UNIVERSIDAD DEL BÍO-BÍO

Tipos de datos compuestos en C

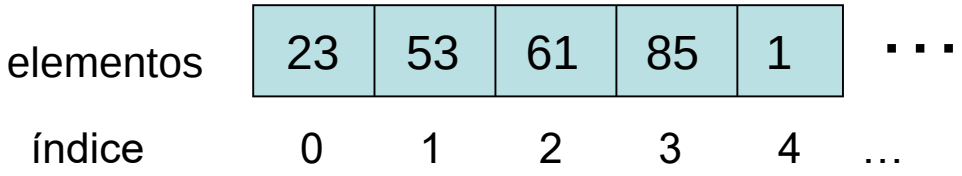
Fernando Santolaya
fsantolaya@ubiobio.cl

Material proporcionado por Luis Gajardo

ARREGLOS

• Un arreglo es una colección ordenada de elementos de un mismo tipo, agrupados en forma consecutiva en la memoria.

• Cada elemento tiene asociado un índice (número entero natural) que le permite al programador acceder a la celda que lo contiene.



• El formato de la declaración es el siguiente:

tipo nombre[tamaño1]…[tamañoN];

• Ejemplos: *char arreglo[100];*
int matriz[10][20];

• Acceso aun elemento en el arreglo:

arreglo[2]= 55;

OJO: C no comprueba los índices al accesar una posición del arreglo. Esto podría dar lugar a errores si el programador no verifica su código.

Ejemplo:

```
char arreglo[100];
char n;
n= arreglo[100];
```

INICIALIZACION DE ARREGLOS

- Existe un formato general para inicializar un arreglo:

  tipo nombre[tamaño1]…[tamañoN]= {lista de valores}

- La lista de valores es una lista de constantes separadas por comas cuyo tipo debe ser compatible con el tipo especificado en la declaración del arreglo.

- Ejemplos:

```
int arr[10] = {1, 2, 3, 4, 5, 6, 7, 8, 9, 10};

double vec[6] = {1.0, 2.0, 3.0, 4.0, 5.0, 6.0};

float datos[ ] = {1.2, 3.4, 5.1};  /*se crea un arreglo de 3 posiciones implícitamente*/
```

Una declaración de la forma : int datos[]; no reserva memoria, por lo tanto se debe utilizar malloc()

```
int f[100] = {0};  /*se inicializa todo el arreglo en cero*/

int h[10] = {1, 2, 3}; /*se inicializa el resto en cero*/
```

CADENAS

- Las cadenas o strings, son en realidad arreglos de caracteres, su declaración es como sigue:

```
char cadena[25];
```

- Una cadena en C, siempre termina con el carácter nulo (‘\0’).

- Los siguientes son ejemplos de declaraciones e inicializaciones de una cadena:

```
char cadena[11] = “Me gusta C”;
```
inicialización implícita

```
char cadena[11] = {‘M’,’e’,’ ’,’g’,’u’,’s’,’t’,’a’,’ ’,’C’,’\0‘}
```
inicialización explícita

0 1 2 3 4 5 6 7 8 9 10

Obs: Los arreglos de caracteres siempre deben considerar un espacio más para el carácter nulo (‘\0’)

MATRICES

- Las matrices se manipulan de la misma forma que los arreglos unidimensionales.

- Un ejemplo de declaración es el siguiente:

```
int mat[3][2] = { {1, 2},
                  {3, 4},
                  {5, 6}};
```

1	2
3	4
5	6

Propuesto: investigar como definir matrices de 3 dimensiones.

ESTRUCTURAS

- Una estructura (struct) es un tipo de datos compuesto que agrupa un conjunto de tipos de datos (no necesariamente iguales) en un único tipo.

- Dado que C no es un lenguaje orientado a objetos, la manipulación de entidades del tipo “persona”, “auto”, “cuenta”, etc. no se puede hacer mediante clases. En cambio, el mecanismo que provee C se denomina estructura.

- A diferencia de una clase, a una estructura no se le pueden asociar métodos.

