

CURSOS
INTERSEMESTRALES



PROTECO

Apuntadores en Lenguaje C

Apuntadores en Lenguaje C.

- Unidad Aritmético Lógica
- Unidad de Control
- Unidad de Memoria
- Unidad de Entrada y Salida

Unidad de Memoria:

En esta parte de la computadora se almacena información de entrada y de salida. Cuando se ejecuta un programa, este reside en la memoria, y en ella están los datos necesarios para el proceso y así mismo aquí se aloja el resultado.



Memoria RAM.

- Unidad de almacenamiento primario.
- Almacenamiento temporal.
- Memoria volátil.
- Está dividida en localidades (localidad de memoria).
 - El tamaño de las localidades es igual.
 - Las localidades poseen una dirección de memoria única



Mapa de Memoria.

Capacidad de memoria

Direcciones de memoria

1 1 0 0 1 1 0 1

Tamaño de la memoria: 1 Byte

Tamaño de la memoria



PROTECO

Definición.

Un puntero es una variable que contiene la dirección de memoria de un dato o de otra variable que contiene al dato. Quiere esto decir, que el puntero **apunta al espacio físico** donde está el dato o la variable.

¿A qué apuntan?

Un puntero puede apuntar a un objeto de **cualquier tipo**, como por ejemplo, a una estructura o a una función. Los punteros se pueden utilizar para referenciar y manipular estructuras de datos, para referenciar bloques de memoria asignados dinámicamente y para proveer el paso de argumentos por referencia en las llamadas a funciones.

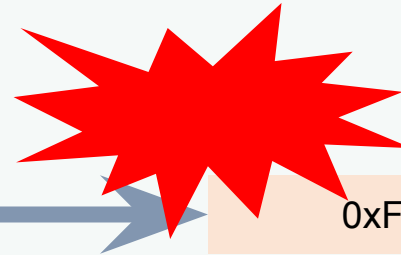


¡ATENCIÓN!

Cuando se trabaja con apuntadores son frecuentes los errores debidos a la creación de punteros que apuntan a alguna parte inesperada, produciéndose una violación de memoria. Por lo tanto, debe ponerse la máxima atención para que esto no ocurra, inicializando adecuadamente cada uno de los punteros que utilizemos.

Apuntador

0xFFA5445FA



PROTECO

Declaración de apuntadores.

La sintaxis es:

tipo *nombreApuntador;

Ejemplo 1:

```
#include <stdio.h>
main(){
    int *ptr=NULL;
}
```

Intersemestral Junio 2015

Operadores: * &

&: operador de dirección.

*: operador de indirección.

El operador unitario &, devuelve como resultado la dirección de su operando y el operador unitario *, toma su operando como una dirección y nos da como resultado su contenido. Es decir, * regresa el contenido de la variable que es apuntada por el operando.

Especificador de formato %p se utiliza para hacer referencia a las direcciones de memoria. O para su representación en hexadecimal: %x.



Ejemplo 2:

```
#include <stdio.h>
```

```
main(){
```

```
    int a=10;
```

```
    int *ptr;
```

```
    ptr=&a;
```

```
    printf("La variable a está en: %p\n",&a);
```

```
    printf("ptr almacena la dirección: %p\n",ptr);
```

```
    printf("La variable a tiene: %d\n",*ptr);
```

```
}
```

