LICENCIATURA EN INFORMATICA

PROGRAMADOR UNIVERSITARIO

INGENIERÍA EN INFORMÁTICA

PROGRAMACIÓN

Unidad 6 - Parte 1: Variables puntero. Operadores. Punteros y direcciones.

2022

Lic. Mariela A. Velázquez

Introducción

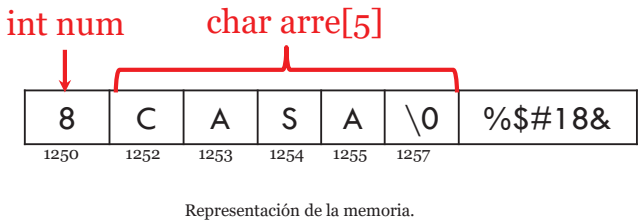
Cada vez que se declara una variable C, el compilador establece un área de memoria para almacenar el contenido de la variable.

Por ejemplo cuando se declara una variable int, el compilador asigna dos bytes de memoria. El espacio para esa variable se sitúa en una posición específica de la memoria, conocida como dirección de memoria . Cuando se referencia, es decir hace uso de la variable, el compilador de C accede automáticamente a la dirección de memoria donde se almacena el entero.

Se puede ganar en eficacia en el acceso a esta dirección de memoria utilizando un puntero .

Introducción

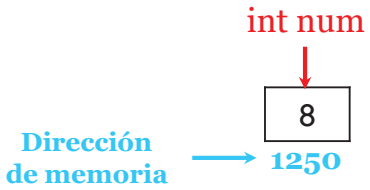
Cada vez que se declara una variable C



Representación de la memoria.

El compilador establece un área de memoria para almacenar el contenido de la variable

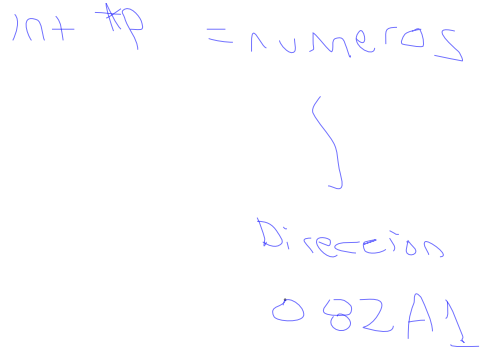
El espacio para esa variable se sitúa en una posición específica de la memoria, conocida como dirección de memoria



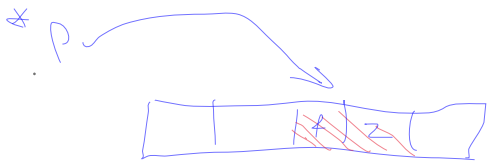
Se puede ganar en eficacia en el acceso a esta dirección de memoria utilizando un puntero

¿Qué es un puntero?

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.




```c
#include <stdio.h>

int main() {
    int numeros[5] = {10, 20, 30, 40, 50};
    int *puntero;

    puntero = numeros; // Asignar la dirección de inicio del arreglo al p

    printf("Elementos del arreglo usando punteros:\n");
    for (int i = 0; i < 5; i++) {
        printf("Elemento %d: %d\n", i+1, *puntero);
        puntero++; // Mover el puntero al siguiente elemento del arreglo
    }

    return 0;
}
```

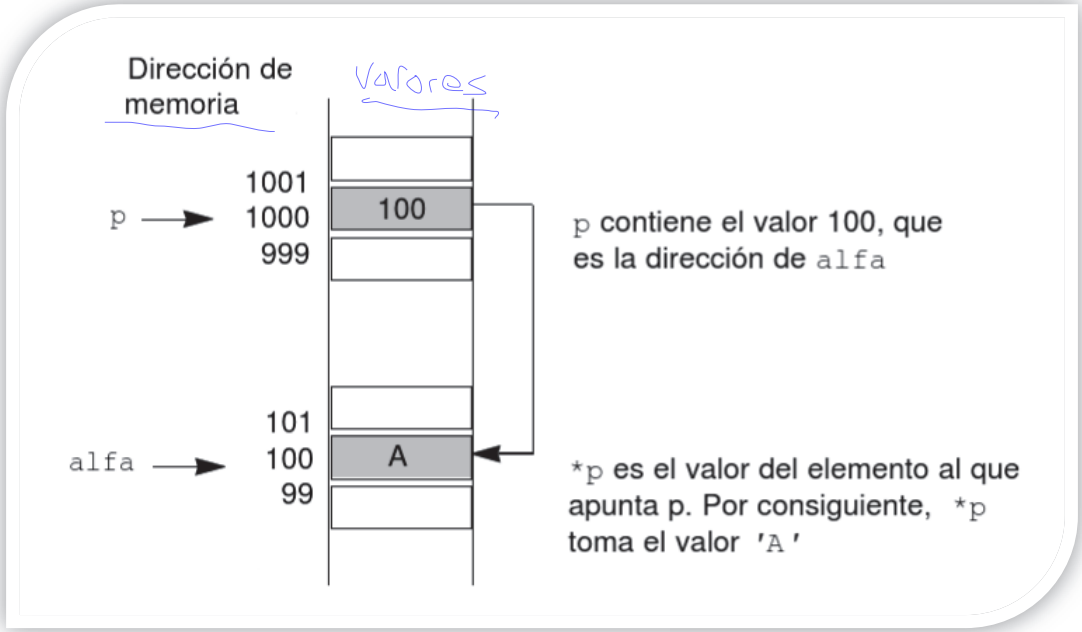


Los punteros se rigen por estas reglas básicas…

- Un puntero es una variable como cualquier otra;
- Una variable puntero contiene una dirección que apunta a otra posición en memoria;
- En esa posición se almacenan los datos a los que apunta el puntero;
- Un puntero apunta a una variable de memoria.

```c
#include <stdio.h>

int main() {
    int numeros[5] = {10, 20, 30, 40, 50};
    int *puntero;

    puntero = numeros; // Asignar la dirección de inicio del arreglo al p

    printf("Elementos del arreglo usando punteros:\n");
    for (int i = 0; i < 5; i++) {
        printf("Elemento %d: %d\n", i+1, *puntero);
        puntero++; // Mover el puntero al siguiente elemento del arreglo
    }

    return 0;
}
```

Grafica de un puntero…



Declaración de punteros

La declaración de un puntero consiste en:

tipo *nombre-vble-puntero

Donde:

tipo: cualquier tipo de c (int, char, float).

