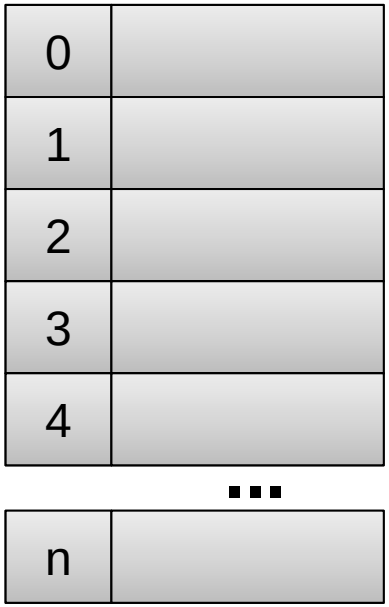
Prácticas de Programación

La memoria

La memoria

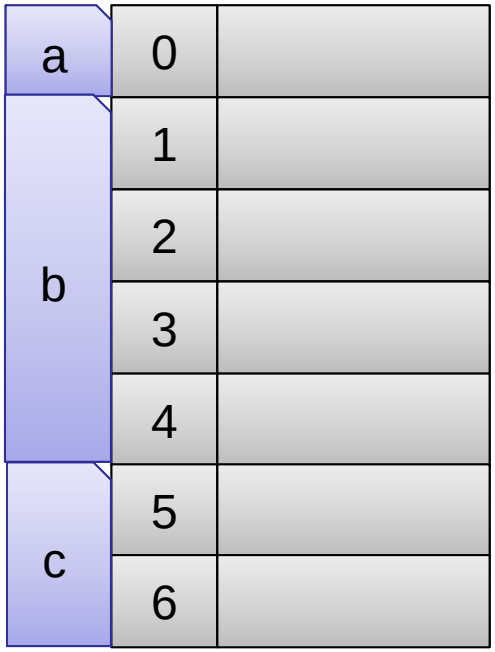
- La memoria es el recurso de la maquina que nos permite almacenar información de forma **temporal.** Es un recurso escaso, y debe ser gestionado correctamente.
- Generalmente, la memoria se representa en forma de vector.
  - Las posiciones se asumen continuas.
  - Las posiciones se enumeren, obteniendo las direcciones, como las casas dentro de una calle. Las direcciones se suelen mostrar en formato Hexadecimal. La dirección de una variable se accede con el operador ***&***.



La memoria

- Cada posición puede almacenar un byte. Por lo tanto, una variable puede ocupar mas de una posición, según su tipo, la máquina y el sistema operativo que estemos utilizando.
- A continuación se muestra un ejemple de la evolución de la memoria con un código simple.

```
char a;
int b;
char c[2]:

a='i'
b=4;
c[0]='h';
c[1]=a;
```

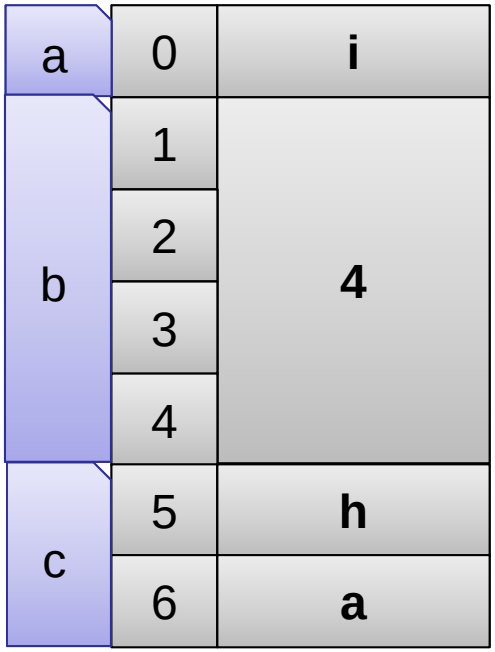


Se asignan posiciones de memoria libres a cada variable. Se asume char (1 byte) e int=float (4 bytes)

La memoria

- Cada posición puede almacenar un byte. Por lo tanto, una variable puede ocupar mas de una posición, según su tipo, la máquina y el sistema operativo que estemos utilizando.
- A continuación se muestra un ejemplo de la evolución de la memoria con un código simple.

```
char a;
int b;
char c[2]:

a='i'
b=4;
c[0]='h';
c[1]=a;
```