0x7ffd464

0x7ffd468

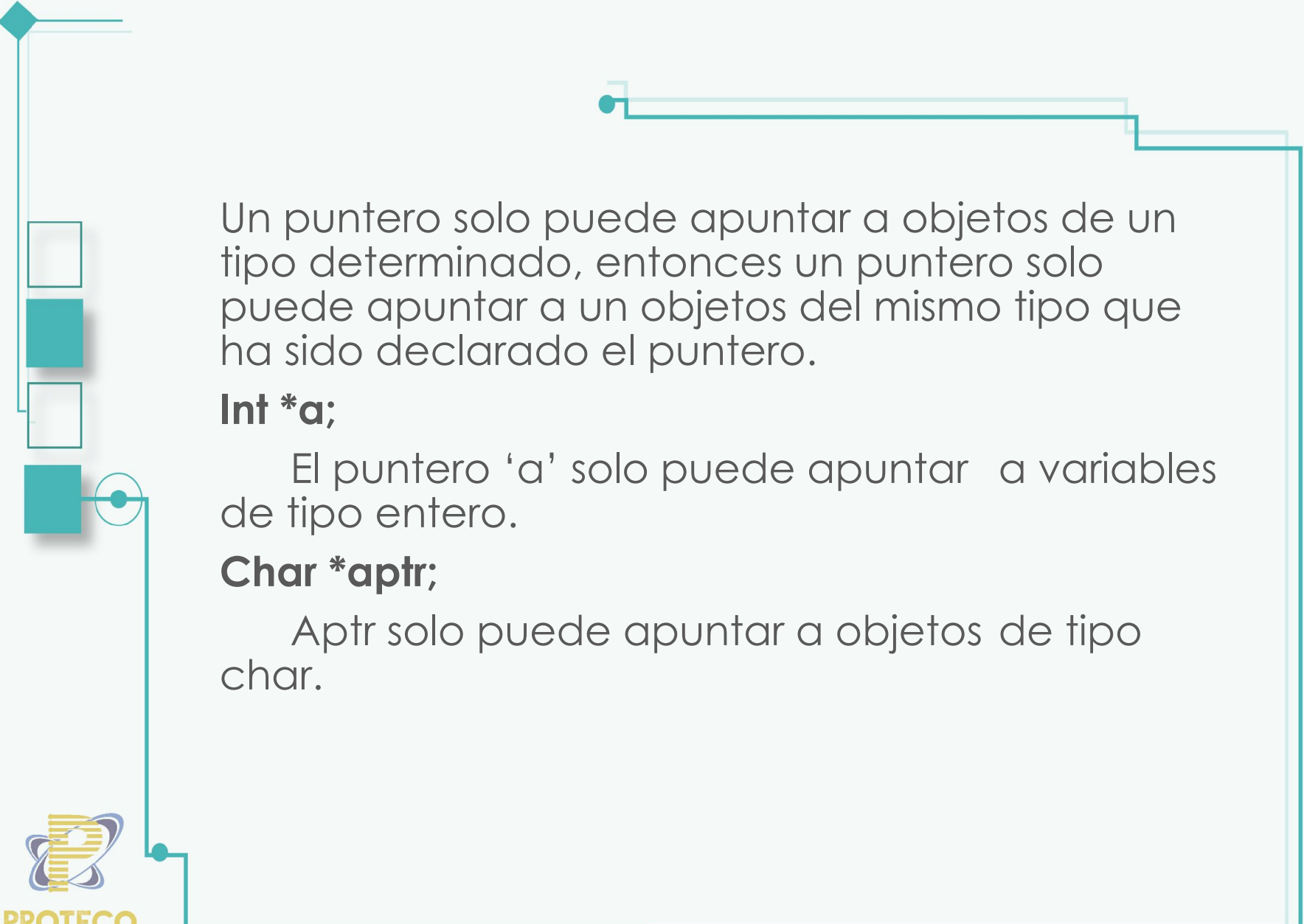
10

0x7ffd464

a

ptr





Un puntero solo puede apuntar a objetos de un tipo determinado, entonces un puntero solo puede apuntar a un objetos del mismo tipo que ha sido declarado el puntero.

Int *a;

El puntero 'a' solo puede apuntar a variables de tipo entero.

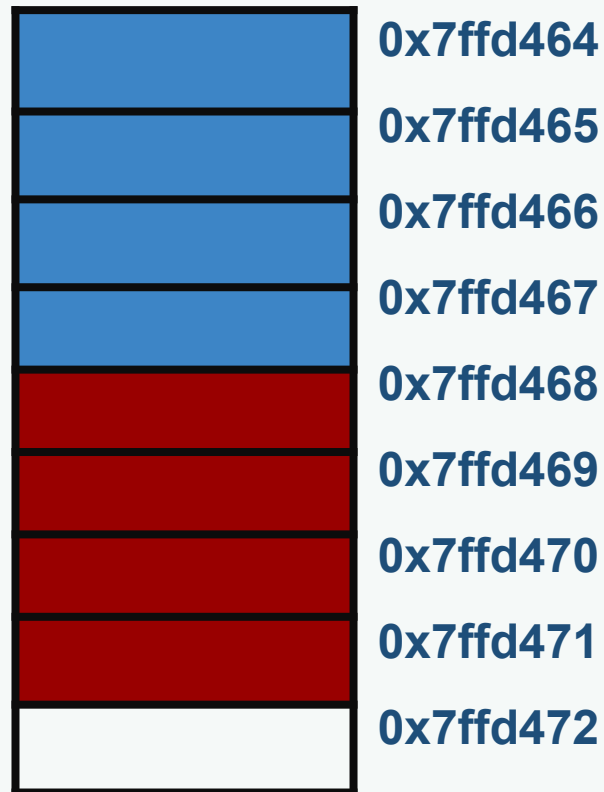
Char *aptr;

Aptr solo puede apuntar a objetos de tipo char.

