Conceptos y Paradigmas de lenguajes de Programación
2025

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?
Sistema de tipos: Conjunto de reglas usadas por un lenguaje para estructurar y organizar sus tipos. El objetivo de un sistema de tipos es escribir programas seguros.
Conocer el sistema de tipos de un lenguaje nos permite conocer de una mejor forma los aspectos semánticos del lenguaje
Funciones de un sistema de tipos:

- 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.

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.

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.

- 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: Tipos de datos:

1. Dar una definición de tipo de dato.

Tipo: Conjunto de valores y un conjunto de operaciones asociadas para manipularlos

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

Los tipos de datos predefinidos o built-in son los que vienen equipados con el lenguaje y que normalmente reflejan el comportamiento del hardware subyacente y son 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: Caracteres, reales, booleanos, enteros.

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. Esto se logra mediante constructores. A estas agrupaciones se las conoce como tipos de datos definidos por el usuario

Ejemplo: Listas, Arreglos, Registros

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

El producto cartesiano de n conjuntos A1, A2,...,An, denotado A1xA2x...xAn, es un conjunto cuyos elementos están ordenados n-tuplas (a1, a2,...,an), donde cada ak pertenece a Ak. El Producto cartesiano de n conjuntos de tipos variados. Permite producir registros (Pascal) o struct (C).

Conceptos y Paradigmas de lenguajes de Programación
2025

Correspondencia finita:

La Correspondencia Finita es una función que mapea un conjunto finito de valores de un tipo de dominio DT (por ejemplo, un entero) a valores de un tipo de dominio RT, accesible a través de un índice. Son Las Listas indexadas, vectores, arreglos, matrices, etc.

Union y union discriminada:

La Unión o Unión Discriminada de uno o más tipos representa la disyunción de los tipos dados, con campos mutuamente excluyentes que no pueden tener valores al mismo tiempo, permitiendo manipular diferentes tipos en distintos momentos de la ejecución. Este tipo de estructura requiere chequeo dinámico, ya que no se puede asegurar en tiempo de compilación qué variante o tipo adquiere una variable. La Unión Discriminada mejora la unión al agregar un descriptor enumerativo que indica cuál de los campos tiene valor, brindando mayor seguridad al manipular el elemento según el discriminante, aunque este descriptor puede omitirse si se desea.

Recursión

Un tipo de dato recursivo T se define como una estructura que puede contener componentes de su mismo tipo T, como árboles o listas en Pascal, que son ejemplos comunes de esta categoría. Los datos agrupados definidos de esta manera tienen un tamaño que puede crecer arbitrariamente y una estructura que puede ser arbitrariamente compleja. Los lenguajes de programación soportan la implementación de tipos de datos recursivos mediante el uso de punteros, que permiten referenciar elementos del mismo tipo de manera dinámica

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

- Java: Producto cartesiano
- C (typedef struct…): Producto cartesiano y recursión
- C (union…): Union
- Ruby: Correspondencia finita
- PHP: Correspondencia finita
- Python: Producto cartesiano
- Haskell (data ArbolBinario…) : Recursión y union
- Haskell (data Color…): Union

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.

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.

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) es mutable basándonos en el código proporcionado. La razón es que el código no intenta modificar el estado del objeto Dato.new(1); simplemente reasigna la variable a a un nuevo objeto Dato.new(2). La reasignación no implica mutabilidad, ya que no altera el estado interno del objeto original. Sin conocer la definición de la clase Dato, no podemos determinar si Dato.new(1) es mutable (es decir, si permite modificar su estado) o inmutable (es decir, si cualquier cambio genera un nuevo objeto). Para responder definitivamente, necesitaríamos ver la implementación de la clase Dato y verificar si tiene métodos que permitan modificar su estado interno

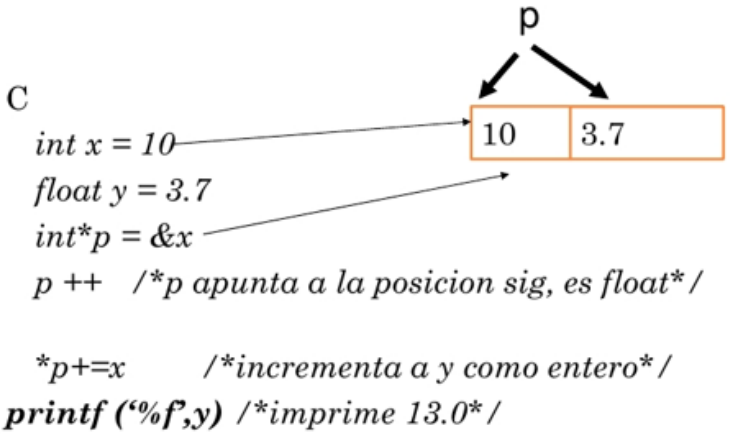
Ejercicio 5: Manejo de punteros:

1. ¿Permite C tomar el l-valor de las variables? Ejemplificar.
Si, se puede usando el operador &. Ejemplo:

```
int x= 10;  //declaro una variable de tipo entera
int *puntero;     //declaro un puntero a entero
puntero= &x;      //el puntero a entero toma el l-valor (direccion) de
la varible x
```

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

- Violación de tipos: Cuando un puntero se usa para acceder a datos de un tipo diferente al esperado, lo que puede llevar a comportamientos indefinidos o errores

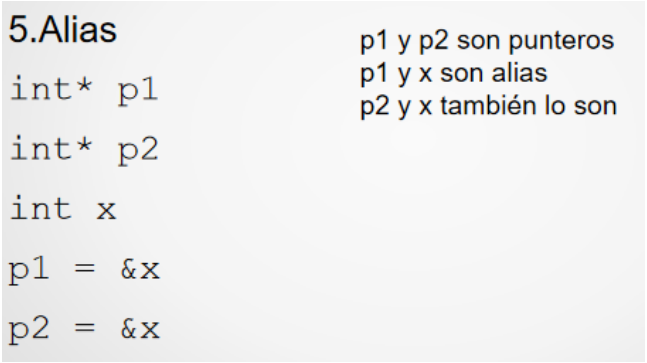


- Referencias sueltas – referencias dangling: Una referencia suelta o dangling es un puntero que contiene una dirección de una variable dinámica que fue desalocada. Si luego se usa el puntero producirá error.

- Punteros no inicializados:Un puntero no inicializado contiene una dirección aleatoria, lo que puede provocar accesos descontrolados a memoria si se usa sin inicialización previa, requiriendo verificación dinámica

- Punteros y uniones discriminadas: El uso de punteros con uniones discriminadas puede permitir accesos indebidos si no se verifica correctamente el tipo de datos almacenado, accediendo a campos incorrectos

- Alias:Cuando dos o más punteros comparten un alias (apuntan a la misma memoria), una modificación a través de uno afecta a los demás, lo que puede causar efectos secundarios inesperados.



- Liberación de memoria: objetos perdidos : Si los objetos en la heap dejan de ser accesibles (porque ninguna variable en la pila los apunta directa o indirectamente), se convierten en basura y la memoria no se libera, causando fugas de memoria.

Ejercicio 6: TAD :

1. ¿Qué características debe cumplir una unidad para que sea un TAD?
   - Encapsulamiento: la representación del tipo y las operaciones permitidas para los objetos del tipo se describen en una única unidad sintáctica.
   - Ocultamiento de la información: la representación de los objetos y la implementación del tipo permanecen ocultos.

2. Dar algunos ejemplos de TAD en lenguajes tales como ADA, Java, Python, entre otros.
   - ADA → paquete
   - Python → clase
   - Java → clase
   - Modula-2 →módulo