Clase 7

Tipos de datos

Un tipo de dato es un conjunto de valores y un conjunto de operaciones asociadas al tipo para manejar esos valores. A los datos los interpreto según su tipo.

<table>
<tr><th></th><th>ELEMENTALES</th><th>COMPUESTOS</th></tr>
<tr><td rowspan="4">PREDEFINIDOS</td><td>ENTEROS</td><td rowspan="4">STRING</td></tr>
<tr><td>REALES</td></tr>
<tr><td>CARACTERES</td></tr>
<tr><td>BOOLEANOS</td></tr>
<tr><td rowspan="4">DEFINIDOS POR EL USUARIO</td><td rowspan="4">ENUMERADOS</td><td>ARREGLOS</td></tr>
<tr><td>REGISTROS</td></tr>
<tr><td>LISTAS</td></tr>
<tr><td>ETC.</td></tr>
<tr><td colspan="3">DOMINIO DE UN TIPO ----> VALORES POSIBLES</td></tr>
</table>

Cualquier lenguaje de programación esta “equipado” con un conjunto de tipos predefinidos o primitivos que reflejan el comportamiento del hardware además de esto, los lenguajes permiten al programador especificar agrupaciones de objetos de datos elementales, con el fin de definir lo que denominamos como “tipo de dato definido por el usuario”.

Definición del lenguaje != implementación

Si un conjunto de valore son definidos por la implementación del lenguaje, significa que será seleccionado por el compilador (depende de la maquina) mientras que, si el tipo es definido por el lenguaje, será definido por su definición (depende del lenguaje)

Tipos predefinidos

- Reflejan el comportamiento del hardware de abajo y son una abstracción de él.
- Ventajas:
  - invisibilidad de la representación
  - verificación estática
  - desambiguar operadores
  - control de precisión

Tipos definidos por el usuario

Los lenguajes de programación permiten al programador especificar agrupaciones de objetos de datos elementales y de forma recursiva, agregaciones de agregados

- Separa la especificación de la implementación.
- Se definen los tipos que el problema necesita.
  - Definir nuevos tipos e instanciarlos
  - Chequeo de consistencia {cosas como la concatenación de strings de diferentes tamaños no sería posible con tipos elementales, debido a que su composición es diferente}
- Ventajas:
  - Legibilidad: elección apropiada de nuevos nombres
  - Modificabilidad: solo se cambia en la definición
  - Factorización: Se usa la cantidad de veces necesarias
  - La instanciación de los objetos en un tipo dado implica una descripción abstracta de sus valores. Los detalles de la implementación solo quedan en la definición del tipo
  - Estructura jerárquica de las definiciones de tipos: proceso de refinamiento
    {se obtiene una definición del dato más específico, como se ve en la imagen}

```
Record Persona {
  String nombre,apellido;
   int edad;
   ……
 }
Persona vecino= new Persona(…);
vencinos = array[1..10] of Persona;
```

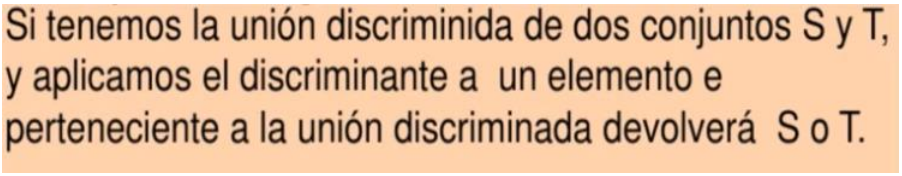
Constructores

Los constructores son los que permiten definir datos definidos por el usuario.

- Producto Cartesiano: es el producto entre conjuntos de diferentes tipos. Produce registros (Pascal) o struct (C).
- Correspondencia Finita: un conjunto de valores en un conjunto de dominio. Son los vectores.
  - Es una función de 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)

- Unión y unión discriminada: la unión de uno o mas tipos, es la disyunción de los tipos dados. Se trata de campos mutuamente excluyente (uso uno o el otro), no pueden estar al mismo tiempo con valores.
    - La unión discriminada se agrega un descriptor (enumerativo) que me permite saber con quien estoy trabajando y acceder correctamente a lo que tengo que acceder. Básicamente manipulo el elemento según el valor del discriminante, es una mejora de la unión.
    - La unión es insegura, ya que en este ejemplo, si el dato almacenado es un short int, si quisiese acceder al otro dato generaría un error, para evitar esto, se utiliza el discriminante
    - El chequeo de tipos debe hacerse en ejecución



