Ayudantía 3:
Tipos de Datos

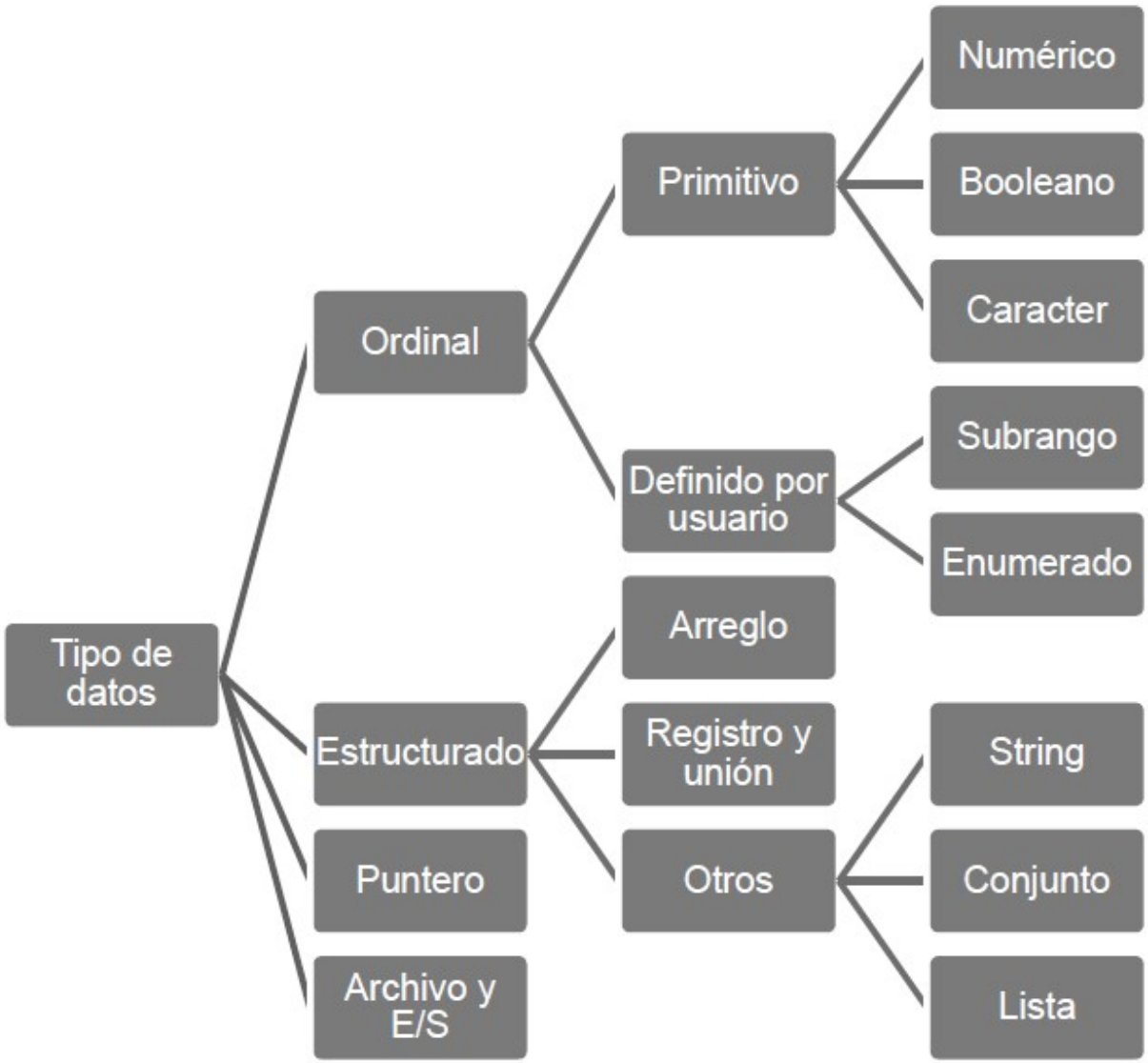
Profesores: José Luis Martí Lara, Roberto Díaz Urra
Ayudantes:
Hugo Sepúlveda Arriaza
Gabriela Acuña Benito
Lucio Fondón Rebolledo
lucio.fondon@sansano.usm.cl

Universidad Técnica Federico Santa María
Departamento de Informática

Tipos de Datos

- **Sistemas de Tipos**: Se define como la manera en que un lenguaje de programación clasifica los tipos de valores y como interactúan entre ellos.
- **Tipos de Datos**: Define conjunto de datos y además un conjunto de operaciones predefinidas para operar sobre éstos datos.
  - Primitivos
  - Estructuras de Datos
  - Tipos definidos por el usuario
  - Tipos de Datos Abstractos (TDA)

Tipos de Datos



Definiciones y Conceptos Básicos

- **Verificación de tipo (type checking)**: El lenguaje de programación se asegura de que los operandos de un operador sean compatibles.
- **Tipo compatible**: Un tipo legal, es decir, cumple con la verificación de tipo.
- **Conversión de tipos**
  - Coerción: Conversión automática de un tipo a otro.
  - Casting: Conversión definida por el programador, generalmente explícita.
- **Error de tipo**: Cuando trabajas con tipos inapropiados.

Taxonomía de los Tipos de Datos

- **Estática o Dinámica**
  - **Tipificación Estática**: Se determina el tipo de todas las variables y expresiones antes de la ejecución. No cambian durante la ejecución del código.
  - **Tipificación Dinámica**: Se determina el tipo durante la ejecución, y normalmente éste es inferido.
- **Explícita o Implícita**
  - **Tipificación Explícita**: Tipos de datos están bien declarados y definidos por el programador.
  - **Tipificación Implícita**: Tipos de datos no se declaran y generalmente se infieren mediante reglas.
- **Fuerte o Débil**
  - **Tipificación Fuerte**: Siempre se detectan errores de tipo. Establece fuertes restricciones en los tipos de datos.
  - **Tipificación Débil**: Se realizan implícitamente conversiones.