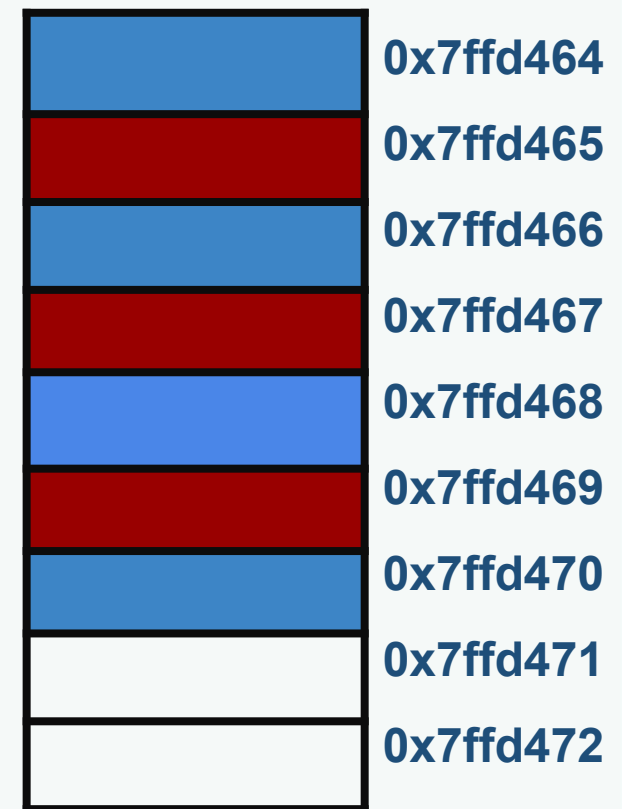


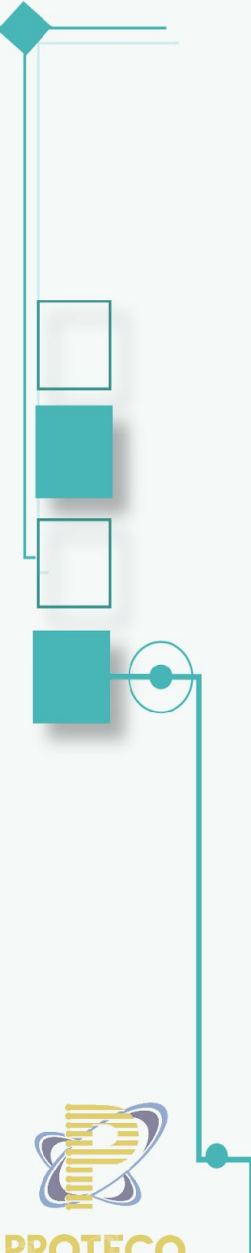
Importancia del tipo de apuntador.

int



char





Type	Size
char, unsigned char, signed char	1 byte
short, unsigned short	2 bytes
int, unsigned int	4 bytes
long, unsigned long	4 bytes
float	4 bytes
double	8 bytes
long double	8 bytes

size_t sizeof(type)

OPERACIONES CON PUNTEROS.

Si p es un puntero, $p++$ **reasigna** p para que apunte al siguiente elemento de los que apunta p ; es decir, C sabe que tiene que avanzar un número de bytes igual al tamaño de un objeto de los que apunta p . La aritmética de direcciones es una de las principales virtudes de C.





Ejercicio:

Hacer un programa que demuestre que no es lo mismo sumar un uno a un apuntador tipo entero que a uno de tipo carácter.



Ejemplo 3:

```
#include <stdio.h>
main(){
    char a=10;
    char b=9;
    char *ptr;
    ptr=&a;
    printf("La variable a esta en: %p\n",&a);
    printf("La variable b esta en: %p\n",&b);
    printf("La variable ptr esta en: %p\n",&ptr);
    printf("La variable ptr almacena: %p\n",ptr);
}
```



0x7ffd464

10

a

0x7ffd465

9

b

0x7ffd466

0x7ffd464

ptr

0x7ffd467

0x7ffd468

0x7ffd469

0x7ffd469

0x7ffd469



PROTECO

Operación de asignación.

