Práctica 7 Sistemas y tipos de Datos

> **Objetivo** Comprender las nociones fundamentales sobre las diversas propiedades de los sistemas de tipos y los tipos de datos

Ejercicio 1 Sistemas de tipos

a) ¿Qué es un sistema de tipos y cuál es su principal función?

- Un sistema de tipos es un conjunto de reglas que clasifica y manipula los datos en un programa. Podemos definir a un tipo como un conjunto de valores y un conjunto de operaciones que se pueden utilizar para manipularlos.
- Su función principal es garantizar la integridad de los datos, prevenir errores y facilitar el mantenimiento del código.

b) Definir y contrastar las definiciones de un sistema de tipos fuerte y débil

> probablemente en la bibliografía se encuentren dos definiciones posibles. Volcar ambas en la respuesta

Un sistema de tipos `fuerte` es aquel en el que se aplican restricciones estrictas sobre las operaciones y conversiones entre diferentes tipos de datos. En este sistema, se garantiza que las operaciones se realicen únicamente entre tipos compatibles y se previenen las conversiones automáticas o implícitas entre tipos que no sean directamente compatibles.

Ejemplificar con al menos 2 lenguajes para cada uno de ellos y justificar

Python, es un lenguaje con un sistema de tipos fuerte, lo que significa que no se permiten conversiones automáticas entre tipos incompatibles. Por ejemplo, si intentamos sumar una cadena de texto con un número, Python generará un error, ya que no se permite la concatenación de una cadena y un número sin una conversión explícita.

```python
a = "Hola"
b = 5
c = a + b  # Esto generará un error
```

Otro ejemplo puede ser en Java, las conversiones entre tipos deben ser explícitas y se verifican rigurosamente. Si intentamos asignar un tipo de dato incompatible a una variable, el compilador generará un error.

```java
String a = "Hola";
int b = 5;
String c = a + b;  // Esto generará un error
```

En ambos ejemplos, Python y Java no permiten la realización de operaciones entre tipos incompatibles sin una conversión explícita, lo que demuestra su enfoque en la rigurosidad y seguridad del sistema de tipos.

Un sistema de tipos `débil` , por otro lado, permite conversiones implícitas y automáticas entre diferentes tipos de datos, incluso cuando no son directamente compatibles. En este sistema, las operaciones pueden realizarse entre tipos diferentes sin requerir una conversión explícita.

Un ejemplo es JavaScript, permite realizar operaciones entre tipos diferentes y realiza conversiones automáticas para

PHP también se considera un lenguaje con un sistema de tipos débil. Al igual que JavaScript, PHP realiza

adaptar los tipos. Por ejemplo, JavaScript permite sumar una cadena de texto con un número, convirtiendo automáticamente el número en una cadena.

```
var a = "Hola";
var b = 5;
var c = a + b; // El resultado será la cadena "Hola5"
```

conversiones automáticas entre diferentes tipos de datos. Por ejemplo, en PHP, se puede sumar una cadena de texto y un número sin una conversión explícita.

```
$a = "Hola";
$b = 5;
$c = $a + $b;  // El resultado será el número 5
```

En estos ejemplos de JavaScript y PHP, se permite la realización de operaciones entre tipos diferentes sin una conversión explícita, lo que demuestra su enfoque más flexible y menos riguroso en el sistema de tipos.

c) Además de la clasificación anterior, también es posible caracterizar el tipado como estático o dinámico. ¿Qué significa esto?

El tipado **`estático`** realiza verificaciones de tipos en tiempo de compilación, mientras que el tipado **`dinámico`** realiza verificaciones de tipos en tiempo de ejecución. El tipado estático proporciona seguridad en la detección temprana de errores de tipo, mientras que el tipado dinámico ofrece flexibilidad en la manipulación de tipos durante la ejecución del programa.

Ejemplificar con al menos 2 lenguajes para cada uno de ellos y justificar

Ejemplos de lenguajes con tipado estático son:

C es un lenguaje con tipado estático. Antes de compilar un programa en C, se deben declarar y asignar tipos a todas las variables utilizadas. Las verificaciones de tipos se realizan en tiempo de compilación, lo que significa que se detectarán errores de tipos antes de la ejecución del programa.

```
#include <stdio.h>

int main() {
   int a = 5;
   float b = 2.5;
   int c = a + b;  // Esto generará un error de compilación
   printf("%d", c);
   return 0;
}
```

Java también es conocido por tener un tipado estático. verificaciones de tipos se realizan en tiempo de compila requiere la declaración y asignación de tipos a todas las expresiones. El compilador de Java verifica la coherenc en el código antes de generar el bytecode ejecutable.

```
public class Main {
   public static void main(String[] args) {
      int a = 5;
      float b = 2.5;
      int c = a + b;  // Esto generará un error de
      System.out.println(c);
   }
}
```

