**PRÁCTICA 7**

**Ejercicio 1**

1. ¿Qué es un sistema de tipos y cuál es su función principal?
2. Definir y contrastar las definiciones de un sistema de tipos fuerte y débil (probablemente en la bibliografía se encuentran en las 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 punto como estático o dinámico. ¿Qué significa esto? Ejemplificar con al menos 2 lenguajes para cada uno de ellos y justificar.
4. Conjunto de reglas que estructuran y organizan una colección de tipos. El objetivo del sistema es lograr que los programas sean tan seguros como sea posible.

* **Sistema de tipos fuerte** : Se dice que el sistema de tipos es fuerte cuando se especifican las condiciones que involucran valores de diferentes tipos de operación. Ejemplo: Haskell, C ++, etc.
* **Sistema de tipos débil** : Se dice que el sistema de tipos es débil cuando el lenguaje no responde. Ejemplo: PERL, JavaScript, etc.
* **El tiempo** está dentro de la compilación, y no durante la ejecución. Ejemplos de lenguajes que usan el estado estático son C, C ++, Java y Haskell. Comparado con el tipado dinámico, el estático permite que los errores de tipificación sean detectados antes, y que la ejecución del programa sea más eficiente y segura.
* **Tipado dinámico** : Se **trata** de un lenguaje de programación que se usa un tipado dinámico cuando la comprobación de tipificación se realiza durante su ejecución en la vez que durante la compilación. Ejemplos de lenguajes que usan tipado dinámico son Perl, Python y Lisp. Comparado con el modelo estático, el sistema de enlace temprano, el estado dinámico es más flexible (debido a las limitaciones teóricas de la decisión de los problemas de análisis de los programas estáticos, que impiden el mismo nivel de flexibilidad que se obtiene con el tipado estático ), a pesar de ejecutarse más lentamente y ser más propensos a contener errores de programación.

**Ejercicio 2**

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.
4. Un tipo de dato puede ser concebido de tres formas diferentes que se complementan entre si:

* Desde el punto de vista **denotacional** , es un conjunto de valores sobre un dominio.
* Desde el punto de vista **constructivo** , puede ser:
* *Primitivo* (incorporado o predefinido): provisto por el lenguaje.
* *Compuesto* (compuesto o derivado): empleando constructores de tipos.
* Desde el punto de vista de la **abstracción** , puede ser:
* Una interfaz a una representación.
* Conjunto de operaciones con semántica bien definido y consistente.

1. Un tipo predefinido elemental es aquel que ya está definido con el lenguaje y que no está compuesto de otros tipos. Ejemplo: Los caracteres e integrales son predefindos (tipo de elemento elemental dado que no se puede dividirlos en otros tipos). Un string podría ser predefinido (depende del lenguaje), pero no es elemental y que un string es un conjunto de char.
2. Un tipo definido por el usuario es un tipo que se define por el tipo predefinido que el usuario elije. Algunos ejemplos son los arreglos o las listas. Una lista por si sola no es nada, no tiene sentido; no asi una lista de enteros o una lista de cuerdas en el cual es su tipo ya ha sido definido por el usuario.

# Ejercicio 3

1. Dar una breve definición de producto cartesiano, correspondencia finita, uniones (tipo de producto) y tipos recursivos.
2. Identificar una clase de tipo de datos que se detallan a continuación. En algunos casos puede corresponder más de una:
3. El producto cartesiano de los dos conjuntos es una operación, que resulta en otro conjunto, los elementos son todos los pares se ordenan el primer elemento. La unión / unión discriminada de dos o más tipos define un tipo como la disyunción de los tipos dados. Permite manipular diferentes tipos en distinto momento de la ejecución. Un tipo de dato recursivo Se define como una estructura que puede contener componentes del tipo T. Definir datos agrupados cuyo tamaño puede crecer arbitrariamente y su estructura puede ser arbitrariamente compleja.

* Java
  + Clase Persona: definido por el usuario compuesto tad.

|  |  |  |
| --- | --- | --- |
| **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;    Identificación de caracteres;  }; |
| **Ruby** hash = {    uno: 1,    dos: 2,    tres: 3,    cuatro: 4  } | **La** función **PHP** doble ($ x) {    return 2 \* $ x;  } | **Python** tuple = ('física', 'química', 1997, 2000) |
| Datos de **Haskell** ArbolBinario Int =    Nil |    Nodo int    (ArbolBinarioInt Dato)    (ArbolBinarioInt Dato)  **Ayuda para Interpretar:** 'ArbolBinarioInt' Es Un  tipo de dato Que Puede Ser  Ninguna ( “vacío”) o Un Con Nodo  ONU Dato Número entero (int)  Junto a Un Árbol de Como hijo  izquierdo y Otro árbol como  hijo derecho. | Datos de **Haskell** Color =    Rojo |    Verde |    Azul  **Ayuda para interpretar:** 'Color' es un tipo de dato que  puede ser Rojo, Verde o Azul. |  |