*: operador de puntero.

nombre-vble-puntero: identificador.

Declaración de punteros

Esta declaración significa que:

- La variable definida de tipo puntero apunta a objetos del tipo base especificado.
- La variable puntero es capaz de almacenar la dirección del objeto al cual apunta.
- El valor de un puntero es una dirección.
- El objeto al cual apunta un puntero se llama objeto referenciado. Los objetos referenciados son del tipo base especificado.

Los operadores de un puntero

& Operador de dirección

El &, devuelve la dirección de memoria de su operando.

p1 = &car; //p1 recibe la dirección de car.

Podemos ver que el operador & sirve para acceder a la dirección, NO al contenido del objeto referenciado.

Recuerdan:

```
scanf("%d", &numero);
```

Ahora podemos entender, que la función scanf(), tiene como parámetro la dirección de memoria de la variable en la cual queremos almacenar un valor.

p1 = &car;
p1 = dirección memoria de car.

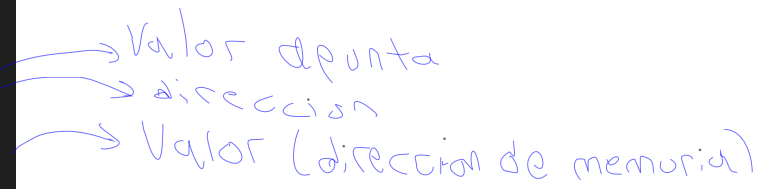
Los operadores de un puntero

- Operador de indirección

El operador * es el complemento del operador **&**, devuelve el contenido del objeto referenciado.

Importante: Los operadores & y * tienen precedencia superior a todos los operadores aritméticos.

```c
#include <stdio.h>

int main() {
    int x = 10;
    int *puntero;

    puntero = &x;

    printf("El contenido de la variable x es: %d\n", x);
    printf("El contenido al que apunta el puntero es: %d\n", *puntero);
    printf("La dirección de memoria de la variable x es: %p\n", &x);
    printf("El puntero almacena la dirección de memoria: %p\n", puntero);

    return 0;
}
```



Inicialización de punteros

Como otras variables en C, los punteros pueden ser inicializados en su definición. Por ejemplo:

```c
#include <stdio.h>

int main()
{
    int prim;
    int *punt_num = &prim;

```

Ejemplos de uso de punteros

El siguiente programa cambiará los contenidos de las variables prim, seg.

```c
#include <stdio.h>

int main()
{
    int prim = 10, seg=20, temp;
    int *punt_num;

    printf("Al inicio - prim: %d, seg:%d\n", prim, seg);

    punt_num = &prim;
    temp = *punt_num;

    *punt_num = seg;
    seg = temp;

    printf("Despues - prim: %d, seg:%d", prim, seg);

    return 0;
}
```

salida:

Al inicio prim: 10, seg: 20
Despues prim: 20, seg: 10

Operaciones con Punteros

- Asignaciones de Punteros.
- Aritméticas de Punteros.
- Comparaciones de Punteros.

Asignaciones con Punteros

Como cualquier variable, se puede usar un puntero en la parte derecha de una sentencia de asignación.

```c
#include <stdio.h>

int main()
{
    int num;
    int *punt;

    num = 101;
    punt = &num;

    printf("La dirección es: %p\n", punt);

    printf("El valor de x es: %d", *punt);

    return 0;
}
```

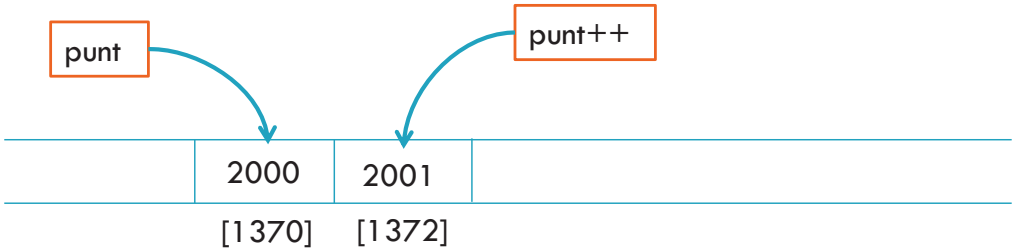
Aritméticas de Punteros (de Direcciones)

Solo se pueden realizar dos operaciones: +, -

Para entender que ocurre en la aritmética de direcciones,

lo veremos mediante un ejemplo:

```
int *punt;
```



int *p

Aritméticas de Punteros (de Direcciones)

Cada vez que incrementamos el puntero punt, apuntará tantas direcciones más como bytes tenga el tipo base.

Pensemos un entero tiene 2 bytes de longitud, cuando se incrementa un puntero entero, su valor crece en dos. Es decir, el incremento o decremento se produce en función de su tipo base.

```c
#include <stdio.h>

int main()
{
    printf("Hasta la próxima clase!!\n");

    return 0;
}
```