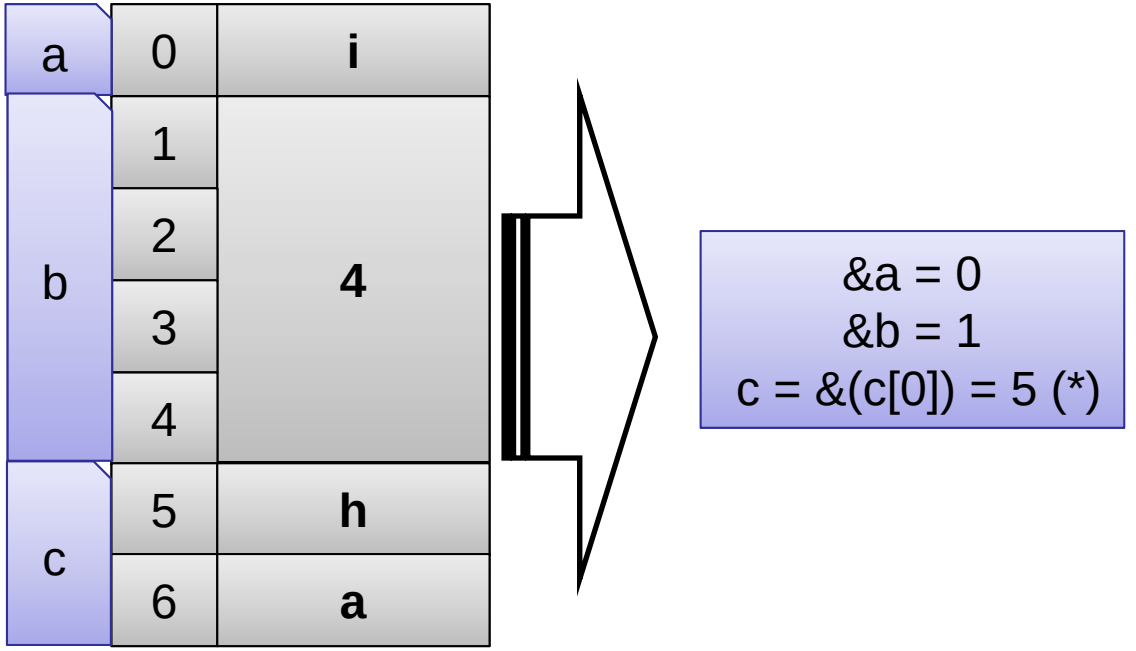


Se cambia el valor del contenido de las posiciones de memoria asociadas a cada variable.

La memoria

- Cuando declaramos una variable, el nombre de esta variable se asigna a la dirección de memoria de su primera posición. Por ejemplo, en el ejemplo anterior, las direcciones correspondientes a las variables declaradas, obtenidas con el operador &, serian:

```
char a;
int b;
char c[2]:

a='i'
b=4;
c[0]='h';
c[1]=a;
```