Ejemplos de lenguajes con tipado dinámico son:

Python es un lenguaje con tipado dinámico. Las variables en Python pueden tomar diferentes tipos en tiempo de ejecución y las operaciones se verifican en tiempo de ejecución. No se requiere una declaración de tipo explícita para las variables.

```
a = 5
b = 2.5
c = a + b  # El resultado será 7.5 (float)
print(c)
```

JavaScript también es un lenguaje con tipado dinámico. Las variables en JavaScript pueden cambiar de tipo en tiempo de ejecución, y las operaciones se verifican en tiempo de ejecución.

```
var a = 5;
var b = 2.5;
var c = a + b; // El resultado será 7.5 (number)
console.log(c);
```

Ejercicio 2 Tipos de datos

a) Dar una definición de tipo de dato

Un tipo de dato es una categoría o clasificación que define la naturaleza y el comportamiento de un valor en un lenguaje de programación.

¿Qué es un tipo predefinido elemental? Dar ejemplos.

Cualquier lenguaje de programación viene con un conjunto de tipos llamados **`predefinidos`** , quienes reflejan el comportamiento de hardware que tienen debajo y son una abstracción de él.

Sus ventajas son la invisibilidad de la representación, verificación estática (el lenguaje solo verifica que se cumplan las reglas del tipo), desambiguar operadores y control de precisión. Por ejemplo: números (enteros), caracteres y booleans.

¿Qué es un tipo definido por el usuario? Dar ejemplos

Los lenguajes de programación permiten especificar agrupaciones de objetos de datos elementales o predefinidos, y de forma recursiva agregaciones de agregados. Esto se logra mediante la prestación de una serie de constructores que permiten definir lo que denominamos tipo de dato `definido por el usuario` . Separan la especificación (**crear**) de la implementación (**usar**).

Se definen los tipos que el problema necesita: definir nuevos tipos o instanciarlos y chequeo de consistencia. Dos ejemplos son:

Clases en Java: En Java, se pueden crear clases para definir tipos de objetos personalizados. Por ejemplo, se podría crear una clase "Persona" con propiedades como nombre, edad y dirección.

```
class Persona {
   String nombre;
   int edad;
   String direccion;
}

Persona persona1 = new Persona();
persona1.nombre = "Juan";
persona1.edad = 30;
persona1.direccion = "Calle Principal";
```

Estructuras en C: En C, se pueden definir estructuras para agrupar diferentes tipos de datos en una sola entidad. Por ejemplo, se podría crear una estructura "Punto" con coordenadas x e y.

```
struct Punto {
   int x;
   int y;
};

struct Punto punto1;
punto1.x = 2;
punto1.y = 5;
```

Ejercicio 3: Tipos compuestos

a) Dar una breve definición de: producto cartesiano, correspondencia finita, uniones y tipos recursivos.

Los compuestos son construcciones a partir de tipos predefinidos:

- **Producto cartesiano**: es un **conjunto cuyos elementos están ordenados n-tupla,** es decir, es una construcción en teoría de conjuntos y programación que combina dos conjuntos o tipos de datos para formar un nuevo conjunto o tipo cuyos elementos contienen una combinación de elementos de ambos conjuntos. En programación, esto puede representarse mediante una estructura de datos que contiene múltiples campos o propiedades**.** Por ejemplo, los registros. Es una relación 1 a muchos.
- **Correspondencia finita**: La correspondencia finita se refiere a una relación uno a uno entre los elementos de dos conjuntos finitos. Para cada elemento en el primer conjunto, hay exactamente un elemento correspondiente en el segundo conjunto, y viceversa. Esta correspondencia puede ser representada mediante una función que asigna cada elemento del primer conjunto a un único elemento del segundo conjunto. El tipo de dato serían los arreglos.

- **Uniones**: Las uniones, también conocidas como sum type o tipo suma, son una construcción en programación que permite combinar varios tipos de datos en uno solo. En una unión, un valor puede pertenecer a uno de los tipos dentro de la unión. Esto se puede utilizar para representar alternativas o opciones donde un valor puede ser de diferentes tipos.
- **Tipos recursivos**: Los tipos recursivos son aquellos que se definen en términos de sí mismos. Esto significa que un tipo puede contener instancias de sí mismo como parte de su estructura. Los tipos recursivos son útiles para modelar estructuras de datos que contienen referencias a sí mismas, como árboles o listas enlazadas. Esta recursividad permite la construcción de estructuras de datos complejas y anidadas.

b) Identificar a qué clase de tipo de datos pertenecen los siguientes extractos de código. En algunos casos puede corresponder más de una