A un puntero se le puede asignar otro puntero.

Ejemplo 4:

```
#include <stdio.h>
```

```
main(){
```

```
    int a=10;
```

```
    int *p1;
```

```
    int *p2;
```

```
    p1=&a;
```

```
    p2=p1;
```

```
    printf("La variable a tiene: %d, %d, %d\n",*p1,*p2,a);
```

```
}
```



0x7ffd464

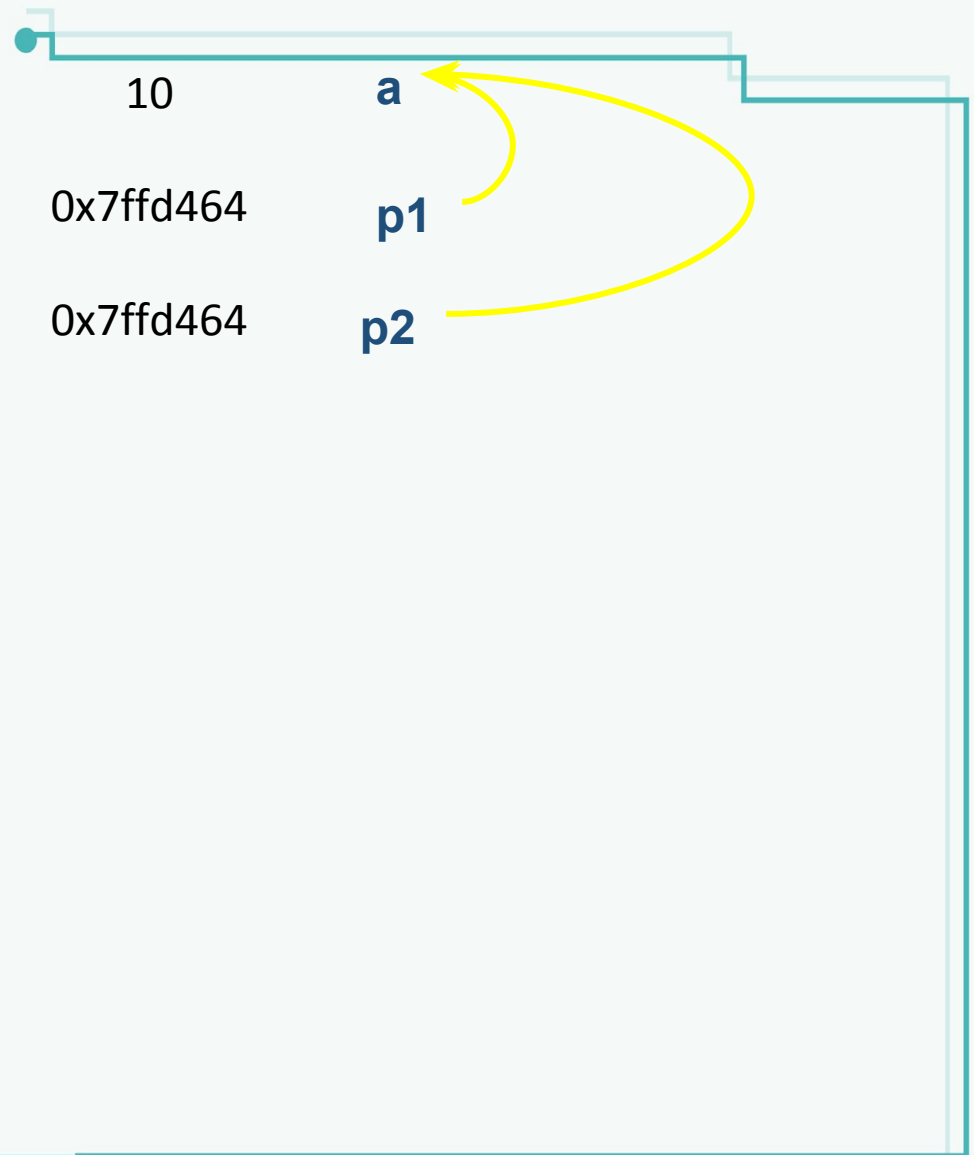
0x7ffd468

0x7ffd472

0x7ffd476

0x7ffd480

0x7ffd484



Operaciones aritméticas.