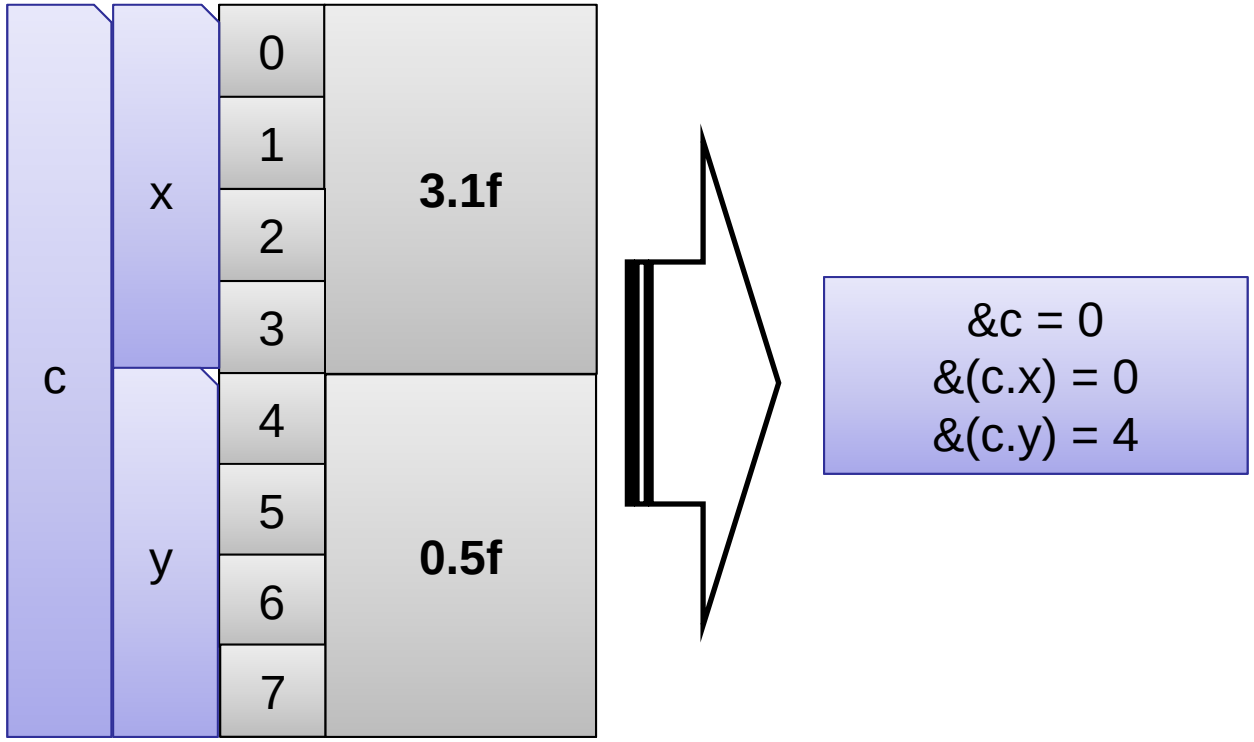


(*) El caso de la variable **c** es ligeramente distinto, lo veremos mas adelante, cuando hablemos de punteros.

La memoria

- Cuando trabajamos con tuplas, la memoria funciona exactamente de la misma forma, siendo el tamaño de la tupla la suma de los tamaños de sus campos(*). Por ejemplo, la tupla *coordenadas*, quedaría de éste modo:

```
struct {
    float x,y;
} coordenadas;

coordenadas c;
c.x=3.1f;
c.y=0.5f;
```



(*) Algunos sistemas ajustan el tamaño múltiples de n bytes, por lo que la tupla puede tener un tamaño ligeramente mayor. Esto se conoce como "*Alineación de memoria*".

La memoria

- Para conocer el tamaño exacto en bytes de un tipo de datos, se utiliza ***sizeof***. Esta función nos retorna el valor exacto, teniendo en cuenta el sistema en el que nos encontramos y el compilador utilizado.
- A continuación se muestran unos cuantos ejemplos que podéis probar. Intentad entender los resultados y de donde sale cada tamaño. Podéis intentar hacer el dibujo de la memoria en cada caso.

```
char c[26];
struct {
    int nCaracteres;
    char caracteres[26];
} linea1;
struct {
    int nCaracteres;
    char caracteres[30];
} linea2;
struct {
    char caracteres[13];
    int nCaracteres;
    char caracteres2[13];
} linea3;
printf("a) El tamaño de un tipo <%s> es %d bytes\n","char",sizeof(char));
printf("b) El tamaño de un tipo <%s> es %d bytes\n","int",sizeof(int));
printf("c) El tamaño de un tipo <%s> es %d bytes\n","float",sizeof(float));
printf("d) El tamaño de un tipo <%s> es %d bytes\n","char[26]",sizeof(c));
printf("e) El tamaño de un tipo <%s> es %d bytes\n","linea1 => [ int + char[26] ]",sizeof(linea1));
printf("f)  El tamaño de un tipo <%s> es %d bytes\n","linea2 => [ int + char[30] ]",sizeof(linea2));
printf("g) El tamaño de un tipo <%s> es %d bytes\n","linea3 => [ char[13] + int + char[13] ]",sizeof(linea3));
```

La memoria

Autoaveluación:

1. Dibuja la memoria para el siguiente código:

```
int b;
char a;

a='A'
b=(int)a;
```

2. Dibuja la memoria para el siguiente código :

```
float a;

a=1.23f;
```

3. Dibuja la memoria para el siguiente código :

```
float a[2];

a[0]=1.23f;
a[1]=2.0f;
```

La memoria

Autoevaluación:

4. Dibuja la memoria para el siguiente código:

```
coordenada c[2];
c[0].x=10.0f;
c[0].y=20.0f;
c[1].x=30.0f;
c[1].y=40.0f;
```

