Introducción a la programación

Cecilia Manzino

Paradigmas

- Un **paradigma** es un modelo o patrón en cualquier disciplina científica.
- Un **paradigma de programación** representa un estilo de programación en cual se escriben soluciones a problemas en términos de algoritmos.

Paradigmas de lenguajes de programación

Los paradigmas de programación más comunes son:

- Imperativo
- Funcional
- Lógico
- Orientado a objetos

Hay otros: dirigido por eventos, concurrente, etc.

Programación imperativa

- Describe la programación en términos del **estado** del programa y sentencias o instrucciones que cambian dicho estado.
- El estado se modifica mediante la asignación de variables.
- El nivel de abstracción es bajo.
- La evaluación es eficiente
- Ejemplos: ASP, BASIC, C, C++, Fortran, Pascal, Java, Perl y PHP.

Ejemplo
Programa en **C**

```
#include <stdio.h>
#include <conio.h>
#include <stdlib.h>
#define TAM 10

void main(){
int a[TAM], temp, i, j;

randomize(); //Inicializa el generador de numeros aleatorios

for (i=0; i< TAM; i++)
       a[i]=random(100);

for (j=1; j <= TAM; j++) // Ordena el arreglo
   for (i=0; i< TAM-1; i++)
          if (a[i] < a[i+1]){ temp = a[i];
                              a[i] = a[i+1];
                              a[i+1] = temp;} }
```

La programación funcional

- Los programas se construyen a través de la aplicación y composición de funciones.
- No existe un estado global.
- No existen las asignaciones de variables ni las construcciones estructuradas como la secuencia o la iteración.
- El nivel de abstracción es alto.
- Se obtienen lenguajes expresivos.
- Se pueden probar propiedades sobre los programas.

Lenguajes funcionales

Satisfacen la propiedad **transparencia referencial**:

"El significado de una expresión depende únicamente de los elementos que la constituyen."

Ésto implica que:

- Cada expresión del lenguaje representa siempre el mismo valor en cualquier lugar del programa.
- Podemos razonar algebraicamente sobre los programas (muy útil para probar la corrección de los mismos).

Ejemplo
Programa en Haskell

```
quickSort []     = []
quickSotr (x:xs) = [y|y←xs, y<x] ++ [x] ++ [z|z←xs, z ≥ x]
```

Lenguajes funcionales puros e impuros

Lenguajes funcionales impuros: tienen algunos conceptos tomados de los lenguajes imperativos (como las secuencias de instrucciones o la asignación de variables).

Ejemplos: Standard ML, Scala, Lisp, Scheme, Ocaml y SAP.

Lenguajes funcionales puros: conservan la propiedad transparencia referencial, tienen mayor expresividad. Ejemplos:

Haskell y Miranda.

La programación lógica

- El objetivo es expresar programas utilizando la lógica matemática.
- Se pueden expresar formalmente problemas complejos y resolverlos mediante la aplicación de reglas, hipótesis y teoremas.
- Un programa lógico está formado por predicados.
- No existe un estado global.
- El nivel de abstracción es alto.
- Su ejecución es lenta.

La programación lógica

- La mayoría se basa en la teoría lógica de primer orden, aunque algunos incorporan comportamientos de lógicas de orden superior.
- Son muy utilizado en el área de la Inteligencia Artificial.
- Ejemplos de lenguajes lógicos: Prolog y Alice.

Ejemplo
Programa en Prolog

```
factorial(0, 1).
factorial(N, F) :- N>0, N1 is N - 1, factorial(N1, F1),
F is N*F1.

longitud([],0).
longitud([H|T],N):-longitud(T,N0), N is N0 + 1.

pertenece(X,[X|_]) :- !.
pertenece(X,[_|R]):- pertenece(X,R).
```

La programación orientada a objetos

- El universo del problema se examina en términos de entidades y relaciones entre las entidades.
- Las entidades son representadas en el programa como **objetos**.
- Los programas se expresan como un conjunto de objetos, que colaboran entre sí (envíandose **mensajes**) para realizar tareas.
- El envío de un mensaje tiene como consecuencia: un efecto o un resultado. Por ejemplo,

  ```
  7 storeOn: someFile
  7 factorial
  ```

