  procedure IrAColor(c) {
4    IrAPrimeroT(Norte, Este)
5
6    while(not hayBolitas(c)) {
7      MoverT(Norte, Este)
8    }
9
10 }
```

- Pensar en recorridos de busqueda.
- Las variables se usan para recordar valores a los que no tenemos acceso de otra forma.
- NO USARLAS COMO FLAGS

Variables: resumen

- Las variables son expresiones que recuerdan valores.
- La operación de asignación (`:=`) permite establecer el valor que denota una variable.
- Las variables sólo se pueden utilizar si fueron inicializadas.
- Como cualquier expresión, las variables tienen tipo, y pueden ser usadas como argumentos y SOLO como argumentos.
- Recordar: el alcance de las variables y los parámetros es el procedimiento, mientras que el alcance de los índices es el bloque del `foreach`.
- No usar variables innecesarias; siempre privilegiar las expresiones.