

Programación básica

Alma González
Octubre 2021



Apuntadores

- Cada variable de un programa tiene un espacio asignado y una dirección para ubicarlo en la memoria, a la cual podemos acceder, usando el operador &
 - Ej. `scanf("%f",&a)`.
- Los apuntadores en C (C++, y otros lenguajes de programación) son usados para acceder a la memoria y manipular la dirección asociada a las variables. A veces es más sencillo realizar tareas usando los apuntadores que usando las variables.
- Un apuntador es una variable cuyo valor asignado es la dirección en la memoria de otra variable. Debemos declarar las variables de tipo apuntadores de la siguiente forma:

```
int    *ip;    /* apuntador a un entero */
double *dp;    /* apuntador a un double */
float  *fp;    /* apuntador a un float */
char   *ch     /* apuntador a un char */
```

ip, dp, fp, ch, son solo nombres de las variables, i.e. pueden ser diferentes.

Operaciones con apuntadores

- Las operaciones mas importantes que realizaremos con apuntadores son:
 - 1) Definir las variables de tipo apuntador
 - 2) Asignar la dirección de otra variable a un apuntador
 - 3) Acceder al valor guardado en la dirección dada por la variable apuntador.

```
#include <stdio.h>
int main () {
    int var = 20;
    int *ip;
    ip = &var;

    printf("La dirección de la variable var es: %p\n", &var );
    printf("Direccion guardada en el apuntador ip: %p\n", ip );
    printf("El valor escrito en la dirección %p es : %d\n", ip, *ip );
    return(0);
}
```

Uso de apuntadores

- Es buena practica asignar el valor NULL al apuntador, cuando aún no se tiene la dirección a asignar. De esta forma no tendremos apuntadores sin asignar.

```
#include <stdio.h>
int main () {
    int var = 20;
    int *ip = NULL;

    printf("La dirección inicial almacenada en el apuntados es %p \n", ip );

    ip = &var;

    printf("La dirección de la variable var es: %p\n", &var );
    printf("Direccion guardada en el apuntador ip: %p\n",ip );
    printf("El valor escrito en la dirección %p es : %d\n", ip, *ip );

    return(0);
}
```

Uso de apunadores

- Es posible checar si un apuntador tiene dirección nula, o no:

```
if(ptr)          /*  
verdadero si ptr es no  
NULL */  
if(!ptr)         /*  
verdadero si ptr es  
NULL */
```

```
#include <stdio.h>  
int main () {  
    int var = 20;  
    int *ptr = NULL;  
  
    if(ptr){  
        printf("La dirección de la variable var es: %p\n", &var );  
        printf("Direccion guardada en el apuntador ptr: %p\n",ptr );  
        printf("El valor escrito en la dirección %p es : %d\n", ptr, *ptr );  
    } else{  
        printf("No se ha asignado dirección al apuntador\n");  
    }  
  
    ptr = &var;  
    printf("Después de hacer ptr=&var\n");  
  
    if(ptr){  
        printf("\t La dirección de la variable var es: %p\n", &var );  
        printf("\t Direccion guardada en el apuntador ptr: %p\n",ptr );  
        printf("\t El valor escrito en la dirección %p es : %d\n", ptr, *ptr );  
    } else{  
        printf("\t No se ha asignado dirección al apuntador\n");  
    }  
  
    return(0);  
}
```

Aritmetica de apuntadores

- Un apuntador contiene una dirección en la computadora, la cual tiene un valor numérico. Podemos realizar 4 operaciones aritméticas: ++, --, +, -
- Ejemplo: Supongamos que la variable **ptr** es un apuntador a una variable entera, que tiene la dirección 1000.
- La operación **ptr++** apuntara entonces a la posición 1004, porque cada vez que le sumamos 1 nos movemos una posición en la dirección de la memoria.
- Ejemplo:

```
int main () {  
    int n=3;  
    int var[n];  
    int i, *ptr;  
  
    ptr = &var[0];  
  
    var[0]=10;  
    var[1]=100;  
    var[2]=200;  
  
    for (i = 0; i < n; i++) {  
        printf("la dirección de la variable var[%d] es %p\n", i, ptr);  
        printf("Valor de la variable var[%d] = %d\n", i, *ptr);  
        /* nos movemos a la siguiente posición en la memoria */  
        ptr++;  
    }  
    return 0;  
}
```

Apuntadores para definir arreglos.