- Los objetos se agrupan en **clases**, las cuales definen el comportamientos de los objetos de esa clase.

```
Object subClass : Human [
                | name age |
              setName : aName [ name := aName .]
              getName : [^name]
              setAge : anAge [ age := anAge .]
              getAge : [^age]
              printName [ Transcript show : 'Mi nombre es'
                                          ]

Juan := Human new
Juan setName : 'Juan Perez'
```

La programación orientada a objetos

- El nivel de abstracción es alto.
- Se obtienen programas fáciles de mantener y reutilizar.
- Los programas son no procedimentales.
- Ejemplos de lenguajes orientados a objetos: C++, Java y Smalltalk.

Paradigmas

- Ningún paradigma es mejor que otro, dependiendo de la situación un paradigma resulta más apropiado que otro.
- Existen muchos lenguajes que mezclan características de varios paradigmas.

  Por ejemplo Python es un lenguaje multiparadigma.

  El programador puede utilizar cualquier de los estilos de programación: orientado a objetos, imperativo y funcional.

expresiones cadenas de símbolos utilizados para denotar valores.
Ejemplos:

- 5, 7+2, 10 / 2
- `True`, `True || false`
- `(0.5, False)` , `(1 / 2, False && True)`

Pueden ser:

- **atómicas** (les diremos valores).
  Ejemplos: `4`, `true`, `(1,d)`.

- **compuestas**, se construyen combinando expresiones atómicas (les diremos expresiones).
  Ejemplos: `4−3`, `true || false`, `1==2`.

Tipos y expresiones bien formadas

- Los valores se agrupan en **tipos** (conjunto de valores con propiedades comunes).

  Ejemplos de tipos: `Int`, `Float`, `Bool`, `Char`, `[Int]`.

- Algunas cadenas de símbolos no forman expresiones
  - por problemas sintácticos.
    - `* 1`
    - `(, 5)`
  - o por errores de tipo
    - `1 * True`
    - `False || 'a'`

- Una expresión está "bien formada" si cumple con ciertas reglas sintácticas y de tipado.

Sintaxis de Haskell

- La definición de una variable tendrá la forma

```
nombre_variable :: Tipo
nombre_variable = expresion
```

Por ejemplo,

```
size :: Integer
size = 12 * 13
```

Funciones

Notación de tipo:

`f :: t1` $\rightarrow$ `t2` $\rightarrow \cdot \ldots \rightarrow$ `tn`

- La operación básica de una función es la aplicación.
- En Haskell la aplicación se escribe por yuxtaposición. Sea `f:: t1` $\rightarrow$ `t2`, si `x :: t1`, entonces `f x` denota el valor que se obtiene al aplicar `f` sobre `x`.
- La composición se escribe con un punto.

  Por ejemplo, si `g :: t2` $\rightarrow$ `t3`, podemos componer `f` con `g`: `f . g`

Definición de funciones

- La declaración de una función f está formada por un conjunto de ecuaciones con el formato:

```
f <pat1><pat2>. . <patn>= <expresion>
```

- Ejemplos:

```
double :: Int → Int
double x = 2 * x

fst :: (a,b) → a
fst (x,y) = x

promedio :: Float → Float → Float
promedio x y = (x+y) / 2
```

Definición de funciones con guardas

- La definición de una función puede no tratarse de una única ecuación, sino estar separada en casos.

- Ejemplo:

```
signo :: Int  →  Int
signo x | x==0 = 0
        | x>0  = 1
        | x<0  = −1
```

Identificadores

- Los identificadores de variables o funciones deben comenzar con una letra minúscula, seguido, opcionalmente por una secuencia de caracteres, cada uno de los cuales es: una letra, un dígito, un apóstrofe o un guión bajo.

- Las palabras reservadas no pueden utilizarse como nombres de funciones o variables.
  Ejemplos: `case`, `where`, `of`, `then`, `else`.