Introducción a registros

Introducción a la programación

Tecnicatura en Programación Informática,
Universidad Nacional de Quilmes

Introducción a la programación

Hoy en Introducción a la Programación

Motivación

- Lenguaje de programación
  - Comunicación Humano-Computadora
  - Comunicación Humano-Humano
- Abstracción $\rightarrow$ describir el problema en términos del dominio
- Procedimientos – Funciones – Listas – Registros

Idea de los registros

- Un registro es una forma de agrupar un conjunto de campos (datos)
  - Siempre tiene la misma cantidad de campos
  - Los campos pueden tener tipos diferentes
  - No hay orden entre los campos; los valores se referencian con identificadores (nombres de campos)

Idea de los registros

- Un registro es una forma de agrupar un conjunto de campos (datos)
  - Siempre tiene la misma cantidad de campos
  - Los campos pueden tener tipos diferentes
  - No hay orden entre los campos; los valores se referencian con identificadores (nombres de campos)
- Objetivo: modelar entidades del dominio de aplicación

Idea de los registros

- Un registro es una forma de agrupar un conjunto de campos (datos)
  - Siempre tiene la misma cantidad de campos
  - Los campos pueden tener tipos diferentes
  - No hay orden entre los campos; los valores se referencian con identificadores (nombres de campos)
- Objetivo: modelar entidades del dominio de aplicación
- Abstracción, reutilización

Idea de los registros

- Un registro es una forma de agrupar un conjunto de campos (datos)
  - Siempre tiene la misma cantidad de campos
  - Los campos pueden tener tipos diferentes
  - No hay orden entre los campos; los valores se referencian con identificadores (nombres de campos)
- Objetivo: modelar entidades del dominio de aplicación
- Abstracción, reutilización

- Ejemplos:
  - Posición: fila + columna
  - Jugador: puntos, vidas y movimiento
  - Celda: cantidad de azul, negro, rojo y verde

Declaración de registros I

- Para utilizar un registro es necesario declarar su estructura

Declaración de registros I

- Para utilizar un registro es necesario declarar su estructura
  - Fuera de toda función o procedimiento

Declaración de registros I

- Para utilizar un registro es necesario declarar su estructura
  - Fuera de toda función o procedimiento
  - Antes del primer uso de un registro con dicha estructura

Declaración de registros I

- Para utilizar un registro es necesario declarar su estructura
  - Fuera de toda función o procedimiento
  - Antes del primer uso de un registro con dicha estructura
  - Con la palabra clave `type ... is record`

Declaración de registros I

- Para utilizar un registro es necesario declarar su estructura
  - Fuera de toda función o procedimiento
  - Antes del primer uso de un registro con dicha estructura
  - Con la palabra clave `type ... is record`
  - Especificando el nombre del registro NombreRegistro (primer letra en mayúscula)
  - Especificando cada uno de los campos del registro

Declaración de registros I

- Para utilizar un registro es necesario declarar su estructura
  - Fuera de toda función o procedimiento
  - Antes del primer uso de un registro con dicha estructura
  - Con la palabra clave `type ... is record`
  - Especificando el nombre del registro NombreRegistro (primer letra en mayúscula)
  - Especificando cada uno de los campos del registro

- Ejemplos de declaraciones:

```
1  type Posicion is record {
2    field fila
3    field columna
4  }
5
```

```
type Jugador is record {
  field puntos
  field vidas
  field movimiento
}
```

Declaración de registros II

- En general, si los campos del registro son:

  $$c_1, c_2, \ldots, c_k$$

  la declaración toma la forma siguiente:

```
1  type NombeRegistro is record {
2    field c1
3    field c2
4    ...
5    field ck
6  }
```

