Práctica 7

Ejercicio 1: Sistemas de tipos:

1. **¿Qué es un sistema de tipos y cuál es su principal función?**

   Conjunto de reglas usadas por un lenguaje para estructurar y organizar sus tipos. El objetivo de un sistema de tipos es escribir programas seguros.

   Funciones:
   - 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

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

   TIPADO FUERTE – TIPADO DÉBIL Se dice que el sistema de tipos es fuerte cuando especifica restricciones sobre como las operaciones que involucran valores de diferentes tipos pueden operarse. Lo contrario establece un sistema débil de tipos.

   Tipado débil → C
   Tipado fuerte → Python

3. **Además de la clasificación anterior, también es posible caracterizar el tipado como estático o dinámico. ¿Qué significa esto? Ejemplificar con al menos 2 lenguajes para cada uno de ellos y justificar.**

   Tipado estático: ligaduras en compilación.
   Para esto puede exigir:
   - Se puedan utilizar tipos de datos predefinidos
   - Todas las variables se declaran con un tipo asociado
   - Todas las operaciones se especifican indicando los tipos de los operandos requeridos y el tipo del resultado.

   Tipado dinámico: ligaduras en ejecución, provoca más comprobaciones en tiempo de ejecución

   Tipado estático → C
   Tipado dinámico → Python

Ejercicio 2: Tipos de datos:

1. **Dar una definición de tipo de dato.**

   Podemos definir a un tipo como un conjunto de valores y un conjunto de operaciones que se pueden utilizar para manipularlos.

2. **¿Qué es un tipo predefinido elemental? Dar ejemplos.**

   Reflejan el comportamiento del hardware de abajo y son una abstracción de él.

   Ejemplos: enteros, reales, caracteres, booleanos, strings

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

   Los lenguajes de programación permiten al programador especificar agrupaciones de objetos de datos elementales y de forma recursiva, agregaciones de agregados

   Ejemplos: arreglos, listas, registros, enumerados.

Ejercicio 3: Tipos compuestos:

1. **Dar una breve definición de: producto cartesiano (en la bibliografía puede aparecer también como product type), correspondencia finita, uniones (en la bibliografía puede aparecer también como sum type) y tipos recursivos.**

   - Producto Cartesiano: es el producto entre conjuntos de diferentes tipos. Produce registros (Pascal) o struct (C).
   - Correspondencia Finita: un conjunto de valores en un conjunto de dominio. Son los vectores.
     - Es una función de un conjunto finito de valores de un tipo de dominio DT en valores de un tipo de dominio RT

     - DT-> tipo de dominio (int por ej)
     - RT-> resultado del dominio (acceso a través del índice)
   - Unión y unión discriminada: la unión de uno o más tipos, es la disyunción de los tipos dados. Se trata de campos mutuamente excluyente (uso uno o el otro), no pueden estar al mismo tiempo con valores.
     - La unión discriminada se agrega un descriptor (enumerativo) que me permite saber con quién estoy trabajando y acceder correctamente a lo

que tengo que acceder. Básicamente manipulo el elemento según el valor del discriminante, es una mejora de la unión.
    - La unión es insegura.
    - El chequeo de tipos debe hacerse en ejecución
- Recursión: un tipo de dato recursivo T se define como una estructura que puede contener componentes de tipo T. Un ejemplo son las listas.
    - Define datos agrupados
        - Cuyo tamaño puede crecer arbitrariamente
        - Cuya estructura puede ser arbitrariamente compleja
    - Los lenguajes soportan la implementación de tipos de datos recursivos a través de los punteros

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

Java

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

Producto cartesiano

C

```
typedef struct _nodoLista {
  void *dato;
  struct _nodoLista *siguiente
} nodoLista;

typedef struct _lista {
  int cantidad;
  nodoLista *primero
} Lista;
```

Producto cartesiano y recursión

C

```
union codigo {
  int numero;
  char id;
};
```

Unión

Ruby correspondencia

```
hash = {
  uno: 1,
  dos: 2,
  tres: 3,
  cuatro: 4
}
```

PHP

```
function doble($x) {
  return 2 * $x;
}
```

Python

```
tuple = ('physics',
'chemistry', 1997, 2000)
```

Correspondencia finita

Haskell

```
data ArbolBinarioInt =
  Nil |
  Nodo int
    (ArbolBinarioInt dato)
    (ArbolBinarioInt dato)
```

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 derecho

Recursión

Haskell

```
data Color =
  Rojo |
  Verde |
  Azul
```

Ayuda para interpretar:
'Color' es un tipo de dato que puede ser Rojo, Verde o Azul.

Unión

Ejercicio 4: Mutabilidad/Inmutabilidad:

1. **Definir mutabilidad e inmutabilidad respecto a un dato. Dar ejemplos en al menos 2 lenguajes. TIP: indagar sobre los tipos de datos que ofrece Python y sobre la operación #freeze en los objetos de Ruby.**

   Mutabilidad e inmutabilidad son términos que se refieren a la capacidad o no de un dato de ser modificado después de su creación. Un dato mutable es aquel que se puede cambiar después de haber sido creado, mientras que un dato inmutable es aquel que no se puede cambiar después de haber sido creado.