- El formato de la definición es el siguiente:

```
struct nombreEstructura {
  declaracion de variable 1;
  …
  declaracion de variable N;
};
```

- Para declarar una variable del tipo estructura lo hacemos así:

```
struct nombreEstructura variable;
```

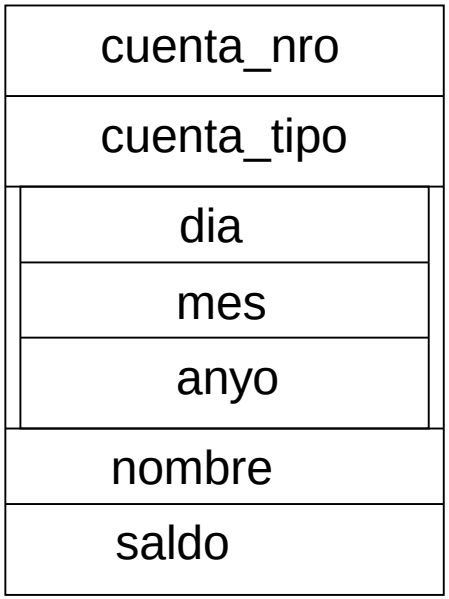
ESTRUCTURAS

- Ejemplo:

```
struct fecha {
  int dia;
  int mes;
  int anyo;
}

struct cuenta {
  int cuenta_nro;
  int cuenta_tipo;
  struct fecha ultimo_pago;
  char nombre[80];
  float saldo;
};
```

cuenta



- Cada uno de los elementos de la estructura se denomina “miembro” o “campo”.

- Los campos pueden ser de cualquier tipo, excepto del tipo void.

ESTRUCTURAS

- Para acceder a un campo dentro de una estructura utilizamos el operador punto “.” de la siguiente forma:

- Ejemplo:

```
struct cuenta c;
c.saldo= c.saldo – 100000;

struct fecha hoy;
...
printf(“%d:%d:%d\n”, hoy.dia, hoy.mes, hoy.anyo);
```

UNIONES

- Corresponde a una posición de memoria que es compartida por una o más variables diferentes, generalmente de distinto tipo en distintos momentos.

- La declaración es la siguiente:

```
union nombreUnion {
  declaracion de variable 1;
  ...
  declaracion de variable N;
}
```

```
union numero {
  int entero;
  float real;
};
```

- Tanto el entero como el real comparten el mismo espacio de memoria. Sólo almacena el valor de uno de sus miembros a la vez.

- El tamaño en bytes de la estructura será el tamaño del mayor de sus campos.

UNIONES

- Ejemplo:

```
union salario {
  int en_pesetas;
  float en_euros;
};

union salario sueldo;
/*inicialmente usamos la moneda en pesetas*/
sueldo.en_pesetas= 250000;
...
/*luego transformamos a euros*/
sueldo.en_euros= sueldo.en_pesetas / 166.386;
```

TIPOS ENUMERADOS

- Un tipo enumerado es similar a las estructuras.
- Sus miembros son constantes de tipo int.
- El formato de la declaración es el siguiente:

  enum nombre {m1, m2, …, mN};

  Ejemplo: enum color {negro, blanco, rojo};

- Por defecto C asigna el número entero a cada elemento, partiendo desde cero.

TIPOS ENUMERADOS

- Ejemplo:

```
#include <stdio.h>

enum diasemana {lunes, martes, miercoles, jueves, viernes, sabado, domingo};

void main () {
  enum diasemana dia;
  for (dia= lunes; dia<= domingo; dia++)
    switch (dia) {
      case lunes:
         printf("Lunes es laboral\n");
         break;
      case martes:
         printf("Martes es laboral\n");
         break;
      ...
      case domingo:
         printf("Domingo es para descansar\n");
         break;
    }
}
```

TIPOS DE DATOS DEFINIDOS POR EL PROGRAMADOR

- C permite la definición, por parte del programador, de nuevos tipos de datos.

- El formato es el siguiente:

typedef tipo_de_variable nombre_nuevo;

- Ejemplo:

```
struct complejo {
  double real;
  double imaginario;
}

struct complejo numero;
```

Usando typedef:

```
typedef struct {
  double real;
  double imaginario;
} COMPLEJO;

COMPLEJO numero;
```