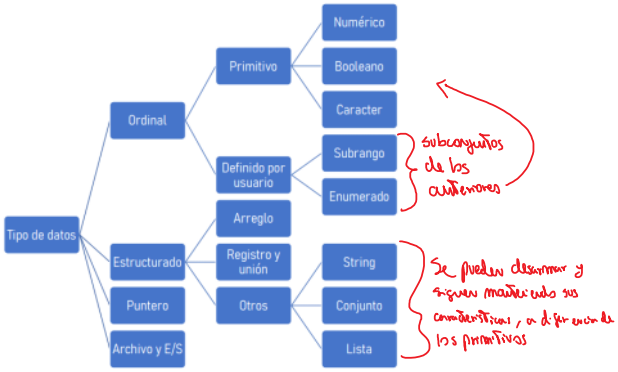
# Sistema de Tipo de Datos

lunes, 11 de octubre de 2021 22:09

- **Definición**
  - Define la manera en que un lenguaje clasifica en tipos los valores y expresiones, y como interactúan entre sí.
  - Permite definir nuevos tipos de datos.
- **Ventajas**
  - Permite verificar el uso correcto de datos y detectar errores de tipos, en tiempos de compilación y ejecución:
  - Ayuda a modularizar (bibliotecas y paquetes).
- **Desventajas**
  - Puede rechazar programas correctos.

## Taxonomía de Tipos de Datos




- **Tipos de datos**
  - Define un conjunto de valores de datos y conjunto de operaciones predefinidas sobre los objetos de datos
  - Disponen de un gran número de tipos de datos que se adaptan a diversos problemas.
  - Los lenguajes de programación proveen conjuntos de datos primitivos y mecanismo que permiten definir nuevos tipos enfocado a la resolución de un problema específico.
  - Ejemplos:
    - Primitivos (Enteros, flotantes y caracteres)
    - Estructuras de datos (Arreglos y Registros)
    - Tipos definido por usuario (structs) y tipos de datos abstractos.

- **Conceptos Básicos**
  - Chequeo o Verificación de tipo:
    - Asegura que los operandos de un operador son de tipos compatibles
  - Tipo compatible:
    - Tipo legal o que puede ser convertido a un tipo legal
  - Conversión de tipos:
    - Hay dos tipos, la *coerción* (Conversión automática) y el *casting* (Conversión definida por el programador)
  - Error de tipo:
    - Aplicación de un operador a un tipo inapropiado

- **Equivalencia de Tipos**
  - Se dice que dos tipos son equivalentes si un operando de un tipo en una expresión puede ser sustituido por otro tipo sin necesidad de coerción (O sea, si se puede hacer con un cast)
  - Las reglas de compatibilidad de tipos del lenguaje determinan los operandos aceptables para cada operador, por ejemplo, hay lenguajes que no permiten sumar un string con un carácter mientras que en otros simplemente concatena.
  - Dos tipos son compatibles cuando en tiempo de compilación/ejecución los tipos de los operandos pueden ser implícitamente convertidos (coerción) para ser aceptables por el operador
  - La equivalencia es una forma más estricta de compatibilidad pues no requiere coerción
  - La compatibilidad en tipos estructurados son más complejos que en escalares.

- **Compatibilidad de Tipos Estructurados**
  - Equivalencia de tipo Nominal:
    - Las variables están en la misma declaración
  - Equivalencia de tipo Estructural:
    - Los tipos tienen estructura idéntica, pero tienen un nombre distinto (Var1, Var2, etc.)

Ejemplos en Pascal:

```
TYPE
  mes_t = 1..12;
VAR
  indice : mes_t;
  contador: integer;
*****************************************
TYPE
  celsius = real;
  fahrenheit = real;
(otro ejemplo)
TYPE
  tipo1 = ARRAY [1..10] OF integer;
  tipo2 = ARRAY [1..10] OF integer; {tipo1 <> tipo2}
  tipo3 = tipo2; {equivalencia declarativa}
```

Eg. Tipo Nominal

Eg. Tipo Estructural

Ejemplo en C:

```
struct s1 {int c1; real c2 };
struct s2 {int c1; real c2 };

s1 x;
s2 y = x;  /* error de compatibilidad */
```