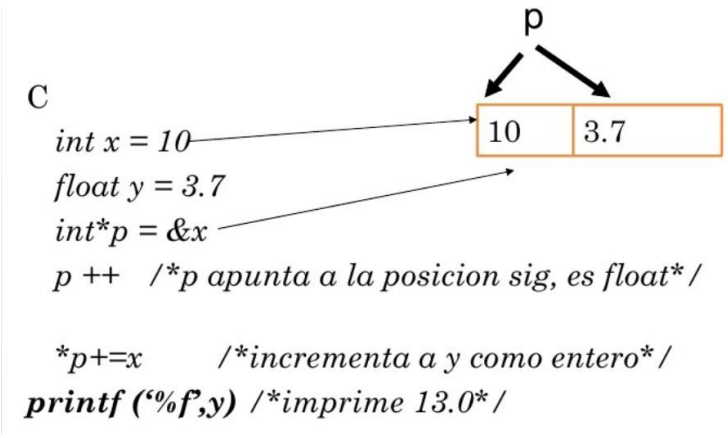
- Recursión: un tipo de dato recursivo T se define como una estructura que puede contener componentes de tipo T. Un ejemplo son las listas.
    - Define datos agrupados
        - Cuyo tamaño puede crecer arbitrariamente
        - Cuya estructura puede ser arbitrariamente compleja
    - Los lenguajes soportan la implementación de tipos de datos recursivos a través de los punteros.
    - Un puntero es una referencia a un objeto
    - Una variable puntero es una variable cuyo r-valor es una referencia a un objeto

Punteros

- No necesariamente tienen nombre, permitiendo conectar muchos ítems sin tener un nombre explicito para todos ellos.
- Los punteros permiten que el dato sea puesto en varias estructuras sin necesidad de duplicarlo.
- Los punteros poseen un acceso a bajo nivel (cerca de la maquina)
- Por acceder a bajo nivel, pueden hacer inseguros los programas
- Son un mecanismo muy potente para definir estructuras recursivas
- Amplian rango de celdas de memoria y amplian los tipos de datos que pueden tener las estructuras.
- Los punteros constan de un l-valor r-valor r-valor variable apuntada
- Desreferenciación implícita: accedo al dato puntual (r-valor de la variable apuntada)

INSEGURIDAD DE LOS PUNTEROS

1. Violación de tipos
    - la imagen explica mejor



2. 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

3. Punteros no inicializados
    - Peligro de acceso descontrolado a posiciones de memoria
    - Soluciones:
        - Nil en pascal
        - Void en C/C++
        - Null en ADA y Python
4. Punteros y uniones discriminadas

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

    - Me permite acceder a algo que no debo
    - En el caso de C, este es el mismo efecto que causa la aritmética de punteros. Para resolver este problema asociado con los punteros Java elimina la noción de puntero explicito completamente.

5. Alias
    -

```
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
```

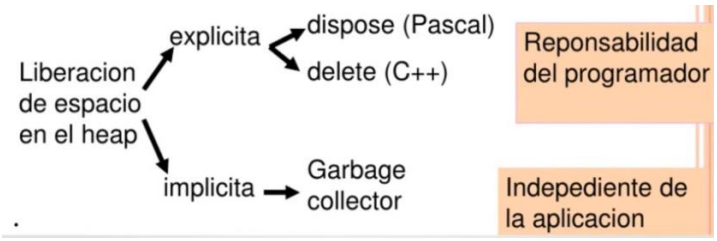
- Si uno cambia su valor se ve reflejado en el otro. Sucede lo mismo si libre y esto genera el punto 2.

6. 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

Manejo de memoria

- Si los objetos en el 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

Liberación de memoria



- Requiere o no la intervención del usuario
- Reconocimiento de que porción de memoria es basura
- Explicita
    - El reconocimiento de la basura recae en el programador, quien notifica al sistema cuando un objeto ya no se usa
    - No garantiza que no haya otro puntero que apunte a esa dirección

  Ejemplos de esto serian el dispose en pascal, el cual es una forma explícita de liberación de memoria

- Implícita
    - El sistema, durante la ejecución tomará la decisión de descubrir la basura por medio de un algoritmo de recolección de basura (garbage collector)
    - Importante para los lenguajes que utilicen frecuentemente variables dinámicas (LISP, Python)
    - Se ejecuta durante el procesamiento de las aplicaciones
    - Debe ser muy eficiente.