Conceptos y Paradigmas de lenguajes de Programación
2024

Práctica Nro. 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:

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

Ejercicio 2: Tipos de datos:

1. Dar una definición de tipo de dato.
2. ¿Qué es un tipo predefinido elemental? Dar ejemplos.
3. ¿Qué es un tipo definido por el usuario? Dar ejemplos.

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.
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;
}
```

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

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

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

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

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

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

Haskell	**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
	`data Color =`
	`  Rojo |`
	`  Verde |`
	`  Azul`
	
	Ayuda para interpretar:
	'Color' es un tipo de dato que puede ser Rojo, Verde o Azul.	

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.*
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.

Ejercicio 5: Manejo de punteros:

1. ¿Permite C tomar el l-valor de las variables? Ejemplificar.
2. ¿Qué problemas existen en el manejo de punteros? Ejemplificar.

Ejercicio 6: TAD :

1. ¿Qué características debe cumplir una unidad para que sea un TAD?
2. Dar algunos ejemplos de TAD en lenguajes tales como ADA, Java, Python, entre otros.

Ejercicio 1

Inciso 1.

- Conjunto de reglas que usa un lenguaje para estructurar y organizar sus tipos.
- **El objetivo de un sistema de tipos es escribir programas seguros.**
- Conocer estos sistemas nos permite conocer mejor los aspectos semánticos de los diversos lenguajes.
- **Funciones**
    - **Provee Mecanismos De Expresión:**
        - Expresar tipos predefinidos, definir nuevos tipos y asociarlos con constructores del lenguaje.
    - **Define Reglas De Resolución:**
        - Equivalencia de tipos → ¿Dos valores tienen el mismo tipo?.
        - Compatibilidad de tipos → ¿Puede usarse el tipo en este contexto?.
        - Inferencia de tipos → ¿Cuál tipo se deduce del contexto?
    - **Mientras más flexible en el tipado sea el lenguaje, más complejo será su sistema de tipos.**

Inciso 2.

- Se dice que el **sistema de tipos es fuerte** cuando especifica **restricciones** de forma clara sobre cómo las **operaciones** que involucran valores de **diferentes tipos** pueden **operarse**. Lo contrario establece un **sistema débil de tipos**.
- **Un Sistema Débil** es **menos seguro** pero ofrece una **mayor flexibilidad.**
- **Un Sistema Fuerte** es **más seguro** pero ofrece una **menor flexibilidad**. El compilador asegura la detección de todos los errores de tipos y la ausencia de estos en los programas.
- **Ejemplos**
    - Python es Fuertemente Tipado.
    - C es débilmente Tipado.
    - GOBSTONE es Fuertemente Tipado.

Inciso 3.

- **Tipos Ligadura**
    - **Tipado Estático:** Ligaduras en compilación. Puede exigir lo siguiente
        - Se pueden 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 tiempo de ejecución. Que las ligaduras se den en ejecución no vuelve a este tipado un tipado inseguro.
    - **Tipado Seguro**: No Es Estático, ni inseguro.
- **Ejemplos**
    - Python tiene tipado dinámico.
    - C tiene tipado estático.
    - GOBSTONE tiene tipado estático.

Ejercicio 2

Inciso 1.

- Un tipo de datos es un conjunto de valores y un conjunto de operaciones asociadas al tipo para manejar esos valores.

Inciso 2.

- Un tipo de dato predefinido refleja el comportamiento del hardware subyacente y es una abstracción de él. En particular los elementales/escalares son tipos de datos predefinidos indivisibles, es decir, estos no se pueden descomponer a partir de otros.
- **Ejemplos**
    - Enteros.
    - Reales.

    - Caracteres.
    - Booleanos.

Inciso 3.

- Los lenguajes de programación permiten al programador especificar agrupaciones de objetos de datos elementales y de forma recursiva, agregaciones de agregados. Esto se logra mediante constructores. A estas agrupaciones se las conoce como tipos de datos definidos por el usuario.
- Ejemplos
    - Enumerados.
    - Arreglos.
    - Registros.
    - Listas.

Ejercicio 3

Inciso 1.

- **Producto Cartesiano**
    - El Producto cartesiano de n conjuntos de tipos variados. Permite producir registros (Pascal) o struct (C).
- **Correspondencia Finita**
    - Es una función 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).
    - Son Las Listas indexadas, vectores, arreglos, matrices, etc.
- **Unión y Unión Discriminada**
    - La Unión/unión discriminada de uno o más tipos, es la disyunción de los tipos dados. Se trata de campos mutuamente excluyentes (uso uno o el otro), no pueden estar al mismo tiempo con valores.
    - Permite manipular diferentes tipos en distintos momentos de la ejecución.
    - Chequeo Dinámico. No se puede asegurar en compilación qué tipo o variante adquiere una variable.
    - La Unión Discriminada agrega un descriptor (enumerativo) que me permite saber con quien estoy trabajando y acceder correctamente a lo que tengo que acceder, ya que nos dice cual de los campos posee valor. Básicamente manipulo el elemento según el valor del discriminante, es una mejora de la unión que brinda una mayor seguridad. Este discriminante igualmente puede omitirse.
- **Recursión**
    - Untipodedatorecursivo T se define como una estructura que puede contener componentes de tipo T. Ejemplos: árboles o listas de Pascal.

    - Define Datos Agrupados:
        - Cuyo Tamaño Puede Crecer arbitrariamente.
        - Cuyaestructura puede ser arbitrariamente compleja.
    - Los lenguajes soportan la implementación de tipos de datos recursivos a través de los punteros.

Inciso 2.

- **Java**
    - Producto Cartesiano.
- **Primero de C**
    - Producto Cartesiano y Recursión.
- **Segundo de C**
    - Unión.
- **Ruby**
    - Correspondencia Finita.
- **PHP**
    - Correspondencia Finita.
- **Python**
    - Correspondencia Finita.
- **Primero de Haskell**
    - Recursión.
- **Segundo de Haskell**
    - Unión.

Ejercicio 4

Inciso 1.

- **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 otro lado, los datos mutables en Python incluyen listas, conjuntos y diccionarios, ya que se pueden modificar después de haber sido creados.
- 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.

Inciso 2.

- No se puede brindar una respuesta completamente certera con el fragmento de código proporcionado, el hecho de que se modifique el valor de “a” a partir de la

creación de un nuevo objeto podría dar indicios de que el objeto “Dato.new(1)” es inmutable pero no se puede hablar con certezas ya que no tengo acceso a la implementación de Dato, por lo tanto no puedo conocer si posee alguna función o método para poder cambiar el valor de un objeto luego de su creación, si ese fuera el caso, la respuesta sería que es mutable pero como no puedo confirmar eso lo considero inmutable con la información que se me proporciona.

Ejercicio 5

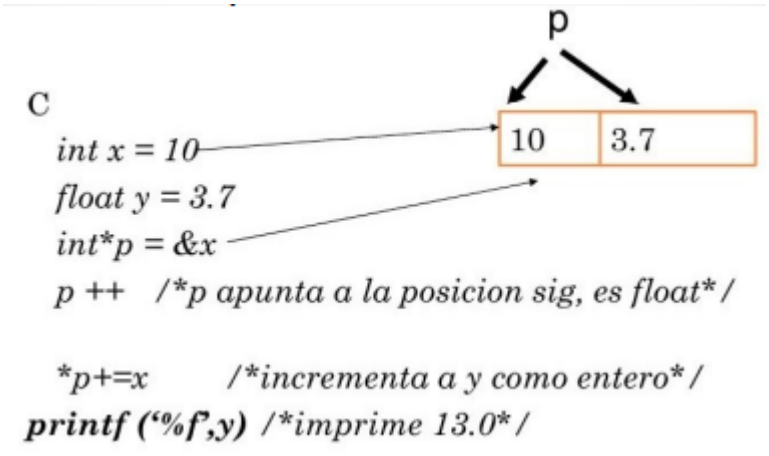
Inciso 1.

- Si, en C se puede tomar el l-valor de una variable utilizando el operador “&”.
- Ejemplo