Ejemplo de equivalencia estruct. y no Nominal

```
*************************************************************

typedef char* pchar; /* define nombre, no nuevo tipo */
pchar     p1, p2;

char* p3 = p1;
```

- **Taxonomía de los Tipos de Datos**
  - Se pueden clasificar en:
    - Estática o Dinámica
    - Explícita o Implícita
    - Fuerte o Débil
  - Estos criterios no se excluyen entre sí, o sea, un tipo de dato puede ser explicito y estático o débil y dinámico

- **Tipificación Estática vs Dinámica**
  - Los sistemas de tipos de datos se pueden clasificar según el momento en que se realiza el chequeo de tipos:
    - Tipificación Estática (Lenguajes Compilados):
      - Se determina el tipo de todas las variables y expresiones **antes de la ejecución**, y luego permanece fijo
      - Los tipos pueden ser determinados explícitamente (Declaraciones) o inferidos mediante reglas (ej: C)
      - Aplica solo si los tipos son ligados estáticamente
    - Tipificación Dinámica (Lenguajes Interpretados):
      - Se determina el tipo **durante la ejecución**, y normalmente es inferido
      - Tipos asociados a una misma variable pueden variar según valor asignado
      - Errores solo pueden ser detectados durante ejecución (Python y Scheme)

- **Tipificación Explícita vs Implícita**
  - Dependiendo del grado de exigencia para definir los tipos asociados a todos los objetos, se puede clasificar como:
    - Tipificación Explícita (Lenguajes Compilados):
      - Todos los tipos de datos asociados a variables y otros elementos deben ser declarados y estar bien definidos
      - Deseable para tipificación estática
    - Tipificación Implícita (Lenguajes Interpretados):
      - Los tipos de datos no se declaran y son inferidos a través de reglas por el programa (Nombre de variables o tipos de datos en las expresiones)

- **Tipificación Fuerte vs Débil**
  - Los sistemas de tipos se pueden clasificar según el grado de exigencia impuesto en la verificación para detectar potenciales errores de tipos:
    - Tipificación Fuerte:
      - Siempre detecta errores de tipo (Estática o Dinámicamente)
      - Pone restricciones fuertes sobre como las operaciones aceptan valores de diferentes tipos de datos e impiden la ejecución del programa si estos tipos son erróneos
      - Promueve la tipificación estática y explicita
      - Ej: A un puntero no le puedes asignar un int
      - Tipificación Fuerte Implícita:
        - Eficiencia de ejecución:
          - Permite al compilador asignar memoria y generar código que manipule los datos eficientemente
        - Seguridad y confiabilidad:
          - Permite detectar un mayor número de errores
          - Permite verificar mejor compatibilidad de interfaces en la integración de módulos/componentes de programa
      - Tipificación Fuerte Explícita:
        - Legibilidad:
          - Mejora al documentar bien los tipos usados
        - Ambigüedad:
          - Permite eliminar ambigüedades
    - Tipificación Débil:
      - Se realizan implícitamente conversiones que permiten relajar las restricciones, pero podrían generarse errores no detectados

# Tipos de Datos Ordinales o Simples

lunes, 11 de octubre de 2021 22:28

- **Definición**
  - Un tipo ordinal es aquel que puede ser asociado a un número natural
  - Incluye tipos primitivos y tipos definidos por el usuario
- **Tipos primitivo**
  - No están basados en otro tipo de lenguaje
    - Numérico
    - Carácter
    - Booleano
- **Tipos definidos por el usuario**
  - Basado en tipos primitivos
    - Enumerado
    - Subrango

- **Tipos Primitivos**
  - Tipos de datos que no están definidos en términos de otros tipos de datos
  - Se soportan directamente por el hardware
  - Numérico:
    - Entero (En C y Java pueden ser: *signed, unsigned, short, long*. Que ocupan diferentes cantidades de memoria)
    - Punto flotante
    - Decimal
  - Booleano:
    - Mejora legibilidad
    - Ocupa 1 byte
  - Carácter:
    - Ocupa 1 byte (Código ASCII)
    - Tamaño variable (UTF-8)
    - Unicode de 16b (UTF-16 en Java)
    - 32b (UTF-32)

