

Práctica de Organización del Computador II

Programación orientada a datos

Primer Cuatrimestre 2024

Organización del Computador II

DC - UBA

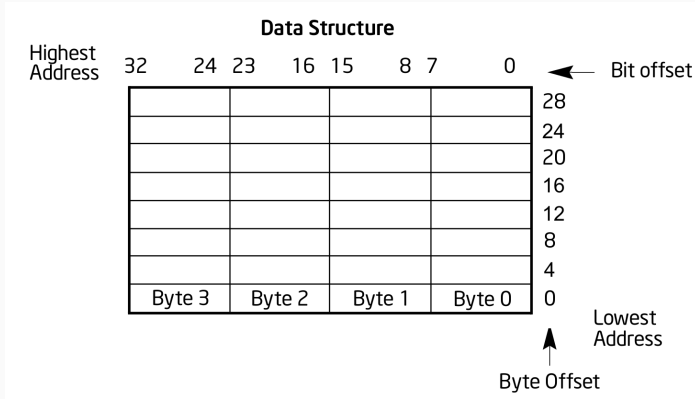
Introducción

La clase de hoy se va a dividir en dos partes:

- Primera parte:
Programación orientada a datos y memoria dinámica.
- Segunda parte:
Labo: programación en C.

Representación de los datos en memoria

Esta es la memoria principal:



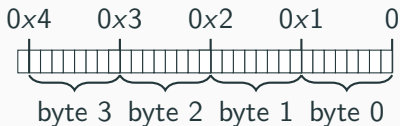
Es un arreglo contiguo y ordenado de bits.

Esta es la memoria principal:



Es un arreglo contiguo y ordenado de bits.

Esta es la memoria principal:



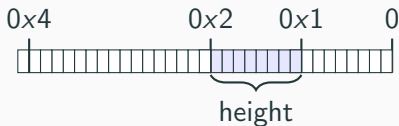
Es un arreglo contiguo y ordenado de bits o bytes.

¿Qué nos interesa saber de la memoria principal?

- Que podemos imaginarla como una secuencia contigua de bits (bytes).
- Que podemos determinar posiciones unívocas en la memoria (direcciones)
- $|dirs| = |Memoria| / |unidad\ direccionable|$.

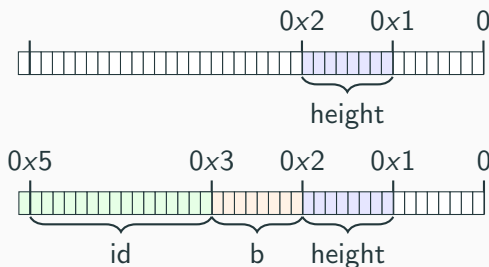
¿Qué nos interesa saber de la memoria principal?

Que nuestros datos e instrucciones van a encontrarse en la memoria principal, a partir de una dirección y ocupando un cantidad acotada de bytes.

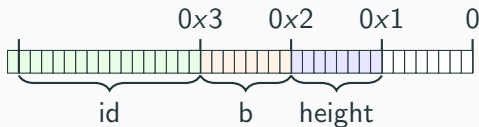


Aquí por ejemplo el entero `height` de 8 bits comienza en el bit 8 y ocupa hasta el bit 15 (posición 1 de memoria).

¿Qué nos interesa saber de la memoria principal?



Si tuviésemos, aparte del entero `height`, un booleano `b` y un arreglo de dos caracteres llamado `id` también podemos suponer que ocupan su lugar en la memoria como secuencia ordenada y contigua de bits.



Veamos que podemos razonar sobre nuestro dato según distintas categorías:

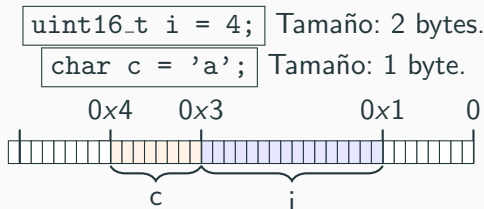
- El identificador de variable (al menos en alto nivel)
- Su dirección y tamaño en memoria
- El valor representado

¿Qué tipos de datos podemos representar y cómo los representamos? Comencemos con el qué:

- Tipos atómicos (`int`, `char`, `float`, `bool`)
- Estructuras (`structs`)
- Arreglos (`arrays`)
- Punteros (`pointers`)

