ESTRUCTURAS ESTÁTICAS

ESTRUCTURA DE DATOS

INSTITUTO DE FORMACIÓN TÉCNICA SUPERIOR No18

Leandro E. Colombo Viña / estructura@ifts18.edu.ar

TIPOS DE DATOS ESTRUCTURADOS

TIPOS DE DATOS ABSTRACTOS - TDA

LA VEZ PASADA...

Vimos como manejar arreglos:

- Unidimensionales = **Vectores**
  - Cadena de Caracteres = **String**
- Multidimensionales = **Matrices**

LOS TIPOS DE DATOS ESTRUCTURADOS

Pueden contener más de un componente simple o estructurado a la vez.

Se caracterizan por:

- El tipo o los los tipos de los componentes.
- La forma de la organización
- La forma de acceso a los componentes

LOS TIPOS DE DATOS ESTRUCTURADOS

- array: arreglo homogéneo de acceso directo por índice.
- string: cadena de caracteres, caso particular del array.
- struct: estructura heterogénea, agrupación de datos.
- union: compartir memoria, superposición de datos.
- bits: campos de bits, partición de datos.

EL TIPO 'ARRAY'

Una organización de datos caracterizada por:

- Todos los componentes son del mismo tipo (homogéneo).
- Acceso directo a cada uno de sus componentes a través de un índice.
- Todos sus componentes se pueden seleccionar arbitrariamente y son igualmente accesibles.

```
int v[100], a[100][100];
v[10] = 5; v[i] = 0; v[i++] = 3;
/* Inicializacion de vector y de array */
for (i = 0; i < 100; i++) v[i] = 0;
for (i = 0; i < 100; i++)
        for (j = 0; j < 100; j++) a[i][j] = 0;
```

EL SUBTIPO 'STRING'

Una cadena de caracteres es **un "array" unidimensional de tipo "char" que termina con el carácter nulo (NULL == '\0')**

El lenguaje no tiene operadores para manejar cadenas de caracteres.

Se manejan con funciones de biblioteca **string.h**

EL TIPO 'STRUCT'

La estructura es un *conjunto* de una o más variables, agrupadas u organizadas bajo un único nombre, cuyos miembros pueden ser de tipos diferentes.

La estructura *la define* el programador y es un *nuevo tipo de dato*.

DEFINICIÓN DE UN STRUCT

```
struct identificador_plantilla {
         tipo_dato1  componente1;
         tipo_dato2  componente2;
         ...    ...
         tipo_datoN  componenteN;
};
```

DECLARACIÓN DE UNA VARIABLE STRUCT

```
struct identificador_plantilla identificador_variable;
```

EJEMPLO: ESTRUCTURA PARA PUNTOS EN LA PANTALLA

```c
/* Definición de la estructura */
struct point {
    int x;
    int y;
};

/* Declaración de una variable "coordenada" */
struct point coordenada;

/* Acceso al elemento con sintáxis operador punto */
coordenada.x = 300;
coordenada.y = 200;
```

EJERCICIOS

1. Cree una estructura para manejar coordenadas en el plano.
2. Cree una estructura para manejar fechas.
3. Cree una estrucutra para nombre, apellido y DNI.
4. Cree una nueva estrucutura para manejar los datos de las últimas 2 en una única estructura.

NUEVOS TIPOS: TYPEDEF

Forma de definir un nuevo tipo de datos.

```
/* Definimos un nuevo Tipo */
typedef int entero;

/* Usamos esa nueva definición */
entero numero;
numero = 4;
printf("%d\n", numero)
```

TYPEDEF Y STRUCT

¿Cómo usamos el typedef con los struct?

```
struct alumno {
    char apellido[30];
    char nombre[20];
    float nota1;
    float nota2;
    float nota3;
    float promedio;
};
typedef struct alumno ALUMNO;
```

TYPEDEF Y STRUCT

o más simple...

```
typedef struct {
    char apellido[30];
    char nombre[20];
    float nota1;
    float nota2;
    float nota3;
    float promedio;
} ALUMNO;
```

ALINEACIÓN DE DATOS EN LAS ESTRUCTURAS

La mayoría de los procesadores requieren que los objetos de **n-bytes** se guarden en memoria en direcciones **n*k** (es decir alineados comenzando en direcciones pares)

Los datos no alineados tienen un efecto colateral en el acceso con buses de gran ancho.

ALINEACIÓN DE DATOS EN LAS ESTRUCTURAS

Por default, en general los compiladores agregan "relleno" (pad) en las estructuras para asegurar la apropiada alineación, en especial con los array.

El relleno (Padding) asegura la alineación con el objeto más grande (que es el que tiene mayor requerimiento).

- Reordená para ahorrar memoria.
- Reordená para acceder más rápido.

Ejercicio: Armá una estructura, averiguá cuanto ocupa en memoria con *sizeof* y reordená para ver los cambios.

BONUS TRACK: LOS TIPOS ENUMERATIVOS

En la declaración o definición se establece un conjunto de valores posibles representados por identificadores.

```
enum colores {rojo, azul, amarillo, verde};
enum colores color;
```

- La variable "color" es de tipo "enum colores"
- Los valores posibles son 4 y están identificados por esos nombres que se llaman enumeradores
- El compilador asigna un valor a cada enumerador empezando en cero.

BONUS TRACK: LOS TIPOS ENUMERATIVOS

Por ejemplo:

```
enum dias {domingo, lunes, martes, miercoles, jueves, viernes, sabado};
enum dias dia;
```

El ENUMERADOR domingo VALE 0.

El ENUMERADOR lunes VALE 1.

...

El ENUMERADOR sabado VALE 6.

BONUS TRACK: LOS TIPOS ENUMERATIVOS

Usándolo con *typedef*.

```
/* Definición */
typedef enum {dom, lun, mar, mie, jue, vie, sab} DIA;

/* Declaración de variable e inicialización */
DIA dia0 = lun;
```

BONUS TRACK: LOS TIPOS ENUMERATIVOS

La utilidad de los tipos enumerativos se basa en dos aspectos:

- Mejoran la claridad del programa, por su carácter autodocumentador.
- Obligan al compilador a hacer test de consistencia de tipo.

CONSULTAS

GRACIAS


That's all Folks!