Conceptos y Paradigmas de Lenguajes de Programación - Práctica 7

EJERCICIO 1-1

Sistemas de tipos: conjunto de reglas usadas por un lenguaje para estructuras 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?
  - Interferencia de tipos - ¿cuál tipo se deduce del contexto?

- Mientras más flexible el lenguaje, más complejo el sistema

EJERCICIO 1-2

- Se dice que el sistema de tipos es **fuerte** cuando específica restricciones de forma clara sobre cómo las operaciones que involucran valores de diferentes tipos puede 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.

EJERCICIO 1-3

Tipos de 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 de resultado.
- Tipado dinámico: ligaduras en tiempo de ejecución. Que las ligaduras se den en tiempo de ejecución no vuelve a este tipado un tipado inseguro. (Provoca más comprobaciones en ejecución)
- Ejemplos:
  - C tiene tipado estático.
  - Python tiene tipado dinámico.
  - GOBSTONE tiene tipado estático.

EJERCICIO 2-1

Tipo de datos: podemos definir a un tipo como un conjunto de valores y un conjunto de operaciones que se pueden utilizar para manipularlos.

EJERCICIO 2-2

Tipo de datos predefinido: refleja el comportamiento del hardware subyacente y son una abstracción de él. En particular los elementales/escalares son tipos de datos predefinidos invisibles, es decir, no se pueden descomponer a partir de otros.

Ventajas

- Invisibilidad de la representación
- Verificación estática
- Desambiguar operadores
- Control de precisión

Ejemplos

- Enteros
- Reales
- Caracteres
- Booleanos

EJERCICIO 2-3

Tipo de datos predefinido: 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 3-1

Producto cartesiano

- Es el producto cartesiano entre n conjuntos de tipos variados. Permite producir registros (Pascal) o struct (C).

Correspondencia finita

- Es una función de un conjunto finito de valores de un tipo de dominio DT en valores de un tipo de del dominio RT. (DT: tipo de dominio. RT: resultado del dominio).
- Ejemplos: listas indexadas, vectores, arreglos, matrices, etc. (arreglos semidinámicos en ADA también)

Unión y unión discriminada

- La unión/unión discriminada de uno o más tipos define un tipo como 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 distinto momento de la ejecución.
- Chequeo dinámico: no se puede asegurar en compilación qué tipo po variante adquiere una variable.

- Agrega un descriptor (enumerativo) que me permite saber con quién estoy trabajando y acceder correctamente a lo que tengo que acceder, ya que nos dice cuál de los campos posee valor. Básicamente manipulo el elemento según el valor del discriminante, es una mejora de la unión que brinda mayor seguridad. Este discriminante puede omitirse.

Recursión

- Un tipo de dato recursivo 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
    - cuya estructura puede ser arbitrariamente compleja
- Los lenguajes soportan la implementación de tipos de datos recursivos a través de los punteros.

EJERCICIO 3-2

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: recursión.	C: unión.
RUBY: correspondencia finita.	PHP: recursión.	PYTHON: correspondencia finita.
HASKELL: recursión.	HASKELL: unión.	

EJERCICIO 4-1

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 duplas. Por otro lado, los datos mutables incluyen listas, conjuntos y diccionarios.
- 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.

EJERCICIO 4-2

No se puede afirmar que el objeto “Dato.new(1)” sea mutable solamente con la información proporcionada en el código.
El hecho de que se cree una nueva instancia de Dato y se le asigna 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 pudiera modificar el valor de la instancia Dato.new(1) después de su creación entonces el objeto sería mutable, en caso contrario seria inmutable.
Se necesita más información sobre la clase Dato y su implementación para determinar la mutabilidad del objeto Dato.new(1).

EJERCICIO 5-1

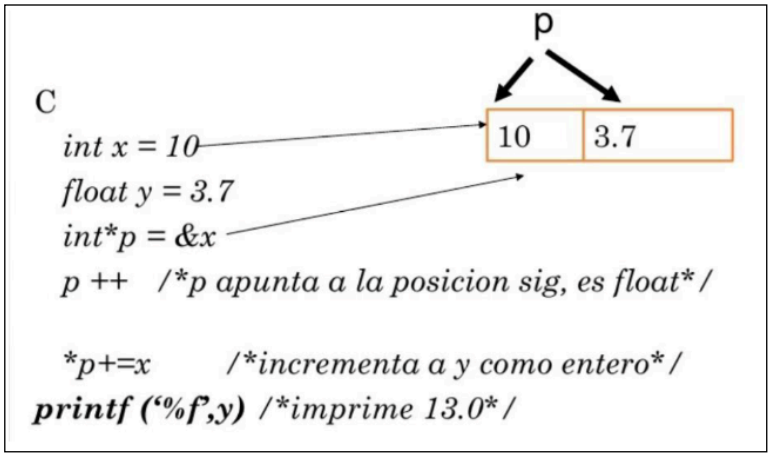
Si, en C se puede tomar el l-valor (dirección de memoria) de una variable utilizando el operador ‘&’.

```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;
}
```

EJERCICIO 5-2

Problemas

- **Violación de tipos**



- **Referencias sueltas**
  - Una referencia suelta o dangling es un puntero que contiene una dirección de una variable dinámica que fue desalojada, si luego se usa el puntero producirá un error.
- **Punteros no inicializados**
  - Peligro de acceso descontrolado a posiciones de memoria
  - Verificación dinámica de la inicialización
  - Solución —> valor espacial nulo:
    - nil en Pascal
    - void en C/C++
    - null en ADA, Python
- **Punteros y uniones discriminadas**

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

  - Puede permitir accesos a cosas indebidas
  - Java lo soluciona eliminando la noción de puntero explícito completamente

- **Alias**
  - Si 2 o más punteros comparten alias, la modificación que haga uno se verá también reflejada en los demás.

```
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

- **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
    - 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 6-1

Características debe cumplir una unidad para que sea un TAD (tipo de dato abstracto):

- **Encapsulamiento**
  - La representación del tipo y las operaciones permitidas para los objetos del tipos 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.

EJERCICIO 6-2

- **ADA:** se definen en paquetes (package).
- **Modula-2:** se definen en módulos (module).
- **C++:** se definen como clases o estructuras (class/struct).
- **Java:** se definen como clases (class).
- **Python:** se definen como clases (class).

Ejemplos

- 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.