

PROGRAMACIÓN

Unidad 6: Mecanismos de Pasaje de parámetros

Lic. Mariela A. Velázquez

Temas a tratar

**P
A
R
T
E**

1

Definición de punteros

Conceptos básicos

Declaración de punteros

Operaciones de punteros

Punteros y Arreglos

Punteros y Funciones

Pasaje de Parámetros

**P
A
R
T
E**

2

Recordemos un poco...

Puntero



```
graph LR; P[Puntero] --> D1[Es una variable de contiene la dirección de memoria de otra variable]; P --> D2[Declaración de Punteros: tipo *nombre-vble-puntero]; P --> D3[* Operador de indirección]; P --> D4[& Operador de dirección]; P --> D5[Inicialización: punt_num = &numero;];
```

Es una variable de contiene la dirección de memoria de otra variable

Declaración de Punteros: **tipo *nombre-vble-puntero**

***** Operador de indirección

& Operador de dirección

Inicialización: `punt_num = №`

Pasaje de Parámetros

El lenguaje C transfiere variables entre funciones usando **dos mecanismos diferentes**.

Pero... de que dependerá el mecanismo que utilice?

- ✓ El método a emplear depende de si se quiera modificar o no los valores de las variables transferidas.
- ✓ De la cantidad de valores “devueltos” por la función.



Revisemos algunos conceptos

```
#include<stdio.h>

int esPerfecto(int num);

int main()
{
    int indice, resultado;

    printf("\n*****NUMEROS PERFECTOS***** \n");
    for (indice=1;indice<=100;indice++)
    {
        resultado = esPerfecto(indice);
        if (resultado == 1)
            printf(" %d Es un numero perfecto \n",indice);
    }
    return 0;
}

int esPerfecto(int num)
{
    int i, suma=0, resto;

    for (i=1;i<num;i++)
    {
        resto=num%i;
        if (resto==0)
            suma=suma+i;
    }
    if(suma == num)
        return 1;
    else
        return 0;
}
```

Parámetros Actuales: son las expresiones pasadas como argumentos en la llamada a una función.

Función Emisora: Función que emite valores para la transferencia de información.

Parámetros Formales: Los parámetros formales sólo se conocen dentro de la función.

Función Receptora: Función que recepciona valores transferidos.

Pasaje de Parámetros

- Pasaje por valor
- Pasaje por Referencia

Pasaje por Valor

Cuando pasamos parámetros por valor:

- ✓ La función que invoca recibe los valores de sus argumentos en **variables temporales y no en las originales.**
- ✓ La función que se invoca no puede alterar directamente una variable de la función que hace la llamada, solo puede modificar su copia temporal.

Pasaje por Valor

Ejemplo: Función para calcular el peso de una persona en la luna.

```
FUNCION pesoenlaluna;  
ARGUMENTO: Peso. real >0;  
SALIDA: real ≥ 0;  
pesoenlaluna ← peso/6;
```

Función emisora: transfiere el peso en la tierra.

Función receptora: recibe el valor del peso en la tierra, calcula y devuelve el valor en la luna, sin modificar el valor del peso en la tierra.

Pasaje por Valor

- En el momento de la invocación, la función receptora toma, en forma temporal, un espacio de almacenamiento de características idénticas a las del Parámetro actual.
- Si se hacen cambios dentro de la función en el valor de la variable transferida, los cambios se reflejan en esa localidad de memoria temporal.
- Cuando la función termina su ejecución, el espacio de almacenamiento temporal que se usó para “la copia” queda liberado, perdiéndose los valores que había allí almacenado.

Pasaje por Valor

```
#include <stdio.h>

float pesoEnLuna(float PesoTierra);

int main()
{
    float PesoTierra, PesoLuna;
    PesoTierra = 95.8;
    PesoLuna = pesoEnLuna(PesoTierra);

    printf("El peso en la luna de Homero es: %.2f \n", PesoLuna);
    printf("El peso en la tierra de Homero es: %.2f", PesoTierra);

    return 0;
}

float pesoEnLuna(float PesoTierra){
    float aux;
    aux= PesoTierra/6;
    return (aux);
}
```

Calcular el peso de Homero Simpson en la luna.