- Nos permiten hacer un manejo dinámico de la memoria asignada a un arreglo. No tengo que definir de antemano el tamaño, puedo hacerlo como parte de la ejecución del programa.
- Usaremos las funciones
 - `malloc()` #Asigna el numero de bytes indicados y devuelve un apuntador al primer byte del espacio asignado. Es un solo bloque
 - `calloc()` #Reserva bloques de memoria todos del mismo tamaño y los inicia a cero.
 - `free()` #Libera la memoria reservada
 - `realloc()` #Si la memoria previamente reservada es insuficiente o es demasiada, es posible ajustarla.

Uso de malloc

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

int main()
{
    int num, i, *ptr;

    printf("Introduce el numero de elementos con que trabajarás: ");
    scanf("%d", &num);

    ptr = (int*) malloc(num * sizeof(int)); //memoria reservada usando malloc

    if(ptr == NULL)
    {
        printf("Error! memoria no reservada.");
        exit(0);
    }

    printf("Introduce la secuencia de numeros: ");
    for(i = 0; i < num; ++i)
    {
        scanf("%d", ptr + i);
    }

    printf("\nLo numeros ingresados y almacenados en la memoria reservada son: \n");
    for(i = 0; i < num; ++i)
    {
        printf("%p \t: %d\n", (ptr + i), *(ptr + i));
    }

    free(ptr);
    return 0;
}
```


Uso calloc

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

int main()
{
    int num, i, *ptr;

    printf("Introduce el numero de elementos con que trabajarás: ");
    scanf("%d", &num);

    ptr = (int*) calloc(num, sizeof(int));
    if(ptr == NULL)
    {
        printf("Error! memoria no reservada.");
        exit(0);
    }

    printf("Introduce la secuencia de numeros: ");
    for(i = 0; i < num; ++i)
    {
        scanf("%d", ptr + i);
    }

    printf("\nLo numeros ingresados y almacenados en la memoria reservada son: \n");
    for(i = 0; i < num; ++i)
    {
        printf("%p \t: %d\n", (ptr + i), *(ptr + i));
    }

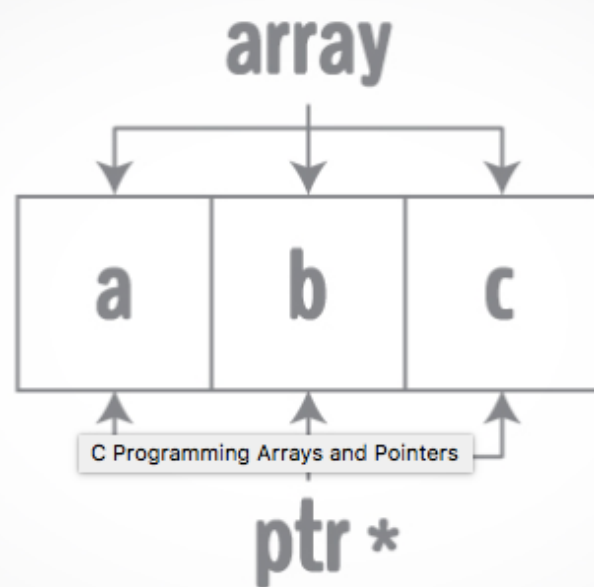
    free(ptr);
    return 0;
}
```

ARREGLO Y APUNTADORES

- Podemos acceder a los elementos de un arreglo a través del arreglo mismo, o bien a través de la dirección en la memoria, usando apuntadores.

Supongamos que se define un arreglo de la forma:

```
int arr[5];
```



`&arr[0]` es equivalente a escribir: `arr`
y se refiere a la dirección del primer elemento de `arr`.

`arr[0]` es equivalente a escribir: `*arr` (Que es el valor guardado en la dirección a la que apunta `arr`)

`&arr[1]` es equivalente a `:(arr+1)` (La dirección del segundo elemento de `arr`)

`arr[1]` es equivalente a `*(arr+1)` (El valor del elemento 1 del arreglo)

Ejercicio

ENCONTRAR LA SUMA DE 6 NÚMEROS
GUARDADOS EN UN ARREGLO
MANIPULANDO EL ARREGLO CON UN
APUNTADOR.

ENCONTRAR LA SUMA DE 6 NÚMEROS
GUARDADOS EN UN ARREGLO,
DECLARANDO Y ASIGNANDO MEMORIA A
UN APUNTADOR Y MANIPULANDO COMO
ARREGLO.