Documentación de registros

- Junto con la declaración

```
1  /**
2   * Posicion modela una celda del tablero de acuerdo a la fila
3   * y la columna en que se encuentra.
4   */
5  type Posicion is record {
6    field fila    //numero
7    field columna //numero
8  }
9
10 /**
11  * Un jugador se modela con la cantidad de puntos y vidas actuales,
12  * y la direccion en la que se movera en el siguiente turno.
13  */
14 type Jugador is record {
15   field puntos     //numero
16   field vidas      //numero
17   field movimiento //direccion
18 }
```

Ejercicios

- Declarar los siguientes registros
  - Rectangulo: con campos `base` y `altura`
  - Avion: con campos `movimiento`, `velocidad`, y `altura`
  - Guerrero: con campos `fuerza`, `vidas`, `raza`
- Declarar un registro que represente una línea que tenga un color, una orientación y una longitud

Valores, expresiones y tipos (repaso)

- Valores: •, ↑, -**1**, **1**, VERDADERO, [], [**1**], etc.
  - Cada valor es igual a sí mismo, y diferente del resto de los valores.
- Expresiones: formas de denotar valores; evaluación.
- Tipos: conjunto de valores a los que se les pueden aplicar las mismas operaciones.
  - Colores, Direcciones, Booleanos, Números, Listas de un tipo.
  - Tipo de una expresión: tipo del valor que denota.

Tipos definidos por el usuario

- Cada registro define un nuevo tipo en GOBSTONES.
  - Tipos: `Posicion`, `Guerrero`, `Jugador`, `Celda`, etc.

Tipos definidos por el usuario

- Cada registro define un nuevo tipo en GOBSTONES.
  - Tipos: `Posicion`, `Guerrero`, `Jugador`, `Celda`, etc.
- Valores de un registro: combinación arbitraria de los valores de los campos.

Tipos definidos por el usuario

- Cada registro define un nuevo tipo en GOBSTONES.
  - Tipos: `Posicion`, `Guerrero`, `Jugador`, `Celda`, etc.
- Valores de un registro: combinación arbitraria de los valores de los campos.

- Valores de registros
  - Posicion(fila $\leftarrow$ **15**, columna $\leftarrow$ **1**)
  - Jugador(puntos $\leftarrow$ **10**, vidas $\leftarrow$ **2**, movimiento $\leftarrow$ "este"(*))
  - Guerrero(fuerza $\leftarrow$ **12**, vidas $\leftarrow$ **2**, raza $\leftarrow$ •)

(*): se evita escribir $\rightarrow$ para el valor de la dirección este, por cuestiones tipográficos

Valores de tipo registro

- Dada la declaración

```
1  type NombeRegistro is record {
2    field c_1
3    ...
4    field c_k
5  }
```

- Los valores de tipo `NombreRegistro` se expresan como

$$\text{NombreRegistro}(c_1 \leftarrow v_1, \ldots, c_k \leftarrow v_k)$$

donde v_i es el valor que denota el campo c_i

Expresiones que denotan registros

- Si $e_1, \ldots, e_n$ son expresiones, entonces

$$\texttt{Tipo\_Registro(campo}_1 \texttt{ <- } e_1\texttt{, ..., campo}_n \texttt{ <- } e_n\texttt{)}$$

- denota un registro de tipo `Tipo_Registro` en el que campo $\texttt{campo}_1$ denota el valor de e_1, $\texttt{campo}_2$ denota el valor de $e_2, \ldots,$ $\texttt{campo}_n$ denota el valor de e_n.
- Ejemplos:

```
1 Posicion(fila <- 2, columna <- 7)
2 Posicion(fila <- n, columna <- m) //n y m son expresiones
3 Guerrero(fuerza <- 140, vidas <- 7, raza <- minColor())
```

Invariante de representación

- Cada registro define un nuevo tipo en GOBSTONES

```
1  /**
2   * Posicion modela una celda del tablero de acuerdo a la fila
3   * y la columna en que se encuentra.
4   */
5  type Posicion is record {
6    field fila    // numero
7    field columna // numero
8  }
```

