Introducción a registros

Eduardo Bonelli

Tecnicatura en Programación Informática,
Universidad Nacional de Quilmes

Introducción a la Programación
Algoritmos y Programación

Hoy en Introducción a la Programación

1. Registros básicos
   - Motivación
   - Declaración
   - Tipos, valores y variables
   - Acceso a campos
   - Ejercicios
2. Registros con registros

Hoy en Introducción a la Programación

1. Registros básicos
   - Motivación
   - Declaración
   - Tipos, valores y variables
   - Acceso a campos
   - Ejercicios
2. Registros con registros

Faltan 4 semanas para el parcial

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 referencias 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 referencias 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 referencias 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 referencias con identificadores (nombres de campos)
- Objetivo: modelar entidades del dominio de aplicación
- Abstracción, reutilización

- Ejemplos:
  - Posición: `fila, columna`
  - Jugador: `puntos, vidas, movimiento`
  - Celda: `azul, negro, rojo, 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 una 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 una de los campos del registro
- Ejemplos de declaraciones:

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

```
1. type Jugador is record {
2.         field puntos
3.         field vidas
4.         field movimiento
5. }
```

Declaración de registros II

- En general, si los campos del registro son:

  $$\mathtt{c}_1, \mathtt{c}_2, \ldots, \mathtt{c}_k$$

  la declaración toma la forma siguiente:

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

Documentación de registros I

- Junto con la declaración

```
//Posición modela una celda del tablero de acuerdo a la fila
//y la columna en que se encuentra.
type Posicion is record {
        field fila // entero
        field columna // entero
}
```

Documentación de registros II

- Junto con la declaración

```
//Un jugador se modela con la cantidad de puntos y vidas actuales
//y la dirección en la que en la que se moverá en el siguiente turno.
1. type Jugador is record {
2.       field puntos // entero
3.       field vidas // entero
4.       field movimiento // direccion
5. }
```

Ejercicios

- Declarar los siguientes registros
  - Rectangulo: `base, altura`
  - Avion: `movimiento, velocidad, altura`
  - Guerrero: `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: Rojo, Norte, -1, 1, True, [ ], [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 miembros.

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 miembros.

- 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 $\leftarrow$ Azul)

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

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

donde v_i es el valor que se asocia al campo c_i

Ejemplo de uso de valores de tipo registro

1. p1 := Posicion(fila $\leftarrow$ 2, columna $\leftarrow$ 7)
2. p2 := Posicion(fila $\leftarrow$ 9, columna $\leftarrow$ -8)
3. g1 := Guerrero(fuerza $\leftarrow$ 140, vidas $\leftarrow$ 7, raza $\leftarrow$ Azul)

Invariante de representación

- Cada registro define un nuevo tipo en GOBSTONES

```
//Posición modela una celda del tablero de acuerdo a la fila
//y la columna en que se encuentra.
type Posicion is record {
        field fila // entero
        field columna // entero
}
```

- ¿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

```
   //PROPOSITO: Posición modela ...
   //INVARIANTE REPRESENTACION: fila > 0, columna > 0
1. type Posicion is record {
2.         field fila // entero
3.         field columna // entero
4. }
```

Variables y parámetros de tipo registro

- Las funciones y procedimientos pueden tener parámetros de de tipo registro y retornar valores de tipo registro
  - `procedure t.IrAPos(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)

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) ...`
  - `faumentada := fuerza(conan) + 1000`
  - `if (raza(conan) == Azul) ...`
  - `while (vidas(conan) > 0) ...`
- Ejemplo completo

```
1. function igualPosiciones(p1, p2) {
2.         return (fila(p1)==fila(p2) && columna(p1)==columna(p2))
3. }
```

Resumen

- Registro: tipo definido por el usuario.
- Campos: cada una de las variables de un registro.
- Valores de registros: combinación de los valores de sus campos.
- Invariante de representación: especifica los valores válidos.
- Variables de registro: se pueden usar como las otras variables.
- Operador punto: permite acceder a los miembros de un registro.