  - Representación de Números:
    - Es un conjunto finito y una aproximación al concepto matemático
    - El rango y la precisión de la representación depende del largo del registro
    - Tiene soporte directo del hardware
    - Tipos:
      - Enteros (C-2, C-1)
      - Punto flotante:
        - Aproximación de números reales (Estándar IEEE 754)
      - Decimal:
        - Para aplicaciones de negocios
        - Mas preciso para números de base 10 pero ocupa más memoria
      - Complejo:
        - Representados como dos números de punto flotante (Python y Scheme)
    - Estándar IEEE 754:
      - Representa un numero punto flotante en precisión simple (32b) y doble (64b)
      - Estas representaciones están en la diapo 26 del PDF

De esta manera se almacena un Numero

| Signo | exponente | mantiza |
|---|---|---|
| s | e | m |
| 1b | 8b/11b | 23b/52b |

  - Representación del tipo carácter:
    - Facilita la comunicación de datos
    - Un sistema de caracteres define un conjunto de caracteres y un sistema de codificación para cada elemento, usando patrones de bytes
    - Estándares más conocidos:
      - ASCII
      - EBCDIC
      - UTF-8
      - UTF-16
    - Lenguajes usan caracteres de escape (ej: \) para representar caracteres sin afectar la sintaxis del lenguaje

- **Tipos definidos por el Usuario**
  - Subrango:
    - Subsecuencia contigua de un tipo ordinal ya definido
    - Mejora lectura y fiabilidad
    - Se implementa en base al tipo entero
  - Enumerado:
    - Tipo donde se enumeran todos los posibles valores a través de constantes literales
    - Establece una relación de orden que permite definir operadores relacionales, predecesor y sucesor
    - Mejora facilidad de lectura y fiabilidad
    - Se implementan mapeando las constantes según su posición a un subrango de enteros
    - No usan E/S

- Ejemplo: Pascal

restricción dentro de las restricciones del lenguaje

```
TYPE
mayúscula = 'A'..'Z';
indice = LUNES .. VIERNES;
```

x. mayúscula

mayúscula x

Ejemplos en C y C++

**Sintaxis:**

```
<enum-type>      ::= enum [<identifier>] { <enum-list> }
<enum-list>      ::= <enumerador> | <enum-list> , <enumerador>
<enumerador>     ::= <identificador> | <identificador> = <constant-exp>
```

**Código:**

0 1 2 3

```
enum color {rojo, amarillo, verde=20, azul};
color col = rojo;
color* cp = &col;

if (*cp == azul) // ...
```

# Arreglos y Strings

lunes, 11 de octubre de 2021 22:28

- **Tipo Arreglo**
  - Es un tipo estructurado consistente en un conjunto homogéneo y ordenado de elementos que se identifican por su posición relativa mediante un índice
  - Existe un tipo asociado a los elementos y otro al índice

- **Tipo Arreglo: Índices**
  - Definen un mapeo desde el conjunto de índices al conjunto de elementos del arreglo
  - Sintaxis:
    - FORTRAN y ADA usan paréntesis
    - Pascal, C, C++, Modula-2 y Java usan corchetes
  - Tipo de datos del índice:
    - C, C++, Java y Perl solo aceptan enteros
    - ADA y Pascal aceptan enteros y numerados (Como el ejemplo de Lunes…Viernes)
  - Prueba de rango de índice:
    - C, C++, Perl no tienen prueba de rango (Previene que sobrescriba datos en otra parte de la memoria)
    - ADA, Pascal, Java, C# si tienen prueba de rango

- **Tipo Arreglo: Ligado**
  - Arreglo Estático:
    - El rango de índice y la memoria son ligados antes de la ejecución (Mas eficiente)
  - Arreglo Dinámico Fijo de Stack:
    - El rango del índice es ligado estáticamente, pero la memoria se asigna dinámicamente (Memoria más eficiente)
    - Ejemplo: Arreglos definidos dentro de una función o procedimiento
  - Arreglo Dinámico de Stack:
    - El rango del índice y la memoria son ligados dinámicamente, permaneciendo fijos durante el tiempo de vida de la va
  - Arreglo Fijo de Heap:
    - El tamaño puede variar pero permanece fijo una vez asignada la memoria (Mas flexible)
    - Ej: Malloc de C
  - Arreglo Dinámico de Heap:
    - El tamaño puede variar durante su tiempo de vida (El más flexible)
    - Ej: Python y PERL