   En Python, algunos ejemplos de datos inmutables son los enteros, las cadenas y las tuplas. Por ejemplo, si creamos una cadena en Python, no podemos cambiar su contenido, solo podemos crear una nueva cadena que contenga los cambios que deseamos. Por otro lado, los datos mutables en Python incluyen listas, conjuntos y diccionarios, ya que se pueden modificar después de haber sido creados.

```python
# Ejemplo de datos inmutables en Python
a = 5         # Entero
b = "Hola"    # Cadena
c = (1, 2, 3)    # Tupla

# No se puede modificar una cadena
# La siguiente línea producirá un error de tipo TypeError
b[0] = "h"
```

   En Ruby, la mayoría de los objetos son mutables por defecto, pero se puede hacer que un objeto sea inmutable utilizando el método freeze. Después de que un objeto es congelado, no se puede modificar.

```ruby
# Ejemplo de datos mutables e inmutables en Ruby
str = "Hola"    # String mutable
arr = [1, 2, 3]    # Array mutable

# Congelar un objeto lo vuelve inmutable
str.freeze
arr.freeze

# No se puede modificar una cadena congelada
# La siguiente línea producirá un error de tipo RuntimeError
str[0] = "h"
```

2. **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 que el objeto "Dato.new(1)" sea mutable solo con la información proporcionada en el código.

El hecho de que se cree una nueva instancia de Dato y se asigne a la variable a después de crear una instancia con el valor 1 no es suficiente para determinar si el objeto Dato.new(1) es mutable o inmutable. Si se puede modificar el valor de la instancia Dato.new(1) después de su creación, entonces el objeto es mutable, pero si no se puede modificar después de su creación, entonces el objeto es inmutable.

Para determinar la mutabilidad del objeto Dato.new(1) se necesitaría más información sobre la clase Dato y cómo está implementada. Si la clase Dato define métodos que permiten cambiar el valor de la instancia después de su creación, entonces el objeto es mutable, pero si la clase Dato define la instancia como inmutable, entonces el objeto es inmutable.

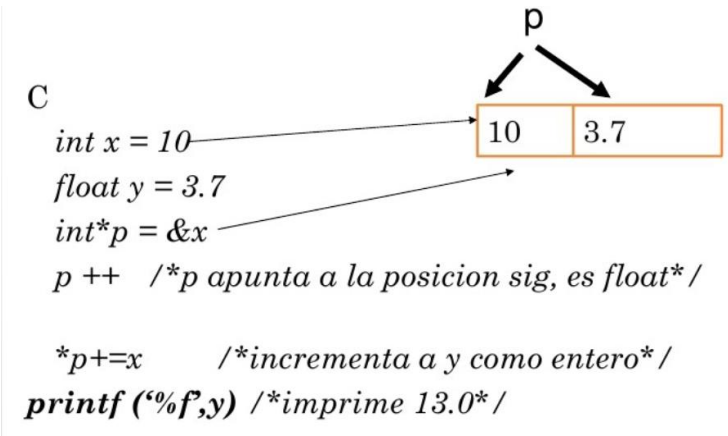
Ejercicio 5: Manejo de punteros:

1. **¿Permite C tomar el l-valor de las variables? Ejemplificar.**

   Sí, C permite tomar el l-valor (dirección de memoria) de las variables utilizando el operador "&".

2. **¿Qué problemas existen en el manejo de punteros? Ejemplificar.**

   1. Violación de tipos
      - la imagen explica mejor



   2. Referencias sueltas – referencias dangling
      - Una referencia suelta o dangling es un puntero que contiene una dirección de una variable dinámica que fue desalocada, si luego se usa el puntero producirá error

   3. Punteros no inicializados
      - Peligro de acceso descontrolado a posiciones de memoria

- Soluciones:
  - Nil en pascal
  - Void en C/C++
  - Null en ADA y Python

4. Punteros y uniones discriminadas

```
union ojo{
int int_var
int* int_ref}
```

   - Me permite acceder a algo que no debo
   - En el caso de C, este es el mismo efecto que causa la aritmética de punteros. Para resolver este problema asociado con los punteros Java elimina la noción de puntero explicito completamente.

5. Alias
   -

```
int* p1
int* p2
int x
p1 = &x
p2 = &x
```

```
p1 y p2 son punteros
p1 y x son alias
p2 y x también lo son
```

   - Si uno cambia su valor se ve reflejado en el otro. Sucede lo mismo si libre y esto genera el punto 2.

6. Liberación de memoria: objetos perdidos
   - Las variables puntero se alocan como cualquier otra variable en la pila de registros de activación
     - Los objetos apuntados que se alocan a través de la primitiva new son alocados en la heap
     - La memoria disponible (heap) podría agotarse a menos que de alguna forma se devuelva el almacenamiento alocado liberado

Ejercicio 6: TAD :

1. **¿Qué características debe cumplir una unidad para que sea un TAD?**

Tipo abstracto de dato (TAD) es el que satisface:

- Encapsulamiento: la representación del tipo y las operaciones permitidas para los objetos del tipo se describen en una única unidad sintáctica. Refleja las abstracciones descubiertas en el diseño

- Ocultamiento de la información: la representación de los objetos y la implementación del tipo permanecen ocultos. Refleja los niveles de abstracción. Modificabilidad

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

   - Java: ArrayList, LinkedList, HashMap, TreeSet, PriorityQueue.
   - Python: list, tuple, set, dictionary, deque.
   - C++: std::vector, std::map, std::queue, std::stack.
   - ADA: Ada.Containers.Vectors, Ada.Containers.Hash_Tables, Ada.Containers.Linked_Lists, Ada.Containers.Priority_Queue.