A un puntero se le puede sumar o restar un entero.

Ejemplo 5:

```
#include <stdio.h>
main(){
    int a[10]={1,2,3,4,5,6,7,8,9,10};
    int *p1=&a[0];
    int *p2=&a[5];
    printf("p1 apunta al: %d\n",*p1);
    printf("p2 apunta al: %d\n",*p2);
    p1=p1+5;
    printf("p1 ahora apunta al: %d\n",*p1);
}
```



a[10]

34	12	63	34	15	16	27	58	19	10
0	1	2	3	4	5	6	7	8	9

IDENTIFICADORES

p1

p2



PROTECO

Comparación de apuntadores.

❖ `*ptr1==*ptr2`

❖ `ptr1==ptr2`

a[10]

1	2	3	4	5	6	7	8	9	10
0	1	2	3	4	5	6	7	8	9

`ptr1=&a[]`

`ptr2=&a[]`



Jerarquía de operadores.

Los operadores unitarios * y & tienen prioridad mayor que los operadores aritméticos + y - e igual prioridad que ++ y --.

Ejemplo 6:

```
int a[10]={1,2,3,4,5,6,7,8,9,10}, variable;  
int *p1=&a[0];  
int *p2=&a[5];  
variable=*p1+1;
```



a[10]



variable=*p1+1

variable=*(p1+1)

p1=&a[]

> *p1=

> *p1+=2 // *p1=*p1+2

(*p1) -- ; // Decrementa el valor

a[]=*p2++; // Asigna a a[] el valor apuntado por p2, p2 cambia



PROTECO

Punteros Genéricos

Un puntero a cualquier objeto no constante se puede convertir a tipo void *. por eso, un puntero de tipo void * recibe el nombre de puntero genérico. Por ejemplo:

```
void *p;  
int a, *q=&a;  
p=q;
```



Puntero Nulo

En general, un puntero se puede inicializar como cualquier otra variable, aunque los únicos valores Significativos son NULL o la dirección de un objeto previamente definido. NULL es una constante definida en el fichero stdio.h así:

```
#define NULL ((void *)0)
```

El lenguaje c garantiza que un puntero que apunte a un objeto válido nunca

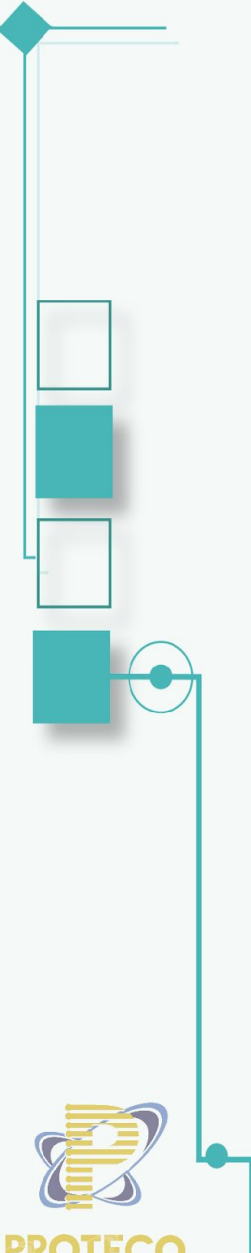
tendrá un valor cero. El valor cero se utiliza para indicar que ha ocurrido un error; en otras palabras, que una determinada operación no se ha podido realizar. Por ejemplo la función gets cuando lee la marca de fin de fichero retorna un puntero nulo, indicando así que no hay más datos para leer.



Relación Array-Puntero

```
#include <stdio.h>
void main() {
    int lista[] = {24, 30, 5, 45, 34};
    int ind;
    for (ind = 0; ind < 5; ind++)
        printf("%d ", lista[ind]);
}
```





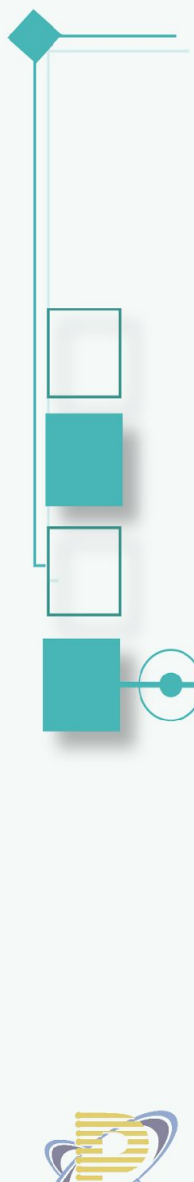
```
#include <stdio.h>
void main(){
    int, lista[] = {24, 30, 15, 45, 34};
    int ind;
    int *plista = &lista[0];
    for (ind = 0; ind < 5; ind++)
        printf("%d",* (plista+ind) );
```