- **Java**: Producto cartesiano.
- **C 1**: Recursivo y producto cartesiano.
- **C 2**: Unión
- **Ruby**: Correspondencia finita
- **PHP**: ??? correspondencia finita **// CONSULTAR**
- **Python**: Producto cartesiano
- **Haskell 1**: Recursivo y producto cartesiano.
- **Haskell 1**: Unión

Java	C	C

`class Persona {`
`  String nombre;`
`  String apellido;`
`  int edad;`
`}`
	`typedef struct _nodoLista {`
	`  void *dato;`
	`  struct _nodoLista *siguiente`
	`} nodoLista;`
	`typedef struct _lista {`
	`  int cantidad;`
	`  nodoLista *primero`
	`} Lista;`
		`union codigo {`
		`  int numero;`
		`  char id;`
		`};`

Ruby	**PHP**	**Python**

`hash = {`
`  uno: 1,`
`  dos: 2,`
`  tres: 3,`
`  cuatro: 4`
`}`
	`function doble($x) {`
	`  return 2 * $x;`
	`}`
		`tuple = (`
		`  'physics',`
		`  'chemistry',`
		`  1997, 2000`
		`  )`

Hashell	**Hashell**	

`data ArbolBinarioInt =`
`Nil |`
`Nodo int`
`(ArbolBinarioInt dato)`
`(ArbolBinarioInt dato)`
	`data Color =`
	`  Rojo |`
	`  Verde |`
	`  Azul`	

> ### Ayuda para interpretar
>
> - `ArbolBinarioInt` es un tipo de dato que puede ser Nil (“vacío”) o un Nodo con un dato número entero (int) junto a un árbol como hijo izquierdo y otro árbol como hijo derech
> - `Color` es un tipo de dato que puede ser Rojo, Verde o Azul

Ejercicio 4: Mutabilidad/Inmutabilidad

a) Definir mutabilidad e inmutabilidad respecto a un dato.

1. Mutabilidad e inmutabilidad son conceptos relacionados con la capacidad de cambiar o modificar un dato una vez que ha sido creado.

- **La `mutabilidad`** se refiere a la capacidad de un dato de ser modificado después de su creación. En otras palabras, un dato mutable puede ser alterado en su contenido o estado sin necesidad de crear un nuevo dato. Esto implica que las referencias a ese dato también reflejarán los cambios realizados.
- **La `inmutabilidad`** se refiere a la incapacidad de un dato de ser modificado después de su creación. Un dato inmutable no puede ser alterado, lo que significa que cualquier operación que parezca modificarlo en realidad crea un nuevo dato con los cambios deseados.

Dar ejemplos en al menos 2 lenguajes.

> TIP: indagar sobre los tipos de datos que ofrece Python y sobre la operación

Lista mutable en Python	Cadena inmutable en Ruby

`lista = [1, 2, 3]`
`lista.append(4)`
`# Modifica la lista existente`
`print(lista)`
`# Salida: [1, 2, 3, 4]`
	`cadena = "Hola"`
	`nueva_cadena = cadena.upcase`
	`# Crea una nueva cadena modificada`
	`puts cadena`
	`# Salida: "Hola"`
	`puts nueva_cadena`
	`# Salida: "HOLA"`

Python ofrece una amplia gama de tipos de datos inmutables, como enteros, flotantes, cadenas, tuplas y frozensets. En Python, los datos inmutables no pueden ser modificados después de su creación. Por ejemplo, al intentar modificar una cadena, se crea una nueva cadena con los cambios deseados.

Ruby también tiene tipos de datos inmutables, como símbolos y cadenas. En Ruby, se puede utilizar el método **`freeze`** para hacer que un objeto sea inmutable, lo que impide cualquier modificación posterior.

```
# Objeto inmutable en Ruby utilizando el método freeze
cadena = "Hola".freeze
cadena << " Mundo"  # Generará un error de tiempo de ejecución
puts cadena
```

En resumen, la mutabilidad se refiere a la capacidad de un dato de ser modificado después de su creación, mientras que la inmutabilidad se refiere a la incapacidad de un dato de ser modificado. Python ofrece varios tipos de datos inmutables, y Ruby proporciona el método **`freeze`** para hacer que los objetos sean inmutables.

b) Dado el siguiente código

```
a = Dato.new(1)
a = Dato.new(2)
```

¿Se puede afirmar entonces que el objeto “Dato.new(1)” es mutable? Justificar la respuesta sea por afirmativa o por la negativa.

No se puede afirmar nada ya que nunca se modifica el dato, sólo se crean nuevos.

Ejercicio 5: Manejo de punteros

a) ¿Permite C tomar el l-valor de las variables? Ejemplificar.