• Ejemplo: Arreglo Dinámico de *Stack* en C:

```
void foo(int n) {
        int a[n];
        for (int i=0; i<n; i++) {
                ...
        }
        ...
}
```

• Ejemplo: Arreglos de *Heap*

C:
```
char *str, *s, *t;
...
str = malloc(strlen(s) + strlen(t)+1);
strcat(strcpy(str, s), t);
```

PERL:
```
@a = split (" ", $text);
```

### Tipo Arreglo: inicialización

• Fortran
```
INTEGER MES(12);
DATA MES /31, 28, 31, 30, 31, 30, 31, 31, 30, 31, 30, 31/
```

• ANSI C y C++
```
char *mensaje = "Hola mundo\n";
char *dias[] = {"lu", "ma", "mi", "ju", "vi", "sa", "do"};
```

• Java
```
int[] unArreglo = {100, 200, 300, 400, 500, 600, 700, 800, 900, 1000};
String[] nombres = {"Pedro", "Maria", "Jose"};
```

• Pascal y Modula-2 no lo permiten

### Tipo Arreglo: multidimensionales

• Fortran IV: primera versión de Fortran que permite declarar hasta 7 dimensiones.

• Pascal: Permite sólo dos dimensiones
```
TYPE
    matriz = ARRAY [subindice, subindice] OF
real;
```

• C y C++:
```
real matriz [DIM1][DIM2];
```

- **Tipos Arreglo: Operadores**
  - APL provee soporte de vectores y matrices con operadores
  - ADA permite la asignación
  - Pascal, C y Java no tienen soporte especial (Solo el [i])
  - C++ permite definir una clase arreglo por el usuario y sus operadores (Subíndice, Asignación, Inicialización, etc.)
  - Python soporta arreglos mediante listas y provee varios operadores

- **Tipos Arreglo: Implementación**
  - Un arreglo es una abstracción del lenguaje
  - Debe ser mapeado a la memoria como un arreglo unidimensional de celdas
  - Los arreglos bidimensionales se almacenan como fila de columnas o viceversa

- **Tipo String**
  - Es una secuencia de caracteres usado para procesamiento de texto y E/S
  - Es implementado en base a un arreglo de caracteres (Aunque a veces un string es un tipo de datos separado)
  - Operaciones típicas:
    - Asignación, copia y concatenación
    - Largo y comparación
    - Referencia a *substring*
    - Reconocimiento de patrón
  - Largo:
    - Largo Estático en Fortran77, Pascal y Java
    - Largo Dinámico limitado en C y C++
    - Largo Dinámico en JavaScript, Python y Perl (Es el más flexible y costoso)

### Tipo String (3)

Ejemplo: C
```
char str[20];
...
if (strcmp(str,"Hola")){
    ...
else {
    ...
}
```

Ejemplo: Java
```
"Este es un String"                    // literal
String palindrome = "reconocer";       // referencia String
char[] ArregloHola = { 'h', 'o', 'l', 'a'};  // arreglo
String StringHola = new String(ArregloHola);
```

# Registros y estructuras relacionadas

lunes, 11 de octubre de 2021 22:29

- **Tipo Registro**
  - Conjunto heterogéneo de elementos de datos, donde cada elemento individual (Llamado campo o miembro) es identificado con un nombre
  - Útil para procesamiento de datos en archivos
  - Estilos:
    - Cobol usa <campo> OF <nombre_registro>
    - Otros usan var.campo
  - Operaciones:
    - Típicamente solo asignación
  - En el pdf hay ejemplos de Cobol, ADA, Pascal, C y C++
  - Referencias Elípticas:
    - En esencia son cuando tienes un struct dentro de un struct o cosas así.
    - C y C++ permiten solo referencias de calificación completa
    - Otros permiten referencias elípticas, lo que es conveniente (Pascal, Cobol y PL/I)