* + String nombre ;: predefinido compuesto.
  + String apellido ;: predefinido compuesto.
  + int edad ;: predefinido elemental.
* do
  + typedef struct nodoLista {: definido por el usuario compuesto producto cartesiano.
  + void \* dato ;: definido por el usuario recursivo.
  + struct nodoLista \* siguiente: definido por el usuario compuesto recursivo.
  + typedef struct lista {: definido por el usuario producto cartesiano.
  + Int cantidad ;: predefinido elemental.
  + nodoLista \* primero: definido por el usuario recursivo.
* do
  + union codigo {: definido por el usuario union.
  + Int numero ;: predefinido elemental.
  + Identificación de caracteres; predefinido elemental.
* Rubí
  + hash = {: definido por el usuario o predefinido.
  + uno: 1 ,: predefinido o definido por el usuario.
  + dos: 2 ,: predefinido o definido por el usuario.
  + tres: 3 ,: predefinido o definido por el usuario.
  + Cuatro: 4: predefinido o definido por el usuario.
* PHP
  + Función doble ($ x) {: definido por el usuario tad.
  + $ x: definido por el usuario o predefinido.
* Phyton
  + tuple = ('physics', 'chemistry', 1997, 2000): definido por el usuario compuesto.
* Haskell
  + datos ArbolBinario Int = Nil | Nodo int (ArbolBinarioInt dato) (ArbolBinarioInt dato): definido por el usuario recursivo.
* Haskell
  + Datos de Haskell Color = Rojo | Verde | Azul: definido por el usuario.

**Ejercicio 4**

1. Definir mutabilidad e inmutabilidad respecto a un dato. Dar ejemplos en al menos 2 lenguajes. CONSEJO: 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 del mar por afirmativa o por la negativa.

1. La mutabilidad de un dato es una propiedad que establece que un objeto puede ser cambiado durante su ejecución; Por su parte, la inmutabilidad es todo lo contrario: se ha establecido una vez creado. En ruby, este concepto es tan literal en el sentido de que permite crear constantes y mediante el método de congelación hacer que su valor no cambie. En python, ocurre algo diferente. Los números, las cuerdas y las tuplas son inmutables. Por el contrario, las listas son mutables. Esto no quiere decir que, estrictamente, un número o una cadena no puedan cambiar el valor. El concepto de inmutabilidad, en este caso, se aplica a si yo tengo dos variables, por ejemplo, "x" e "y" me gustaría asignarle el valor de "x" a "y" y luego cambiara "x", el valor que yo estableci en "y" sigue siendo el que yo habia definido correctamente. El mismo valor ("y" será un alias de "x").
2. Es mutable ya que se está visualizando el objeto, es decir, se está modificando su estado.

**Ejercicio 5**

1. ¿Permite tomar la l-valor de las variables? Ejemplificar.
2. ¿Qué problemas existen en el manejo de punteros? Ejemplificar.
3. Si, anteponiendo y una variable cualquiera se puede saber su referencia. Por ejemplo, si yo tengo una variable xy le antepongo & (& x) puedo saber cual es su direccion de memoria.
4. Hay que tener varias referencias. En este caso el puntero apunta a basura. Tambien podria darse un acceso descontrolado a posiciones de memoria inexistentes. En este caso conviene inicializar los punteros con un valor como podria ser nil o null, considerado el lenguaje.

**Ejercicio 6**

1. ¿Qué características deben cumplir una unidad para que mar un TAD?
2. Dar algunos ejemplos de TAD en lenguajes tales como ADA, Java, Python, entre otros.
3. Un TAD está relacionado con un conjunto de operaciones (funciones) a lo que se denomina usualmente como interfaz pública y representa el comportamiento del TAD; Mientras que la implementación como la parte privada del TAD está oculta en el programa cliente que lo usa. Todos los lenguajes de alto nivel tienen predefinidos TAD; que son los tipos denominados simples y las estructuras predefinidas, y estos tienen sus interfaces públicas que incluyen las operaciones como la +, -, \*, etc. , generalmente es una implementación bastante dependiente de la máquina sobre el que trabaja el compilador.  
   Los datos que se presentan en el futuro son los que se han reflejado en un cierto comportamiento organizativo. Una forma estructurada de almacenar los datos será la que nos refiramos para caracterizar cada TAD. TAD que tienen información simple.  
   Nótese que cuando hablemos de un TAD no haremos ninguna alusión al tipo de los elementos sino tan solo a la forma en que están dispuestos estos elementos. Solo nos interesa la estructura que soporta la información y las operaciones. Para determinar el comportamiento estructural basta con observar la conducta que seguirán los datos.  
   Caracterizamos entonces los TAD. Un TAD tendrá una parte que será invisible al usuario lo que hay que proteger y que se puede decir que es irrelevante para el uso que el usuario y está constituido tanto por la maquinaria algorítmica que implemente la semántica de las operaciones como por ejemplo los datos que sirvan de enlace entre los elementos del TAD, es decir, información interna necesaria para la implementación. Resumiendo podemos decir, que tanto la implementación de las operaciones como los elementos internos del TAD serán privados en el acceso externo y ocultos a cualquier otro nivel.   
   Un TAD representa una abstracción:

* Se detallan los detalles de la especificación (el qué).
* Se ocultan los detalles (casi siempre) de la implementación.

1. Algunos ejemplos del uso de TAD en programación son:

* Conjuntos: implementación de conjuntos con sus operaciones básicas (unión, intersección y diferencia), operaciones de inserción, borrado, búsqueda, etc.
* Árboles Binarios de Búsqueda: Implementación de árboles de elementos, usos para la representación interna de datos complejos. Aunque siempre se han tomado como parte de la familia de los grafos.
* Pilas y colas: Implementación de los algoritmos FIFO y LIFO.
* Grafos: Implementación de grafos; una serie de vértices unidos mediante una serie de arcos o aristas.