El lenguaje de programación C
- Memoria dinámica-

Punteros

- Un puntero es un número (entero largo) que se corresponde con una dirección de memoria En la declaración de un puntero, se debe especificar el tipo de dato que contiene la dirección de memoria
- Se declaran usando el carácter *
- Ejemplos: int *punteroAEntero; char *punteroAChar;

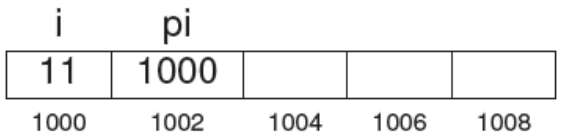
int *VectorPunterosAEntero[tVECTOR];

double **punteroAPunteroAReal;

Dirección y Contenido

`*x`	Contenido de la dirección apuntada por `x`
`&x`	Dirección de memoria de la variable `x`

```
int i=3;
int *pi;
pi=&i;     // pi=direccion de memoria de i
*pi = 11; // contenido de pi=11. Por lo tanto, i = 11
```



Malloc

La función que se encarga de efectuar ese trabajo es malloc (del inglés, reserva de memoria, *memory allocation*). Tras reservar memoria, deberemos hacer uso de free para liberarla cuando no la necesitemos más. Veamos qué parámetros maneja:

- ***Entrada***

tamaño: del tipo size_t, es decir, número de bytes que queremos reservar. Será número de elementos del vector multiplicado por el tamaño del tipo de dato (lo obtenemos con sizeof).

- ***Salida***

Dirección de memoria inicial del vector que se almacenará en un puntero. Es del tipo *void** por lo que deberemos hacerle una conversión explícita o *casting* al tipo que queramos. Será NULL si ha habido problemas ejecutando *malloc*.

Ejemplo

```
#include &lt;stdlib.h&gt;
int main ()
{
   int numeroElementos;
   int* vector;
   printf ("¿Cuántos elementos quieres? ");
   scanf ("%d", &numeroElementos);
   vector = (int*)malloc(numeroElementos*sizeof(int));
   if (vector==NULL)
     {
        perror("Problemas reservando memoria");
         exit (1);
     }
    // trabajamos con el vector
    // lo liberamos
     free (vector);
     return 0;
}
```

Ejemplo

```
int cant;
scanf("%d",&cant);
int* arr;
arr = (int*) malloc(cant*sizeof(int));
int i;
for(i=0; i<cant; i++){
        scanf("%d",&arr[i]);
}
```

Chequeo de asignación

```
int* arr;
arr = (int*) malloc(cant*sizeof(int));
if(arr==NULL){
   printf(“ERROR DE MEMORIA!”);
   exit(EXIT_FAILURE);
}
```

FREE

free(arr)

Libera la memoria asignada al puntero arr.

Matrices

```
int filas, cols;
scanf("%d %d",&filas, &cols);
int** arr;
arr = (int**) malloc(filas*sizeof(int*));
//Chequeo si arr==NULL
int i;
for(i=0; i<filas; i++){
      arr[i] = (int*)malloc(cols*sizeof(int));
      //Chequeo si arr[i]==NULL
}
```

Libero la memoria

```
for(i=0; i<filas; i++){
        free(arr[i]);
}
free(arr);
```

Calloc

Igual que Malloc pero inicializa en 0.

void* calloc(cantElems, bytesElem)

Más costoso que Malloc.

Realloc

void* realloc(void* ptr, cantBytes)

Reasigna cantBytes de memoria y copia lo apuntado por ptr a la nueva dirección de memoria.

Ejemplo

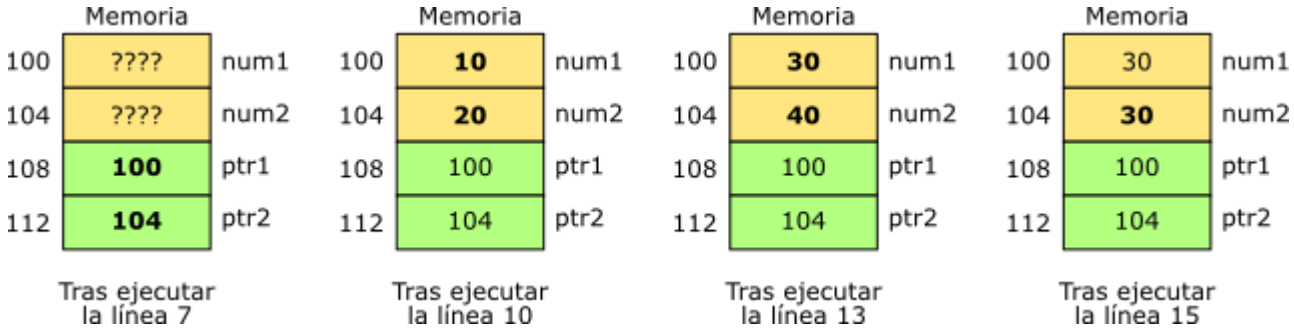
```
//arr = [4, 25, 35]

arr= (int*) realloc(arr, 4*sizeof(int));

arr[3] = 72

//arr = [4, 25, 35, 72]
```

Resumen

- **malloc** para pedir memoria.
- **calloc** para pedir memoria inicializada en 0.
- **realloc** para reasignar memoria (potencialmente pidiendo más).
- **free** para liberar memoria después de su uso.

