Principios de Lenguajes de Programación

Tipo de Datos: Arreglos

Facultad de Informática
Universidad Nacional del Comahue

Primer Cuatrimestre

Arreglos

- Un ***arreglo*** es una agregación de elementos de datos homogéneos, en los cuales cualquier elemento individual se identifica por su posición, relativa al primer elemento
  - Una referencia a un elemento usualmente incluye variables subíndices
  - Requiere de cálculo en tiempo de ejecución para determinar la locación de memoria referenciada

Arreglos

- Consideraciones de Diseño:
  - ¿Qué tipos son válidos como subíndices?
  - ¿Se controla el rango de las expresiones que hacen referencias a elementos?
  - ¿Cuándo se ligan los rangos de subíndices?
  - ¿Cuándo se realiza su locación?
  - ¿Cuál es el número máximo de subíndices?
  - ¿Se inicializan los objetos de datos arreglos?

Arreglos-índices

- Subindizar es mapear desde índices a elementos

  `array_name(index_value_list) → un elemento`

- Sintaxis de subíndices:
  - FORTRAN, PL/I, Ada: usan paréntesis
  - La mayoría de los otros lenguajes usan corchetes

Vectores (array de una dimensión)

- Atributos
  - Número de componentes: uno p/c dim.
  - Tipo de dato de cada elemento
  - Subíndice usado para seleccionar cada elemento.

Ej Pascal

V: array[-5..5] of real; rango

Declaración en C

float a[10]; límite superior

Arreglos

- Tipos de Subíndices:
  - FORTRAN, C – sólo enteros
  - Pascal – cualquier tipo ordinal (integer, boolean, char, enum)
  - Ada – enteros o enumerados (con boolean y char)
  - Java – sólo tipos enteros
- Verificación de Rango:
  - Sin control: C, C++, Perl, Fortran
  - Controlan: Java, ML, C#
  - Control por defecto:
    - Pero puede deshabilitarse: Ada

Arreglos: categorías

- Categorías de Arreglos
  (basado en ligadura de subíndices y ligadura al almacenamiento)
  - Estático
  - Dinámico en pila limitado (Fixed stack-dynamic)
  - Dinámico en pila (Stack-dynamic)
  - Dinámico en heap fija (Fixed heap-dynamic)
  - Dinámico en heap (Heap-dynamic)

Arreglos: categorías

- Estático
  - Rango de subíndices y almacenamiento estático (antes de tpo de ejecución)
    - FORTRAN 77, algunos arreglos en Ada
    - C/C++ declarados *static*
  - Ventajas:
    - Eficiencia en la ejecución (no hay alocación ni desalocación)

Arreglos: categorías

- Dinámico en pila limitado *(Fixed stack dynamic)*
  - Rango de subíndices ligado estáticamente
  - Almacenamiento ligado en la elaboración de la declaración (en ejecución)
  - Ejemplos:
    - Locales de Java, locales de C no *static*
  - Ventajas: eficiencia en el uso del espacio de almacenamiento
    - Usa mismo espacio en dos subprogramas (no simultáneos)

Arreglos: categorías

- Dinámico en pila (stack dynamic)
  - Rango de subíndices y almacenamiento ligado en la elaboración de la declaración (en ejecución)
  - Ejemplo: bloques de declaraciones de Ada

```
declare
     N : Integer;
     STUFF : array (1..N) of FLOAT;
begin
...
end;
```

  - Ventajas: flexibilidad (el tamaño no se conoce sino hasta uso)

Arreglos: categorías

- Dinámico en heap (heap dynamic)
  - Rango de subíndices y ligadura de almacenamiento son dinámicos (no se fijan)
  - Ejemplo: FORTRAN 90

```
INTEGER, ALLOCATABLE, ARRAY (:,:) :: MAT
```

    - Declara a MAT dinámica de dos dimensiones

```
ALLOCATE (MAT (10,NUMBER_OF_COLS))
```

    - Aloca MAT con 10 fijas y N (NUMBER_OF_COLS) columnas

```
DEALLOCATE MAT
```

    - Dealoca MAT

Arreglos: categorías

- Dinámico en heap (continúa)
  - En APL, Perl, y JavaScript, crecen o decrecen por opción:
  - Crecimiento:
    - Implícito: asignaciones de elementos más allá del último elemento actual
    - Explícito: con funciones predefinidas
  - Decrecimiento:
    - Implícitamente
  - Ejemplo:
    - Perl, creación de arreglo de tamaño 5: *@list = (1, 2, 4, 7, 10);*
    - Crecimiento: *push(@list, 13, 17) eq. (1, 2, 4, 7, 10, 13, 17)*

Arreglos: categorías

- Dinámico en heap (continúa)
  - En Java, todos los arreglos son objetos (heap-dynamic)
    - Una vez creados mantienen rango y subíndice
  - C# brinda arreglos dinámicos en heap limitados
  - C# también ArrayList class
    - heap-dynamic
    - Los objetos se crean sin elementos:

```
ArrayList intList = new ArrayList( );
```

    - Se agregan elementos con método *add*:

```
intArray.Add(nextOne);
```

Arreglos: categorías

- Dinámico en heap (continúa)
  - C arreglos usan *malloc* y *free* (standard library).
  - C++: arreglos usan *new* y *delete*.
    - No se controlan rangos de subíndices, es fácil crear arreglos dinámicos
    - Uso generalizado de punteros