Definiciones de Tipos

- **Tipos Ordinales**: Es aquel que puede ser asociado a un número natural.
  - Primitivos
  - Definidos por el usuario
- **Estructurados**
  - Arreglos: Conjunto de variables del mismo tipo ordenadas por índice.
    - Estático
    - Dinámico Fijo de Stack
    - Dinámico de Stack
    - Fijo de Heap
    - Dinámico de Heap
  - Strings: Arreglo de carácteres
    - Estático
    - Dinámico limitado
    - Dinámico

Definiciones de Tipos

- **Estructurados**
  - Registros: Conjunto potencialmente heterogéneo de datos. Basado en otros tipos. Ej: Structs en C/C++
  - Union: Permite almacenar diferentes tipos de datos en una misma variable.
  - Archivo: Tipo especial de dato que comunica el programa con el mundo exterior (E/S).

Definiciones de Tipos

- **Estructurados**
  - Registros: Conjunto potencialmente heterogéneo de datos. Basado en otros tipos. Ej: Structs en C/C++
  - Union: Permite almacenar diferentes tipos de datos en una misma variable.
  - Archivo: Tipo especial de dato que comunica el programa con el mundo exterior (E/S).

```
// Struct
struct Data {
    int i;
    float f;
    char str[20];
};
```

```
// Union
union Data {
    int i;
    float f;
    char str[20];
};
```

Punteros

Tipo Puntero: Los valores que puede tomar un tipo puntero son **direcciones de memoria.**

- Permite mediante las variables puntero acceder a regiones de memoria dinámica
- Útil para diseñar estructuras de datos (Listas, árboles, grafos, etc.)
- **Operaciones:**
  - Asignación
  - Desreferenciación
- **Problemas:**
  - Dangling: Cuando un puntero apunta a una memoria en el heap que ya ha sido liberada.
  - Basura: Cuando se pierde la referencia de un puntero a un objeto de memoria asignada en el heap.

Ejercicios

Ejercicio 1

¿Cuál(es) de la(s) siguiente(s) afirmación(es) es(son) correcta(s)?

I. La ventaja de usar `malloc()` es la posibilidad de variar el tamaño de un arreglo durante su tiempo de vida.
II. Coerción es cuando la conversión de tipos es de manera automática.
III. Dangling se produce cuando se pierde el acceso a un objeto de memoria asignado en el heap, por no existir variables que apunten a él.
IV. Una lista enlazada basada en punteros es un tipo de datos definido por el usuario.

a) I y II
b) Sólo II
c) I, III y IV
d) II y IV

Ejercicio 1

¿Cuál(es) de la(s) siguiente(s) afirmación(es) es(son) correcta(s)?

I. La ventaja de usar `malloc()` es la posibilidad de variar el tamaño de un arreglo durante su tiempo de vida.
II. Coerción es cuando la conversión de tipos es de manera automática.
III. Dangling se produce cuando se pierde el acceso a un objeto de memoria asignado en el heap, por no existir variables que apunten a él.
IV. Una lista enlazada basada en punteros es un tipo de datos definido por el usuario.

a) I y II
b) Sólo II
c) I, III y IV
d) II y IV
R: **Alternativa b**

Ejercicio 2

```
a = 26
b = " de Abril"
print(a + b)
```

Dado el siguiente código en Python, defina la taxonomía del lenguaje (Estático vs Dinámico, Explícita vs Implícita y Tipificado Fuerte vs Débil):

Ejercicio 2

```
a = 26
b = " de Abril"
print(a + b)
```

Dado el siguiente código en Python, defina la taxonomía del lenguaje (Estático vs Dinámico, Explícita vs Implícita y Tipificado Fuerte vs Débil): R:

```
Traceback (most recent call last):  File "test.py", line 6, in
<module> print a + b TypeError:  unsupported operand type(s)
for +:  'int' and 'str'
```

- **Dinámico**
- **Implícito**
- **Fuerte**

Ejercicio 3

```
1  int main(){
2      char miChar = "a";
3      miChar++;
4      printf("%d == %d", 99, miChar + 1);
5      return 0;
6  }
```

Dado el siguiente código en C, defina la taxonomía del lenguaje (Estático vs Dinámico, Explícita vs Implícita y Tipificado Fuerte vs Débil):

Ejercicio 3

```
1  int main(){
2      char miChar = "a";
3      miChar++;
4      printf("%d == %d", 99, miChar + 1);
5      return 0;
6  }
```

Dado el siguiente código en C, defina la taxonomía del lenguaje (Estático vs Dinámico, Explícita vs Implícita y Tipificado Fuerte vs Débil): R:

Output:

```
99 == 99
```

- **Estático**
- **Explícito**
- **Débil**

Ejercicio 4

```
1  int *algopasa(int i){
2      int x = i;
3      return &x;
4  }
5
6  int main(){
7      int algo = 1234;
8      int *puntero;
9      puntero = algopasa(algo);
10     printf("%d\n", *puntero);
11     return 0;
12 }
```

Explicar qué problema produce el siguiente programa.

Ejercicio 4

```
1  int *algopasa(int i){
2      int x = i;
3      return &x;
4  }
5
6  int main(){
7      int algo = 1234;
8      int *puntero;
9      puntero = algopasa(algo);
10     printf("%d\n", *puntero);
11     return 0;
12 }
```

Explicar qué problema produce el siguiente programa.

R: Produce dangling