- **Tipo Unión**
  - Tipo que permite almacenar diferentes tipo de datos en diferentes tiempos en una misma variable
  - Fortran, ADA C y C++ lo soportan
  - Comentarios:
    - Reserva espacio de memoria igual al **mayor miembro definido**
    - Todos los miembros comparten la memoria y comienzan desde la misma dirección
    - Su uso es poco seguro
  - En el pdf hay ejemplos de Cobol, ADA, Pascal, C y C++

- **Tipo Archivo**
  - Tipo especial que facilita la comunicación de un programa con el mundo externo (E/S)
  - Permite leer datos que existen antes de la ejecución del programa y escribir datos que persisten
  - Persistencia:
    - Los datos sobreviven más allá de la ejecución del programa. Persisten en el tiempo
  - Tipos:
    - Persistente vs Temporal
    - Método de acceso (Secuencial, Directo, indexado, etc.)
    - Estructurado (Secuencia de bytes o reconoce tipos estructurados)
  - Caso Especial:
    - C no tiene E/S, se tiene que importar con bibliotecas

- Ejemplo: Pascal

```
empleado.nombre.primer  := 'Juan';
empleado.nombre.paterno := 'Perez';
empleado.nombre.materno := 'Machuca';
```

se repite en todos

```
WITH empleado.nombre DO
  BEGIN
    primer  := 'Juan';
    paterno := 'Perez';
    materno := 'Machuca';
  END
```

# Colecciones

lunes, 11 de octubre de 2021 22:29

- **Definición**
  - Tipo de dato que agrupa un conjunto heterogéneo de elementos con operadores para construirlos y acceder a ellos
- **Tipos de Colecciones**
  - Ordenadas:
    - Los elementos tienen definida una relación de orden (lineal) en base a la posición:
      - Arreglos, vectores, listas y tuplas
      - Stacks y colas
  - No ordenadas:
    - No existe un ordenamiento de los elementos
      - Conjuntos
      - Arreglos asociativos, tablas de hash o diccionarios
- **Comentarios**
  - Lenguajes de scripting modernos proveen tipos de colecciones más flexibles como:
    - Secuencias (String, tupla, lista, etc.)
    - Conjuntos
    - Diccionarios

- **Tipo Conjunto**
  - Permite almacenar un conjunto no ordenado de elementos de datos
  - Caso Pascal y Modula-2:
    - Define un tipo base (O sea, solo se puede ocupar un tipo del mismo tamaño de memoria)
    - No permite manejar eficientemente conjuntos grandes
  - Caso Python:
    - Es una implementación más flexible
    - No requiere definir un tipo base pero ocupa más memoria
    - Los elementos pueden ser de diferentes tipos}
  - Hay ejemplos de Python, Pascale, Java y Scheme en el PDF

- **Tipo Arreglo Asociativo**
  - Conjunto **no ordenado** de elementos de datos que son indexados por un número igual de valores llamados "Claves"
  - En esencia son como los diccionarios de Python o las tablas hash de C++
  - Lenguajes que lo soportan:
    - Python
    - Perl
    - Ruby
    - Java (Mediante biblioteca *package*)

## Ejemplo: Python

```
>>> tel = {'Pedro':123453, 'Maria':234534, 'Juan':453345, 'Ana':645342}
>>> tel
{'Juan': 453345, 'Pedro': 123453, 'Ana': 645342, 'Maria': 234534}
>>> tel['Juan']
453345
>>> list(tel.keys())
['Juan', 'Pedro', 'Ana', 'Maria']
>>> sorted(tel.keys())
['Ana', 'Juan', 'Maria', 'Pedro']
```

```
>>> 'Pedro' in tel
True
>>> 'Catalina' in tel
False
>>> del tel['Maria']
>>> tel
{'Juan': 453345, 'Pedro': 123453, 'Ana': 645342}
```