Si, es posible tomarlo. Esto se realiza con el operador `&` para poder tomar el l-valor. En este ejemplo, se declara una variable "numero" de tipo entero y se inicializa con el valor 10. Luego, se declara un puntero "puntero" de tipo entero y se le asigna el l-valor de la variable "numero" utilizando el operador "&".

```c
#include <stdio.h>

int main() {
   int numero = 10;
   int* puntero = &numero;

   printf("Valor: %d\n", numero);
   printf("Dirección: %p\n", &numero);
   printf("L-valor: %p\n", puntero);

   return 0;
}
```

b) ¿Qué problemas existen en el manejo de punteros? Ejemplificar.

Los punteros son mecanismos muy potentes para generar estructuras recursivas. Por acceder bajo nivel pueden hacer inseguros a los programas que los usan. Las inseguridades del los punteros son:

- **Violación de tipos**: puede que me pare en una dirección y opere sobre lo que contiene esa dirección de forma tal que no cumpla con las operaciones para ese tipo. Se dice que se viola el tipo al que apunta el puntero.
- **Referencias sueltas**: el objeto al que apuntaba el puntero ya no existe, el puntero apunta una variable que fue desalocada. Si el objeto no está alocado se dice que el puntero es peligroso (**dangling**). Si luego se usa el puntero se producirá error.
- **Punteros no inicializados** (puntero = null): Peligro de acceso descontrolado a posiciones de memoria, verificación dinámica de la inicialización. La solución es un valor especial nulo:
  - nil en Pascal
  - void en C/C++
  - null en ADA, Python
- **Punteros y Uniones discriminadas**
- **Alias**: varios punteros apuntando al mismo objeto. Esto provoca que cuando uno cambie el valor todos lo verán reflejado y no es correcto en términos de verificabilidad. Lo mismo si desaloco una variable, los punteros no apuntarán correctamente.
- **Liberación de memoria**: los objetos apuntados que se alocan a través de la primitiva **new** son alocados en la heap, la memoria disponible de heap podría agotarse a menos que de alguna forma se devuelva el almacenamiento alocado liberado. Es decir, un puntero que ya no se utiliza no fue liberado así que ocupa memoria al pedo. Un objeto se dice accesible si alguna variable en la pila apunta directa o indirectamente a ese objeto, un objeto es basura si no es accesible.

Ejercicio 6: TAD

a) ¿Qué características debe cumplir una unidad para que sea un TAD?

Un tipo abstracto de datos es un conjunto de valores y de operaciones definidos mediante una especificación independiente de cualquier representación. Para que una unidad sea considerada un TAD, debe cumplir con características como el encapsulamiento, la abstracción, la modularidad, la independencia de la implementación y tener su propia identidad y conjunto de operaciones definidos. Estas características permiten una gestión más eficiente y comprensión de los datos, facilitando el diseño y la implementación de sistemas de software.

b) Dar algunos ejemplos de TAD en lenguajes tales como ADA, Java, Python, entre otros.

Cada lenguaje tiene su propia implementación de estructuras de datos y TAD, pero todos comparten la idea de proporcionar una abstracción de datos con operaciones definidas para facilitar el manejo de datos en los programas.

Lenguaje	TAD	Descripción	Operaciones típicas
ADA:	Stack (Pila)	Un TAD que representa una estructura de datos LIFO (Last In, First Out), donde los elementos se agregan y eliminan desde el tope de la pila.	Push (agregar elemento), Pop (eliminar elemento), Is_Empty (verificar si está vacía), Top (obtener el elemento del tope).
Java:	ArrayList	Un TAD que representa una lista dinámica de elementos en Java. Proporciona operaciones para agregar, eliminar, buscar y acceder a elementos en función de su índice.	add (agregar elemento), remove (eliminar elemento), get (obtener elemento), size (obtener tamaño de la lista).
Python:	Queue (Cola)	Un TAD que representa una estructura de datos FIFO (First In, First Out), donde los elementos se agregan al final y se eliminan del frente de la cola.	enqueue (agregar elemento), dequeue (eliminar elemento), is_empty (verificar si está vacía), front (obtener el elemento del frente).
C++:	Set (Conjunto)	Un TAD que representa una colección de elementos únicos, donde no se permite la duplicación de valores.	insert (insertar elemento), erase (eliminar elemento), find (buscar elemento), size (obtener tamaño del conjunto).
Ruby:	Hash (Tabla de Hash)	Un TAD que representa una estructura de datos asociativa, donde se almacenan pares clave-valor. Permite buscar, insertar y eliminar elementos de manera eficiente.	store (almacenar elemento con una clave), fetch (obtener elemento por clave), delete (eliminar elemento por clave), size (obtener tamaño del hash).