```c
#include <stdio.h>

int main() {
    int x = 10;     // Declaración de una variable entera
    int *ptr;       // Declaración de un puntero a entero

    ptr = &x;       // El puntero `ptr` toma el l-valor (dirección) de la variable `x`

    printf("Valor de x: %d\n", x);        // Imprime el valor de x
    printf("Dirección de x: %p\n", (void*)&x); // Imprime la dirección de x
    printf("Valor almacenado en ptr: %p\n", (void*)ptr); // Imprime el valor almacenado en ptr (dirección de x)
    printf("Valor apuntado por ptr: %d\n", *ptr); // Imprime el valor apuntado por ptr (valor de x)

    *ptr = 20;      // Modifica el valor de `x` usando el puntero `ptr`

    printf("Nuevo valor de x: %d\n", x); // Imprime el nuevo valor de x

    return 0;
}
```

Inciso 2.

- **Problemas**
  - **Violación de TIpos**



  - **Referencias Sueltas**

        - 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.
    - **Punteros no Inicializados**
        - Peligro de acceso descontrolado a posiciones de memoria.
        - Verificación dinámica de la inicialización.
        - Solución -> valor especial nulo:
            - nil en Pascal.
            - void en C/C++.
            - null en ADA,Python.
    - **Punteros y uniones discriminadas**
        - Puede permitir accesos a cosas indebidas.
        - Javalo Soluciona eliminando la noción de puntero explícito completamente.
    - **Alias**
        - Si 2 o más Punteros comparten Alias, la modificación que haga uno se verá también reflejado en los demás.
    - **Liberación de Memoria - Objetos Perdidos**
        - Si los objetos en la heap dejan de ser accesibles, esa memoria podría liberarse.
        - Un objeto se dice accesible si alguna variable en la pila lo apunta directa o indirectamente.
        - Un objeto es basura si no es accesible .

Ejercicio 6

Inciso 1.

- **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 Información**
    - La representación de los objetos y la implementación del tipo permanecen ocultos.
    - Refleja los niveles de abstracción. Modificabilidad.

Inciso 2.

- En ADA se conocen como Paquete.
- En Modula-2 se conoce como módulo.
- En C++ y Java se conocen como clase.