Arreglos: categorías resumen

	SUBSCRIPT	ALLOCATION
STATIC	STATIC	STATIC
FIXED STACK-DINAM	STATIC	TPO ELAB DECLARAC.
STACK-DINAM	DINAM. TPO ELABOR.	DINAM. TPO ELABOR.
HEAP-DINAM	DINAMI.	DINAM

FIXED HEAP-DYNAM	=FIXED STACT-DINAM <> ALM HEAP Y NO STACK <> ALM. CDO EL PGM LO REQUIERE (TPO EJEC)

Arreglos

- Cantidad de subíndices
  - FORTRAN I hasta 3 (tres)
  - FORTRAN 77 hasta 7 (siete)
  - Otros: sin límite establecido

Arreglos: inicialización

- Inicialización de arreglos
  - Us. lista de valores: en el arreglo en el orden de los elementos tal como se almacenan en memoria
  - Ejemplos:
  - FORTRAN - (DATA statement), o valores en `/ . . . /` en la declaración
  - C y C++: valores entre llaves, cantidad al compilador

```
Integer List(3)
Data List /0, 5, 5/
int stuff [ ] = {2, 4, 6, 8};
char name [ ] = "freddie"; // compiler puts the '\0'
// an array of strings:
char *names [ ] = {"bob", "jake", "darcie"};
```

Arreglos: inicialización

- Inicialización de arreglos: ejemplos
  - Ada – se pueden indicar posición de los valores

```
List : array (1..5)of Integer := (1, 3, 5, 7, 9);

Bunch : array (1..5)of Integer :=

    (1 => 3, 3 => 4, others => 0);
```

  - Pascal no tiene inicialización
  - Java similar a C

```
Char *names[]= {“juan”,”pedro”,”maria”};
```

Arreglos: operaciones

- Consideración: operaciones que modifican a la estructura (arreglo) como una unidad
  - APL: provee muchas op.
  - Ada: Asignación,
    - La parte derecha puede ser constante, agregada o nombre de arreglo
  - Concatenación,
    - Para todos los arreglos unidimensionales
  - Relacionales
    - Unicamente igualdad y desigualdad

Arreglos: operaciones

- Consideración: operaciones que modifican a la estructura (arreglo) como una unidad
  - APL: muchas operaciones
    - N ← 4 5 6 7 (asigna vector N los valores)
    - N + 4
      - (8 9 10 11) vector con los 4 valores sumados
    - +/N
      - Imprime la suma de los elementos de N, 22.

Arreglos: operaciones

- Consideración: operaciones que modifican a la estructura (arreglo) como una unidad
  - FORTRAN 95
    - Elemental. Operaciones entre pares de elementos
      - Asignación, operadores aritméticos, relacionales y lógicos funcionan en arreglos de distintas formas y tamaños.
    - Intrínsecas (subprogramas) para una gran variedad de operaciones de arreglos
      - Multiplicación de matrices, producto vectorial, etc.

Arreglos: evaluación

- Arreglos
  - Virtualmente en todos los lenguajes
  - Casi lo mismo desde FOTRAN 1
  - Novedades
    - Arreglos dinámicos
    - Inclusión de otros tipos ordinales como subíndices

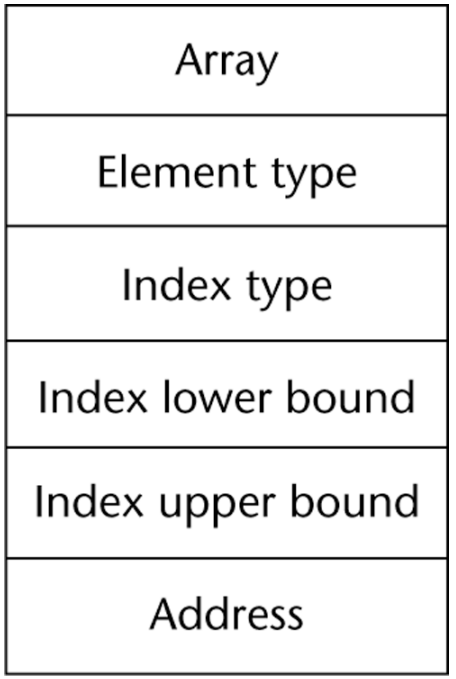
Arreglos: implementación

- Compilador debe generar código para permitir acceso a los elementos del arreglo
  - Ejecutan código en tiempo de ejecución para producir las direcciones de los elementos
  - Ejemplo de acceso uni dimensional:

`address(list[k]) = address (list[0])+ (`*K*element_size*`)`

`address(list[k]) = address (list[lower_bound])+` *((k-lower_bound) * element_size)*

Arreglos: descriptor en ejecución



Arreglo Unidimensional

Arreglo: localizar elemento

localización(a (i)) = dirección de a (1,1) +

*+ ((número de elementos a izq. de i) * tamaño elem)*

- **Constante**
- ***Variable***

Arreglo: descriptor en ejecución

- Arreglo Unidimensional: implementación
  - **Si no hay chequeo de rango**, sólo se necesita la función de acceso
  - **Si hay chequeo de rango**, requiere un descriptor de tiempo de ejecución
  - **Si los rangos son estáticos,** se incorporan los valores del rango en el código de chequeo
    - No se necesita descriptor en la ejecución

Bibliografía

- Pratt, Terrance W., Programming Languages: Design and Implementation, cap 4
- Sebesta, Robert, Concepts of Programming Languages, Cap 6