Apuntador a cadena:

```
char *nombre = "Jorge Bañuelos";  
printf ("%s", nombre);
```





```
#include <stdio.h>
int longcad(char *);
main(){
    char *cadena = "abcd";
    printf("%dn", longcad(cadena));
}
int longcad(char *cad){
    char *p= cad;
    while(*p !='\0')
        p++;
    return (p-cad);
}
```

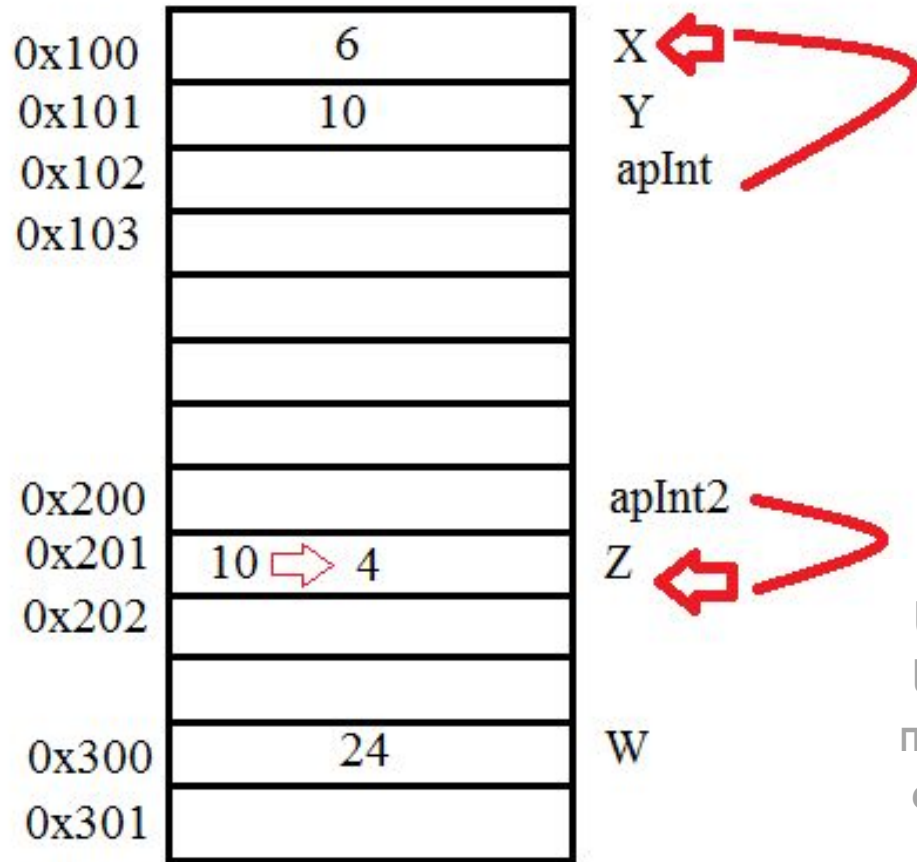


EJEMPLO 01

```
1  #include <stdio.h>
2  main(){
3      int w,x,y;
4      int z=10;
5      int* apInt;
6      int* apInt2;
7      apInt= &x;
8      *apInt=6;
9      apInt2= &z;
10     *apInt2=4;
11     y=*apInt2 + *apInt;
12     w=*apInt2 * *apInt;
13 }
```



Mapa de memoria



NOTA**

Los apuntadores y los enteros ocupan más de una celda. Se elaboró el mapa de esa forma para ser más entendible

EJERCICIO 01

```
1  #include <stdio.h>
2  ▼ main(){
3      int x=10,y=2;
4      float f1=3.14,f2=1.5;
5      int *apInt, *apInt2;
6      float *apFloat;
7      apInt= &y;
8      apInt2= &y;
9      apFloat=&f2;
10     *apFloat= 45;
11     *apFloat= *apInt;
12     *apInt2 *=2;
13 }
14
```