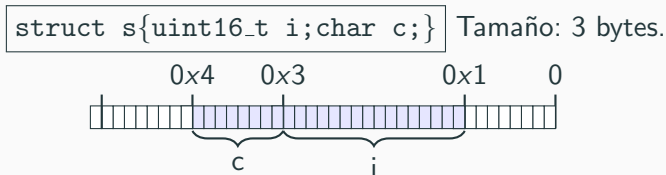
Tipos atómicos(int,char,float,bool):

Los tipos numéricos tienen una longitud fija según su tipo:



Estructuras(structs):

Las estructuras (structs) son una concatenación de otros tipos (atributos) que se pueden acceder a través de su identificador:

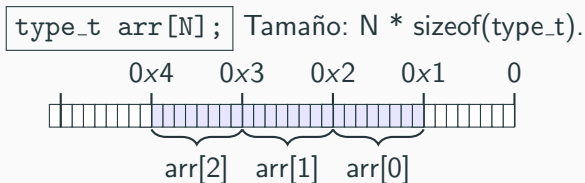


Estructuralmente es igual que el ejemplo anterior, pero su semántica es distinta. Permite hacer lecto-escrituras por atributo.

```
struct s obj; obj.i = 4;
```

Arreglos(arrays):

Las arreglos (arrays) son una concatenación de varias instancias del mismo tipo que se pueden acceder a través de un índice numérico:



Permite hacer lecto-escrituras por índice.

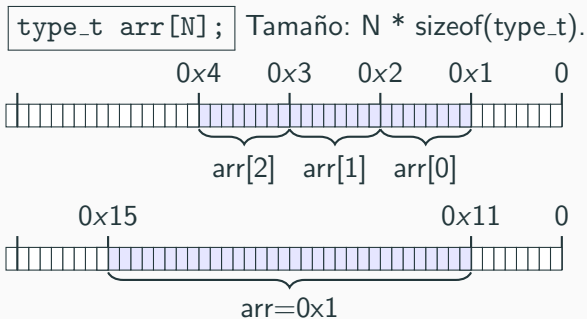
```
char arr[3]; arr[1] = 'a';
```

Arreglos(arrays):

Vamos a ver más adelante que cuando hablamos de arreglos muchas veces nos referimos a:

- La región de memoria donde se encuentran los datos (estructura).
- Y la referencia a la posición en la que comienza ese mismo dato (puntero).

Arreglos(arrays):

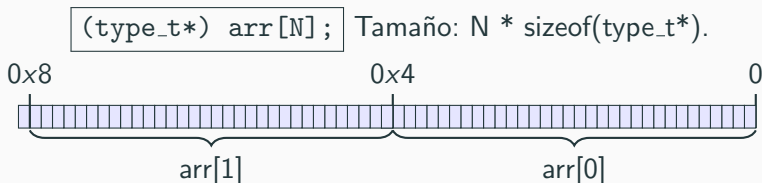


Punteros(pointers):

Los punteros son tipos numéricos que representan una posición en memoria. ¿Qué tamaño deberían tener?

El tamaño de las direcciones de memoria de nuestra arquitectura

Veamos un arreglo de punteros:

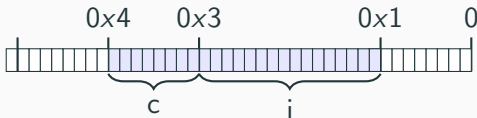


Ya vamos a ver cómo operar con punteros. Lo importante ahora es entender que son un dato numérico cuyo tamaño es igual al de las direcciones de memoria.

Es importante notar que para operar con un dato compuesto (escribir o leer un atributo de un struct o un elemento de un arreglo) debemos:

- Comprender cómo se representa en memoria según su tipo.
- En qué posición de memoria comienza el tipo que lo contiene.
- Cómo calcular el desplazamiento para acceder al dato particular que nos interesa.

```
struct s{uint16_t i;char c;} Tamaño: 3 bytes.
```



Por ejemplo, para implementar la siguiente operación:

```
struct s obj; obj.c = 'a';
```

- Resolvimos la ubicación del atributo(en bytes) como base + desplazamiento: $0x1 + 0x2 = 0x3$.
- Determinamos el tamaño del dato a escribir (1 byte).

¿Consultas?
