Lenguajes de Programación I

Tipos de Datos

Ernesto Hernández-Novich
<emhn@usb.ve>

Universidad "Simón Bolívar"

Tipos de Datos

Contexto para los unos y ceros

- La máquina sólo es capaz de manipular secuencias de bits – con restricciones de tamaño y velocidad particulares.
- Lenguajes introducen la noción de *tipo* para expresiones y objetos.
  - Establecen un contexto implícito de operaciones – ahorran la necesidad de expresar los "detalles".
  - Limitan el conjunto de operaciones disponibles en un programa semánticamente válido.
- Proteger al programador de escribir cosas sin sentido – hacer que el lenguaje sea a prueba de idiotas.

El problema es que no es fácil,
y los idiotas son muy astutos.

¿Para qué los quiero?

- Mejorar la legibilidad y simplificar la comprensión de los programas – una buena escogencia de tipos lleva a algoritmos directos.
- Permiten declarar la *intención de uso* para cada objeto.
  - El compilador o interpretador puede tomar mejores decisiones – tanto en código generado como en opciones de almacenamiento.
  - Permiten detectar usos incorrectos.

Sistemas de Tipos

- **Sistema de Tipos** – establece el tipo para cada valor manipulado.
- Provee mecanismos de expresión:
  - Expresar tipos intrínsecos o definir tipos nuevos.
  - Asociar los tipos definidos con construcciones del lenguaje.
- Define reglas de resolución:
  - *Equivalencia de tipos* – ¿dos valores tienen el mismo tipo?.
  - *Compatibilidad de tipos* – ¿puedo usar el tipo en este contexto?
  - *Inferencia de tipos* – ¿cuál tipo se deduce del contexto?
- Mientras más flexible el lenguaje, más complejo el sistema
  - Tipo de la referencia vs. tipo de lo referenciado.
  - ¿Cuál es el tipo de una función de primera o segunda clase?

Verificación de Tipos

Type Checking

- Proceso que garantiza que el programa obedece las reglas – detecta violaciones (*type clash*) estática o dinámicamente.
- Lenguaje de **Verificación Fuerte** (*Strongly Typed*) – impide operaciones sobre valores cuyos tipos no lo permitan.
- Lenguaje de **Verificación Fuerte Estática** (*Statically Typed*) – puede hacer *toda* la verificación a tiempo de compilación.
- Lenguaje de **Verificación Dinámica** (*Dynamically Typed*) – realiza las verificaciones a tiempo de ejecución.

¿Qué es un tipo?

. . . tenemos rato hablando de ellos

- Desde el punto de vista **Denotacional** es
  - Conjunto de valores sobre un *dominio*.
  - Algo es de un tipo si se garantiza que sus valores están en ese dominio.
- Desde el punto de vista **Constructivo** puede ser
  - *Primitivo* (*built-in* o predefinido) provisto por el lenguaje.
  - *Compuesto* (*composite* o derivado) empleando *constructores de tipos*.
- Desde el punto de vista de la **Abstracción**
  - Una *interfaz* a una representación.
  - Conjunto de operaciones con semántica bien definida y consistente.

Al programar los percibimos como
una mezcla de los tres.

Clasificación de los Tipos

- Simples – escalares, un sólo valor manipulable.
  - Predefinidos – basados en lo que puede hacer el hardware.
    - Booleanos.
    - Caracteres.
    - Numéricos.
  - Enumeración.
  - Sub-rango.
- Compuestos – colecciones, más de un valor manipulable.
  - Registros y Registros Variantes (Uniones).
  - Arreglos y Listas.
  - Conjuntos.
  - Apuntadores (Tipos Recursivos).
  - Otros.

Tipos Predefinidos

- *Booleanos* – también llamados *lógicos*.
  - `true` y `false`.
  - En algunos lenguajes no son más que un byte.
- Los *caracteres* individuales o como *cadenas*.
  - Tradicionalmente, un byte codificado en ASCII – usando los valores entre 128 y 255 como extensiones.
  - Modernamente, multibyte codificado en Unicode – UTF-8 superconjunto de ASCII para representarlos.

Tipos Numéricos

- Enteros con o sin signo (C/C++, C#).
- Números de precisión fija (*fixed point*) (Ada).
- Números racionales (Scheme, Haskell).
- Números en punto flotante (*floating point* – IEEE 754).
- Números complejos (Fortran, Lisp, Scheme, C99).
- Algunos lenguajes dejan la decisión de precisión a la implantación particular ocasionando problemas de portabilidad.

Enumeración

Números con nombre

- Un conjunto de elementos nombrados.

```
type dia = (dom, lun, mar, mie, jue, vie, sab)
```

- Los valores están ordenados haciendo válidas
  - Comparaciones, e.g. `mar < mie`.
  - Sucesor y predecesor.
  - Usarlos en una iteración

```
for hoy := lun to vie do ...
```

- Propuestos en Pascal como mecanismo para mejorar la legibilidad.
- Representados como rango consecutivo de enteros desde cero – pero son un tipo diferente incompatible con los números.
- Disponibles en C pero como un caso de uso de enteros.

```
enum dia { dom, lun, mar, mie, jue, vie, sab };
```

Sub-rango

- Conjunto de valores contiguos tomados de un tipo *padre* discreto

```
type calificacion = 1..5;
     dia_habil = lun..vie;
```

- Rangos de enteros, de caracteres, de enumeraciones o de rangos.
- Propuestos en Pascal como mecanismo para mejorar la legibilidad.
- Pueden utilizarse en una iteración

```
type minusculas = 'a'..'z';
...
for letra := 'a' to 'z' do ...
```

Registros y Registros Variantes

El COBOL que todos los lenguajes llevan por dentro

- **Registros** – colección de campos (*fields*) de tipo más simple.
- Corresponden a las tuplas de la matemática (tipos producto) – producto cartesiano de los tipos de sus campos.
- **Registros Variantes** o **Uniones** (tipos suma) – registros tales que en un momento dado sólo **uno** de sus campos es válido en un momento dado.

Arreglos y Listas

Colecciones con un orden posicional

- El arreglo es probablemente el tipo de datos compuestos más usado.
- Puede considerarse como una *función*
  - Su dominio es el tipo de los *índices*.
  - Su rango es el tipo de los *componentes*.
  - Los arreglos de caracteres suelen llamarse cadenas (*strings*) – ofrecen muchas operaciones especializadas.
- Las listas son similares, pero sin el concepto de índice – cabeza de la lista, cola de la lista, lista como un todo.
- Longitud de arreglo vs. longitud de lista.
- Listas de Asociación (parejas clave-valor) – originales de LISP, *hashes* en Perl, diccionarios en Python.

Apuntadores

- Los apuntadores son *l-values*.
- Su valor es una referencia a un objeto del tipo base del apuntador.
- Generalmente se implantan como direcciones de memoria.
- Son utilizados para construir tipos recursivos.
- Algunos lenguajes proveen mecanismos para construir tipos recursivos sin utilizar apuntadores.

Ortogonalidad

En la construcción de tipos

- ¿Puede usarse cualquier combinación de tipos en la construcción?
- ¿Cuál es el tipo de una instrucción?
- ¿Cuál es el tipo de un procedimiento que no retorna valores?
- ¿Cuál es el tipo para referirse a una función?

Bibliografía

- [*Scott*]
  - Sección 7.1
  - Ejercicios y Exploraciones
- `man unicode` y `man utf-8` – en cualquier sistema Linux decente.
- Página de Wikipedia sobre "Complemento a dos"
- Página de Wikipedia sobre IEEE Floating Point