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.

```
procedure t.IrAPos(p) {
    //t.IrAFila(f) e t.IrAColumna(c): ver práctica 3
 1. t.IrAFila(fila(p));
 2. t.IrAColumna(columna(p));
}
```

Ejercicios

- Considere el tipo `Celda`

```
1. type Celda is record {
2.         field azul // entero
3.         field negro // entero
4.         field rojo // entero
5.         field verde // entero
6. }
```

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

Ejercicios

- Considere el tipo `Celda`

```
1. type Celda is record {
2.       field azul // entero
3.       field negro // entero
4.       field rojo // entero
5.       field verde // entero
6. }
```

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

```
function codificacionDeCelda(t) {

 1. return (Celda(azul ← nroBolitas(t,Azul),
     negro ← nroBolitas(t,Negro),
     rojo ← nroBolitas(t,Rojo),
     verde ← nroBolitas(t,Verde)))

}
```

Ejercicios

- Considere el siguiente registro:

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

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

Ejercicios

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

Ejercicios

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

```
procedure t.Desplazar(d) {
 1. pActual := mkPosActual(t)
 2. t.IrAPos(Posicion(fila ←fila(pActual)+norte(d),
    columna←columna(pActual)+este(d)))
}
```

Ejercicios

- Considere el siguiente registro:

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

- Escribir un procedimiento RenderLinea que dibuje una linea

Ejercicios

- Escribir un procedimiento RenderLinea que dibuje una linea

Ejercicios

- Escribir un procedimiento RenderLinea que dibuje una linea

```
procedure t.RenderLinea(l) {
 1. int punto = 0
 2. while(punto < longitud(l)) {
 3.     t.Poner(color(l))
 4.     t.Mover(orientacion(l))
 5.     punto := punto + 1
 6. }
 7. t.MoverN(opuesto(orientacion(l)), longitud(l))
}
```

Ejercicios

- Escribir un procedimiento redimensionarLinea que, dada una linea *l* y un entero *n*, incremente la longitud *l* en *n* puntos.

Ejercicios

- Escribir un procedimiento redimensionarLinea que, dada una linea *l* y un entero *n*, incremente la longitud *l* en *n* puntos.

```
function redimensionarLinea(l, int n) {
 1. return (Linea(longitud ← longitud(l) + n,
    color ← color(l), orientacion ← orientacion(l)))
}
```

Notación de punto para modificar y acceder a las variables de un registro

- Si x es una variable que recuerda un valor de tipo registro se puede usar la notación $x.c$ para referirse al campo c del registro
- Dado que los campos de registros son variables, $x.c$ se puede usar como cualquier variable
- Veamos un ejemplo

1. l := Linea(longitud $\leftarrow$ 1, color $\leftarrow$ Azul, orientacion $\leftarrow$ Oeste)
2. l := redimensionarLinea(l, 1)
3. l.color := siguiente(color(l))

Ejemplo completo

- ¿Qué hace el siguiente procedimiento?

```
procedure t.StairwayToHeaven() {
 1. t.IrAlExtremo(Norte); t.IrAlExtremo(Este)
 2. l := Linea(longitud←1,color←Azul,orientacion←Oeste)
 3. while(t.puedeMover(Sur)) {
 4.     t.RenderLinea(l)
 5.     l.orientacion := orientacion(l)+1
 6.     l.color := siguiente(color(l))
 7.     t.Mover(Sur)
 8. }
 9. t.RenderLinea(l)
}
```

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. }

1. type Alumno is record {
2.         field datosPersonales // Persona
3.         field legajo // entero
4.         field promedio // entero
5. }
```

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, operador punto
- 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, operador punto
- Registros con registros

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

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, operador punto
- Registros con registros

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

- FALTAN 4 semanas para el parcial