# Tipo Puntero y Referencias

lunes, 11 de octubre de 2021 22:29

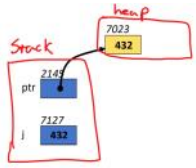
- **Tipo Puntero**
  - Su valor corresponde a una dirección de memoria
  - Existe un valor especial nulo que no apunta a nada
  - No corresponde a un tipo estructurado
  - Aplicaciones:
    - Método de gestión dinámica de memoria:
      - Mediante una variable permite el acceso a la memoria dinámica de *heap*
    - Método de direccionamiento indirecto:
      - Útil para diseñar estructuras (Listas, arboles, grafos)

- **Tipo Puntero: Operaciones**
  - Clases de Operadores:
    - Asignación:
      - Asigna la dirección de algún objeto de memoria del programa a una variable
    - Desreferenciación:
      - Entrega el valor almacenado en el objeto apuntado
      - Puede ser explicito o implícito

Puede ser explícito o implícito.

Ejemplo en C:

```
int *ptr, j;
ptr = (int*) malloc(sizeof(int));
*ptr = 432;
j = *ptr;
```




- **Tipo Puntero: C y C++**
  - Son extremadamente flexibles, por lo que se deben usar con cuidado
  - Son usados para la gestión dinámica de la memoria (*heap*) y para direccionamiento
  - Usan desreferenciación explicita (*) y un operador para obtener la dirección de una variable(&)
  - Definen un tipo de datos al que apunta un puntero
    - *Void** puede apuntar a cualquier tipo
  - Soporta una aritmética de punteros
  - En estos lenguajes, un arreglo es, en realidad, una constante de tipo puntero, que direcciona a la base del arreglo y permite el uso de la aritmética de punteros
  - Hay ejemplos en el PDF

```
struct nodo_t {
  tipodato    info;
  struct nodo_t *siguiente;
};
typedef nodo_t* enlace_t;

enlace_t nodo;
```


```
#ifndef C++
nodo = (enlace_t) malloc(sizeof(nodo_t));
#elif
nodo = new nodo_t;  /* caso C++ */
#endif
(*nodo).info = dato;

/* forma más conveniente de referirse es */
nodo->siguiente = NULL;
```

- **Tipo Referencia**
  - Tipo de variable que realiza desreferenciacion implícita en la asignación, haciéndola más segura
  - Ejemplos:
    - C++:
      - Después de su inicialización, las referencias permanecen constantes.
      - Útiles para usar parámetros pasados por referencia
      - Cuando se realiza una asignación, no se requiere desreferenciar la variable
      - Su uso en parámetros de funciones permite el paso por referencia (Comunicación Bidireccional)
      - La inicialización se produce al momento de la invocación
    - Java:
      - Extiende lo de C++, haciendo que referencias se hagan sobre objetos en vez de direcciones
      - No existe operador de desreferenciación
      - Una asignación provoca que apunte a un nuevo objeto
    - C#:
      - Permite el uso de referencias al estilo Java y punteros como C++

```
int valor = 3;
int &ref_valor = valor;          /* inicializa */

ref_valor = 100;
```

- **Gestión del *Heap***
  - Para manejo de objetos de memoria dinámicos, punteros y referencias son implementados en el *heap*
  - Su implementación debe atacar problemas de *basura* y *dangling* (Manejo del *heap*)
  - Tamaño único:
    - El caso más simple
    - Es administrar objetos (celdas) de memoria de tamaño único:
      - Celdas libres se pueden enlazar con punteros en una lista
      - La asignación es simplemente tomar suficientes celdas (contiguas) de la lista anterior
      - La liberación es un proceso más complicado
  - Tamaño variable:
    - Es lo común que requieren los lenguajes
    - Es complejo de administrar e implementar
  - El administrador de la *heap* marca cada celda de memoria que administra como *libre* u *ocupada*, no obstante, una celda puede tener cuatro estados reales:
    - Libre:
      - No tiene referencias y está marcada correctamente como libre por el administrador de memoria
    - Basura:
      - No tiene referencia y no está marcada como libre
      - El administrador no la puede reasignar
    - Ocupada:
      - Tiene al menos una referencia y esta asignada
    - *Dangling*:
      - Tiene una referencia y está marcada como libre
      - El administrador la puede reasignar