Pasaje por Valor

- **Una desventaja fuerte de este mecanismo:**
Las funciones que lo usan pueden devolver a lo sumo un resultado asociado a la proposición return.
- **Ventaja:** Preserva los valores de las variables transferidas.

Pasaje por Referencia

Ejemplo: cambios de precios de varios productos comerciales.

El precio de un producto **precioProd** se calcula como la suma de los precios de sus insumos **Insumo1** y **Insumo2** más una constante. Los precios de **Insumo1** y **Insumo2** se modifican periódicamente, por lo cual el precio de nuestro producto **precioProd** también se modifica. Calcular los incrementos de **Insumo1**, **Insumo2** y **precioProd** según los porcentajes de cambio indicados.

Porcentajes de Cambios: **Insumo1** se incrementa en un 3% y **Insumo2** en un 5%.

$$\text{Insumo1} \leftarrow \text{Insumo1} * 1.03$$

$$\text{Insumo2} \leftarrow \text{Insumo2} * 1.05$$

$$\text{precioProd} \leftarrow \text{Insumo1} + \text{Insumo2} + \text{cte}$$

Pasaje por Referencia

- **Función emisora:** transfiere los precios de los productos (generalmente main).
- **Función receptora:** recibe los precios de los productos, los modifica y los devuelve modificados.

Pasaje por Referencia

Ejemplo: Cambios de precios de productos comerciales.

El precio de un producto **precioProd**, se calcula como la suma de los precios de los insumos **Insumo1**, **Insumo2**. Los precios **Insumo1**, **Insumo2** se modifican periódicamente, por lo cual el precio de nuestro producto **precioProd** también se modifica. Calcular los incrementos de **Insumo1**, **Insumo2** y **precioProd** según los porcentajes de cambios indicados.

Porcentajes de cambios:

- **Insumo1** se incrementa un 3%
- **Insumo2** se incrementa un 5%

```
float modificarPrecio (float *insumo1, float *insumo2)
{
    float precioProd;

    *insumo1 = (*insumo1) * 1.03;
    *insumo2 = (*insumo2) * 1.05;
    precioProd = (*insumo1)+ (*insumo2);

    return(precioProd);
}
```

```

#include <stdio.h>

float modificarPrecio(float *pinsumo1, float *pinsumo2);

int main()
{
    float insumo1, insumo2, precioProd;

    printf("Ingrese el precio del insumo 1:");
    scanf("%f", &insumo1);

    printf("Ingrese el precio del insumo 2:");
    scanf("%f", &insumo2);

    precioProd = modificarPrecio(&insumo1, &insumo2);

    printf("El precio del producto es: %f \n", precioProd);
    printf("El insumo1 modificado es: %f \n", insumo1);
    printf("El insumo2 modificado es: %f \n", insumo2);

    return 0;
}

float modificarPrecio(float *pinsumo1, float *pinsumo2)
{
    float precioProd;

    *pinsumo1 = (*pinsumo1) * 1.03;
    *pinsumo2 = (*pinsumo2) * 1.05;
    precioProd = (*pinsumo1) + (*pinsumo2);

    return(precioProd);
}

```

Pasaje por Referencia

- La función receptora tendrá como Parámetros formales variables de tipo puntero.
- En el momento de la invocación, la función receptora recibe como parámetros actuales las direcciones de memoria de variables de la función emisora.
- Los cambios realizados adquieren el carácter de permanentes, es decir, mientras dure la ejecución del programa. Cuando la función termina su ejecución, todos los cambios realizados en esa locación de memoria quedan allí, por esto su carácter de permanentes.

Pasaje por Referencia

- Ventajas:
 - Bajo costo en términos de espacio de almacenamiento, No se realizan duplicaciones o copias de variables.
 - La función puede “regresar” más de un valor.
- Desventaja:
 - Usar con precaución, las variables quedan muy vulnerables.

Pasaje por Referencia

Cada vez que se transfiere una variable por dirección, si la función receptora modifica la variable, ésta es modificada también en la función invocante.

Pasaje de Parámetros

- ¿Cómo elegir que mecanismo usar?
 - No modificar los valores de las variables usadas como parámetros actuales: **POR VALOR**
 - