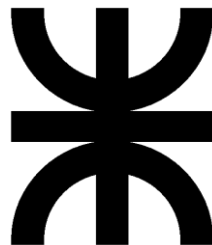


**UNIVERSIDAD TECNOLÓGICA NACIONAL**  
**FACULTAD REGIONAL RESISTENCIA**



**EXTENSIÓN ÁULICA GOYA**

**TRABAJO PRACTICO**  
**“Ejercicios de Repaso sobre**  
**Algoritmos con Arrays con vistas al**  
**2do. Parcial”**

***TECNICATURA UNIVERSITARIA EN PROGRAMACIÓN***

***DOCENTE:***

***TSP WIRZ, JORGE.***

***-Año 2023 -***

## 1- Ejercicios sobre Algoritmos con Arrays.

Antes de codificar algunos Algoritmos para administrar (en particular ordenar) Arrays, veamos algunos "refinamientos" útiles para las Funciones en C.

Las funciones en C pueden ser llamadas a las **por referencia y por valor**. Cuando las llamamos **por referencia** podemos hacer cosas que no podemos con una simple llamada **por valor**.

### Llamada por valor.

Ya conocimos las funciones en las que llamamos por valor. Cabe mencionar que esto sólo aplica a las funciones que reciben argumentos. En esas funciones, una copia de la variable es creada dentro de la llamada de la función, y no afecta a la original. Como en este ejemplo de una función que incrementa un entero:

```
int incrementar (int numero) {  
    //Incrementar en 1  
    numero = numero + 1;  
}
```

En esta función, no se está devolviendo nada, solo se incrementa la var recibida. Ahora veamos cómo funciona, llamándola desde otra región de código.

```
#include<stdio.h>  
  
// Es una buena práctica definir el prototipo de las funciones aquí arriba  
// sólo el prototipo, no la implementación  
int incrementar(int numero);  
  
int main(int argc, char const *argv[])  
{  
    int numero = 10;  
    printf("Antes de llamar a la funcion, numero es %d\n", numero);  
    incrementar(numero);  
    printf("Despues de llamar a la funcion, numero es %d", numero);  
}  
  
// Ahora sí definimos la función con todo y cuerpo  
int incrementar(int numero){  
    //Incrementar en 1  
    numero = numero + 1;  
}
```

La consola de salida debería quedar así:

*Antes de llamar a la funcion, numero es 10*

*Despues de llamar a la funcion, numero es 10*

¿Por qué no cambió su valor? Porque se crea una nueva variable dentro de la función, con el mismo ID inclusive. Y aunque incrementemos a la misma, la variable "original" dentro del main () sigue intacta. (O sea: hay "una" var numero en la función main () y hay "otra" var numero en la función incrementar ()) Esto es a lo que sucede cuando pasamos variables por valor.

### Llamada por referencia.

Una llamada por referencia ocurre cuando no pasamos el nombre de la variable, sino su dirección de RAM. En este caso **no se crea** una nueva variable.

Modifiquemos la función para que ahora reciba el puntero o apuntador o "pointer" a una variable, no una variable en sí. Lo único que hay que hacer es agregar el \* previo al ID en cuestión. Podría quedar así:

```
int incrementar(int *numero){  
    //Incrementar en 1  
    (*numero) = (*numero) + 1;  
}
```

El código anterior ahora recibe direcciones de variables. Y al incrementar, incrementa el valor que haya en la dirección que dijimos. Nótese el operador & que se encarga de obtener la dirección de memoria de una variable:

```
#include<stdio.h>  
  
// Es una buena práctica definir el prototipo de las funciones aquí arriba  
// sólo el prototipo, no la implementación  
int incrementar(int *numero);  
  
int main(int argc, char const *argv[])  
{  
    int numero = 10;  
    printf("Antes de llamar a la funcion, numero es %d\n", numero);  
  
    // Con el operador & obtenemos la dirección de numero  
    incrementar(&numero);  
    printf("Despues de llamar a la funcion, numero es %d", numero);  
}  
  
// Ahora sí definimos la función con todo y cuerpo  
//Notar el * antes de numero
```

```
int incrementar(int *numero){  
    //Incrementar en 1  
    (*numero) = (*numero) + 1;  
}
```

Ahora la salida por consola debería ser:

*Antes de llamar a la funcion, numero es 10*  
*Despues de llamar a la funcion, numero es 11*

### Usos de las llamadas por referencia.

Las llamadas por referencia son más rápidas, ya que no se crean nuevos valores en la memoria. Aparte del rendimiento, las funciones por referencia son usadas cuando queremos devolver dos variables; eso no es posible con una función normal, pero con una función por referencia sí.

Veamos otro ejemplo de una función que devuelve dos números enteros. No hace un return, en cambio recibe a ambos números y a cada uno de ellos le asigna un valor.

```
#include<stdio.h>  
  
int devolverEnteros(int *numero1, int *numero2);  
  
int main(int argc, char const *argv[])  
{  
    int numero1 = 0, numero2 = 0;  
    printf("Antes de llamar a la funcion, numero1 es %d y numero2 es %d\n",  
        numero1, numero2);  
  
    // Con el operador & obtenemos la dirección de numero  
    devolverEnteros(&numero1, &numero2);  
    printf("Despues de llamar a la funcion, numero1 es %d y numero2 es %d\n",  
        numero1, numero2);  
}  
  
int devolverEnteros(int *numero1, int *numero2){  
    (*numero1) = 10;  
    (*numero2) = 20;  
}
```

### Otro uso: para intercambiar variables.

Cuando queremos intercambiar una variable también hacemos uso de estas funciones.

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

```
int intercambiarEnteros(int *numero1, int *numero2);
```

```
int main(int argc, char const *argv[])
```

```
{
```

```
int numero1 = 50, numero2 = 85;
```

```
printf("Antes de llamar a la funcion, numero1 es %d y numero2 es %d\n", numero1,  
numero2);
```

```
intercambiarEnteros(&numero1, &numero2);
```

```
printf("Despues de llamar a la funcion, numero1 es %d y numero2 es %d\n",  
numero1, numero2);
```

```
}
```

```
int intercambiarEnteros(int *numero1, int *numero2){
```

```
    //ejemplo evidente de "triangulación"
```

```
    int temporal = (*numero1);
```

```
    (*numero1) = (*numero2);
```

```
    (*numero2) = temporal;
```

```
}
```

Ahora sí, vistos estos conceptos con utilidad de herramientas, veamos algunos Algoritmos de Ordenamiento.

### **Algoritmo de la Burbuja.**

El método de la burbuja en C es un algoritmo para ordenar estructuras de datos y es fácilmente aplicable a Arreglos; no es el más rápido, pero es uno que sirve para introducir los conceptos de ordenamiento de arreglos en C.

Ordenar un arreglo en C usando el método de la burbuja es sencillo; simplemente se recorre el arreglo en un ciclo for, y dentro de ese ciclo, se hace otro ciclo; es decir, tenemos dos ciclos.

En el segundo ciclo (que va desde 0 hasta la longitud del arreglo menos el paso del primer ciclo) comparamos el elemento actual con el siguiente, y si el actual es mayor, intercambiamos los valores. Esto se repite y al final el arreglo estará ordenado.

El algoritmo es sencillo; hay que recorrer todo el arreglo y si encontramos que el elemento actual (arreglo[x]) es menor al elemento siguiente (arreglo[x+1]) entonces los intercambiamos.

Es importante hacer este recorrido hasta la longitud menos 1 para que cuando lleguemos al penúltimo elemento y hagamos un x+1 el índice no esté fuera de los límites del arreglo.

Con esto habremos ordenado solo una parte del arreglo; hay que hacer todo este recorrido de nuevo, específicamente N veces en donde N es la longitud del arreglo; al terminar, el arreglo estará ordenado.

### Código ejemplo del ordenamiento de burbuja en C.

Como herramienta para intercambiar los elementos vamos a usar una función ya expuesta que utiliza pasaje por referencias y "triangulación", no es estrictamente necesaria, pero ayuda a ahorrar líneas de código.

```
void intercambiar(int *a, int *b) {
    int temporal = *a;
    *a = *b;
    *b = temporal;
}
```

El código del Algoritmo como función podría quedar:

```
void burbuja(int arreglo[], int longitud) {
    for (int x = 0; x < longitud; x++) {
        // -1 es porque no queremos llegar al final ya que hacemos
        // un indiceActual + 1 y si fuéramos hasta el final, intentaríamos acceder a un
        // valor fuera de los límites del arreglo
        for (int indiceActual = 0; indiceActual < longitud - 1; indiceActual++) {
            int indiceSiguienteElemento = indiceActual + 1;
            // Si el actual es mayor que el que le sigue a la derecha...
            if (arreglo[indiceActual] > arreglo[indiceSiguienteElemento]) {
                // ...intercambiarlos, es decir, mover el actual a la derecha y el de la
                // derecha al actual
                intercambiar(&arreglo[indiceActual], &arreglo[indiceSiguienteElemento]);
            }
        }
    }
}
```

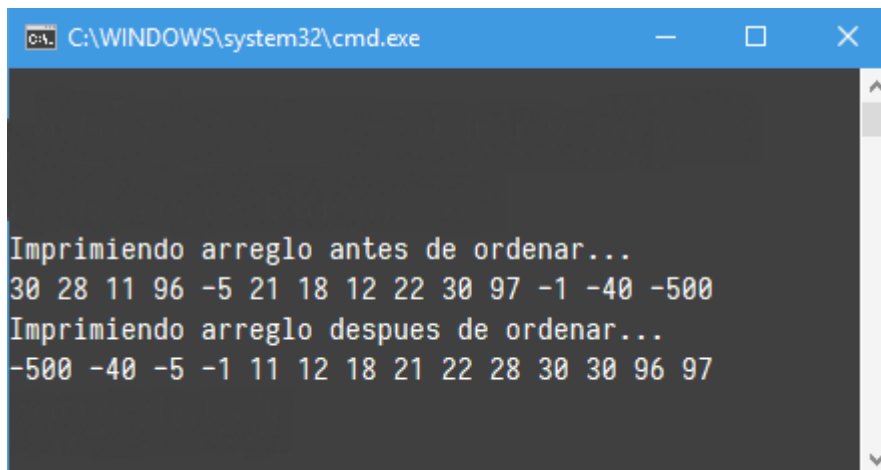
La función recibe el arreglo y la longitud del mismo para funcionar. Eso es lo único que se necesita, a partir de aquí podemos invocarla y el arreglo estará ordenado después de la invocación.

Ahora, un programa ejemplo aplicando ambas funciones sobre un arreglo:

```
int main(void) {
    int arreglo[] = {30, 28, 11, 96, -5, 21, 18, 12, 22, 30, 97, -1, -40, -500};
    /* En este caso, si conviene alcular la longitud*/
    int longitud = sizeof arreglo / sizeof arreglo[0];
    /* Imprimirlo antes de ordenarlo */
    printf("Imprimiendo arreglo antes de ordenar...\n");
```

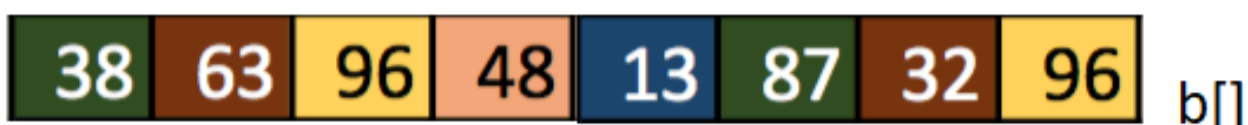
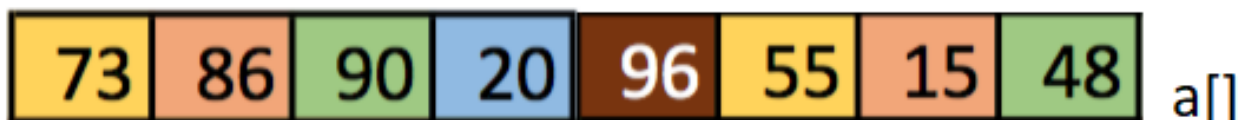
```
for (int x = 0; x < longitud; x++) {  
    printf("%d ", arreglo[x]);  
}  
// Un salto de línea  
printf("\n");  
  
/* Invocar al método de la burbuja indicando todo el arreglo, desde 0 hasta el  
índice final */  
burbuja(arreglo, longitud);  
/* Imprimirlo después de ordenarlo */  
printf("Imprimiendo arreglo despues de ordenar...\n");  
for (int x = 0; x < longitud; x++)  
    printf("%d ", arreglo[x]);  
return 0;  
}
```

La salida por consola debería quedar así:



### Ejercicios para desarrollar.

- 1 – Dados los vectores en la imagen, se pide:





- a) Dados los conceptos vistos, ordene el vector `a[]` mediante el Algoritmo de Burbuja.
- b) Investigue y, aplicando los conceptos vistos ordene el vector `b[]` mediante el Algoritmo de Selección.
- c) Cree un tercer vector `c[]` que contenga los valores de los 2 vectores anteriores concatenados (o sea, 16 subíndices) en el orden ordenado.
- d) Modifique los códigos de a) y b) de manera que se ingresen por teclado : 1) tamaño del vector y dimensionarlos en base a los recibidos. 2) Valores para cada subíndice.

En las resoluciones, deben aplicarse: uso de funciones, pasaje de valores por referencias, "triangulación de valores", comentarios (remarks).