Ejemplo

```
#include <stdio.h>
int main(int argc, char** argv)
{
    int num1, num2;
    int *ptr1, *ptr2;

      ptr1 = &num1;
(7)   ptr2 = &num2;

      num1 = 10;
(10) num2 = 20;

      *ptr1 = 30;
(13) *ptr2 = 40;

(15) *ptr2 = *ptr1;

    return 0;
}
```



Pasaje por Valor

- En el pasaje por valor, cuando hacemos el llamado a la función y especificamos los argumentos, estos se evalúan y se le pasa a la función **una copia** de los mismos. A esto es lo que llamamos **paso por valor** ya que la función trabajará con los valores de esas copias. Si pasáramos variables como argumento, éstas no se modificarían por este sistema de copias.

Funciones

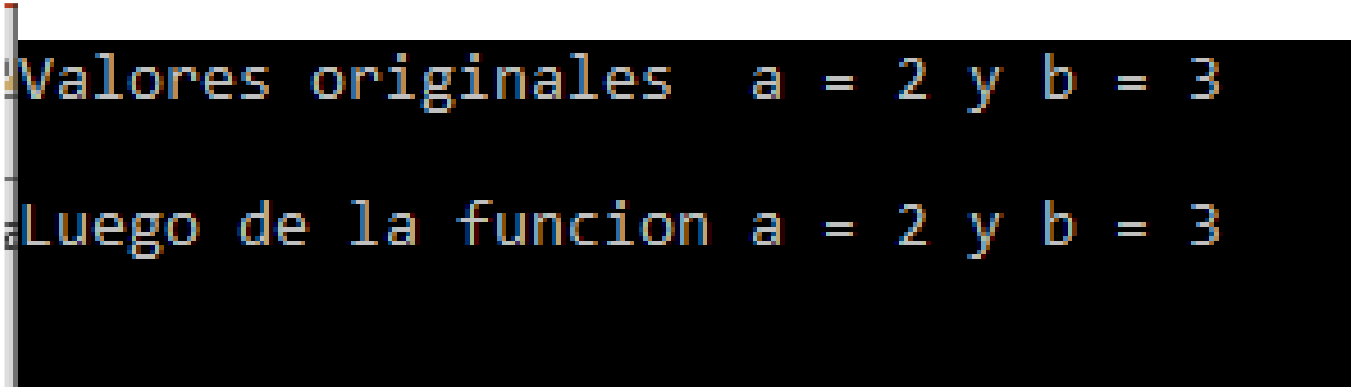
- Pasaje de parámetros por valor.

```
void intercambiar(int i, int j)
    {
            int z;
            z = i;
            i = j;
            j = z;
    }
```

Pasaje por valor

```
void intercambiar(int i , int j ); //Protipo de la función para
intercambiar los valores


int main(void)
{
   int a = 2,b = 3;
   cout<<"Valores originales  a = "<<a<<" y b = "<<b<<endl<<endl;
   intercambiar(a,b); //Llamado a la función intercambiar
   cout<<"Luego de la funcion a = "<<a<<" y b = "<<b<<endl<<endl;

   return 0;
}
```

Pasaje por valor



Pasaje por referencia

- En el pasaje por referencia se pasa a la función las direcciones de memoria de las variables en cuestión en lugar de su valor. A diferencia del paso por valor, aquí no se realizan copias de las variables sino que se trabaja sobre la dirección de memoria que pasamos, lo que nos permite modificar el valor de la variable en cuestión.

Pasaje por referencia

```
void intercambiar(int &i , int &j );

void intercambiar(int &i, int &j)
        {
            int z;
            z = i;
            i = j;
            j = z;
        }
```

Pasaje por Referencia

```
Valores originales  a = 2 y b = 3

Luego de la funcion a = 3 y b = 2
```

Ejemplo

```
int sumarXvalor(int a,int b);
void sumarXreferencia(int a,int b,int &resultado);

int main(){
int a=5;       int b=2;
int resultado=0;printf("Valores:n%in%in",a,b);printf("Paso de Parametros por Valorn");
resultado = sumarXvalor(a,b);

printf("Resultado: %in",resultado);
printf("Paso de Parametros por Referencian Pasamos el valor de posición en memoria de la
variable resultado: %pn",&resultado);

sumarXreferencia(a,b,resultado);
printf("Resultado: %in",resultado);
return 0;}

int sumarXvalor(int a, int b) {          return a+b;      }

void sumarXreferencia(int a,int b,int &resultado) {              resultado = a + b;     }
```

Pasaje de Arrays

```
void intercambia(int []); //declaración de la función.
                          //No es necesario poner el nombre del array
int main()
{
   int A[2]={3,5};
   intercambia(A); //en la llamada sólo se escribe el nombre
                   //del array, sin corchetes
   cout << A[0] << " " << A[1] << endl; //muestra 5 3
   system("pause");
}
void intercambia(int a[]) //definición de la función
{
   int c;
   c=a[0];
   a[0]=a[1];
   a[1]=c;
}
```

Pasaje de Struct

```
struct fecha
        {
                int day;
                int month;
                int year;
        };
        struct fecha Date;

        LeerFecha(Date);
```

Pasaje por referencia

Esto esta mal

```
void LeerFecha(struct fecha *Date)
{
        Date.day=1;
        Date.month=2;
        Date.year=3;
}
```

Pasaje por Referencia

Forma correcta

```
void LeerFecha(struct fecha &Date)
{
        Date->day=1;
        Date->month=2;
        Date->year=3;
}
```

Pasaje por Referencia

Otra alternativa

```
void LeerFecha(struct fecha &Date)
{
        (*Date).day=1;
        (*Date).month=2;
        (*Date).year=3;
}
```