5. Dibuja la memoria para el siguiente código:

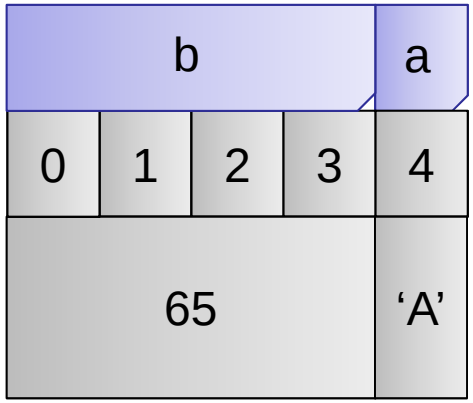
```
Struct {
    coordenada p1,p2;
} vector;
vector  v1;
v1.p1.x=10.0f;
v1.p1.y=20.0f;
v1.p2.x=v1.p1.y+10.0f:
v1.p2.y=v1.p1.x+30.0f:
```

La memoria

Autoevaluación (Respuesta):

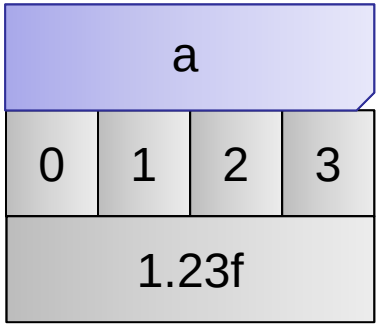
1. Dibuja la memoria para el siguiente código:

```
int b;
char a;

a='A'
b=(int)a;
```



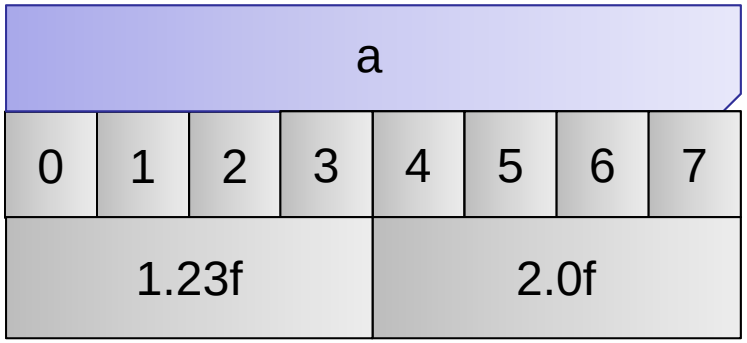
2. Dibuja la memoria para el siguiente código:

```
float a;

a=1.23f;
```



3. Dibuja la memoria para el siguiente código:

```
float a[2];

a[0]=1.23f;
a[1]=2.0f;
```

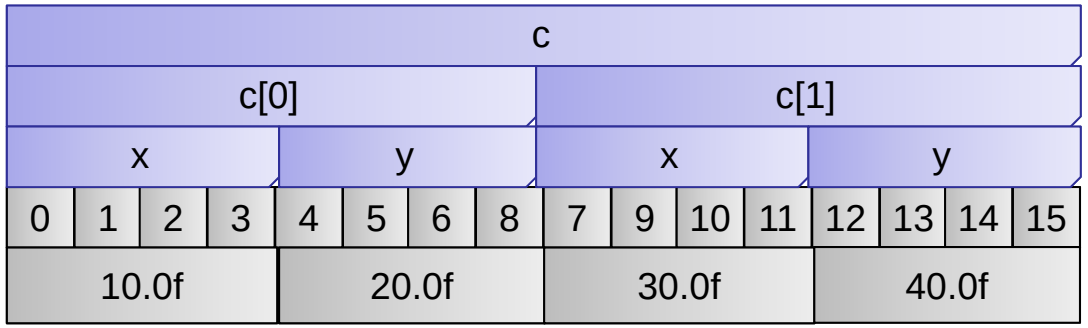


La memoria

Autoevaluación (Respuesta):

4. Dibuja la memoria para el siguiente código:

```
coordenada c[2];
c[0].x=10.0f;
c[0].y=20.0f;
c[1].x=30.0f;
c[1].y=40.0f;
```



5. Dibuja la memoria para el siguiente código :

```
Struct {
        coordenada p1,p2;
} vector;
vector  v1;
v1.p1.x=10.0f;
v1.p1.y=20.0f;
v1.p2.x=v1.p1.y+10.0f:
v1.p2.y=v1.p1.x+30.0f:
```