- ¿Es válida la posición Posicion(fila ← -1, columna ← 0)?
- Valores válidos

Invariante de representación

- Condiciones que satisfacen los valores válidos de un registro

Invariante de representación

- Condiciones que satisfacen los valores válidos de un registro
- Dependen del dominio del problema

Invariante de representación

- Condiciones que satisfacen los valores válidos de un registro
- Dependen del dominio del problema
- Se escriben junto a la declaración del registro

```
1  /**
2   * PROPOSITO: Posicion modela ...
3   * INVARIANTE REPRESENTACION: fila >= 0, columna >= 0
4   */
5  type Posicion is record {
6    field fila    // numero
7    field columna // numero
8  }
```

Variables y parámetros de tipo registro

- Las funciones y procedimientos pueden tener parámetros de tipo registro y retornar valores de tipo registro
  - `procedure IrAPosicion(p) {...}`
  - `function posicionActual() {...}`

- Como las variables y parámetros de tipos basicos...
  - Las variables (de tipo registro) denotan valores (de registro)
  - La operación de asignación `:=` permite recordar un valor
  - El operador relacional `==` permite comparar valores (de registro)
    - `pos := Posicion(fila <- 0, columna <- 0)`
    - `if(posicionActual() == pos) ...`

Uso de los registros

- Cada registro viene munido de funciones que permiten acceder a los valores de sus campos
- Estas funciones se llaman observadores de campos
- Hay una por cada campo y lleva el mismo nombre que el campo

Uso de los registros

- Cada registro viene munido de funciones que permiten acceder a los valores de sus campos
- Estas funciones se llaman observadores de campos
- Hay una por cada campo y lleva el mismo nombre que el campo
  - `if (fila(p) == 2) ...`
  - `f_aumentada := fuerza(conan) + 1000`
  - `if (raza(conan) == Azul) ...`
  - `while (vidas(conan) > 0) ...`
- Ejemplo completo

```
1  function superior(p, q) {
2    return (fila(p) > fila(q) ||
3            (fila(p) == fila(q) && columna(p) > columna(q))
4           )
5  }
```

Ejercicios

- Escribir un procedimiento `IrAPosicion` que, dada una posición, mueva el cabezal a dicha posición.

Ejercicios

- Escribir un procedimiento `IrAPosicion` que, dada una posición, mueva el cabezal a dicha posición.

```
1  procedure IrAPosicion(p) {
2    //IrAFila(f) e IrAColumna(c): ver practica recorridos
3    IrAFila(fila(p))
4    IrAColumna(columna(p))
5  }
```

Ejercicios

- Considere el tipo `Celda`

```
type Celda is record {
  field azul  // numero
  field negro // numero
  field rojo  // numero
  field verde // numero
}
```

- Escribir una función que codifica la celda actual como una `Celda`

Ejercicios

- Considere el tipo `Celda`

```
type Celda is record {
  field azul  // numero
  field negro // numero
  field rojo  // numero
  field verde // numero
}
```

- Escribir una función que codifica la celda actual como una `Celda`

```
function codificacionDeCelda() {
  return (Celda(azul  <- nroBolitas(Azul),
                negro <- nroBolitas(Negro),
                rojo  <- nroBolitas(Rojo),
                verde <- nroBolitas(Verde)))
}
```

Ejercicios

- Considere el siguiente registro:

```
1  /**
2   * PROPOSITO: modela un desplazamiento del cabezal
3   * (hacia el norte o el este).
4   * INVARIANTE REPRESENTACION: no hay condiciones.
5   */
6  type Desplazamiento is record {
7    field este  // numero
8    field norte // numero
9  }
```

- Escribir un procedimiento Desplazar que, dado un desplazamiento, mueva el cabezal de forma acorde.

Ejercicios

- Considere el siguiente registro:

```
1  /**
2   * PROPOSITO: modela un desplazamiento del cabezal
3   * (hacia el norte o el este).
4   * INVARIANTE REPRESENTACION: no hay condiciones.
5   */
6  type Desplazamiento is record {
7    field este  // numero
8    field norte // numero
9  }
```

- Escribir un procedimiento Desplazar que, dado un desplazamiento, mueva el cabezal de forma acorde.

```
1  procedure Desplazar(d) {
2    //fila() y columna(): ver practica recorridos
3    IrAPosicion(Posicion(fila <- fila() + norte(d),
4                         columna <- columna() + este(d)))
5  }
```

Ejercicios

- Considere el siguiente registro:

```
1  //PROPOSITO: modela una linea de ''porotochop''.
2  //INVARIANTE: longitud > 0.
3  type Linea is record {
4    field longitud    // entero
5    field color       // color
6    field orientacion // direccion
7  }
```

- Escribir un procedimiento RenderLinea que dibuje una linea

```
1  procedure RenderLinea(l) {
2    repeat(longitud(l)) {
3      Poner(color(l))
4      Mover(orientacion(l))
5    }
6    MoverN(opuesto(orientacion(l)), longitud(l))
```

Ejercicios

- Escribir una funcion `lineaRedimensionada` que, dada una linea `l` y un entero `n`, incremente la longitud `l` en `n` puntos.

Ejercicios

- Escribir una funcion `lineaRedimensionada` que, dada una linea `l` y un entero `n`, incremente la longitud `l` en `n` puntos.

```
1  function lineaRedimensionada(l, n) {
2      return (Linea(longitud <- longitud(l) + n,
3                    color <- color(l),
4                    orientacion <- orientacion(l))
5              )
6  }
```

Notación para modificar campos seleccionados

`Tipo_Registro(exp_registro | campo`$_1$ `<-` `e`$_1$ `,...,` `campo`$_n$ `<-` `e`$_n$`)`

- Denota el registro `N` que resulta de modificar solamente los campos `campo`$_1$ a `campo`$_n$ del valor de registro denotado por `exp_registro`.
- Los otros campos del `N` coinciden con los de `exp_registro`.
- Ejemplos:

```
1  function lineaRedimensionada(l, n) {
2    return(Linea(l | longitud <- longitud(l) + n))
3  }
4
5  function lineaRotada(l) {
6    return(Linea(l | orientacion <- siguiente(orientacion(l))))
7  }
```

Ejemplo completo

- ¿Qué hace el siguiente procedimiento?
- ¿Cuál es su precondición?

```
1  procedure StairwayToHeaven() {
2    IrAlInicioT(Sur, Oeste)
3    l := Linea(longitud <- 1, color <- Azul, orientacion <- Oeste)
4    while(puedeMover(Sur)) {
5      RenderLinea(l)
6      l := Linea(l | color <- siguiente(color(l)),
7                     longitud <- longitud(l) + 1)
8      Mover(Sur)
9    }
10   RenderLinea(l)
11 }
```

Registros con registros

- Un registro puede tener campos que sean a su vez registros
- En ese caso, `campo(registro)` es un registro

```
1  type Persona is record {
2    field dni  //entero
3    field edad //entero
4  }
5
6  type Alumno is record {
7    field datosPersonales //Persona
8    field legajo          //entero
9    field promedio        //entero
10 }
```

Conclusiones y lo que viene

- Registros: forma de combinar datos
- Motivaciones:
  - Describir el problema en términos del dominio
  - Organizar los datos de acuerdo a su pertinencia
- Tipos de registro, observadores de campos
- Registros con registros

Conclusiones y lo que viene

- Registros: forma de combinar datos
- Motivaciones:
  - Describir el problema en términos del dominio
  - Organizar los datos de acuerdo a su pertinencia
- Tipos de registro, observadores de campos
- Registros con registros

- La clase que viene
  - Listas con registros y registros con Listas
  - Listas de listas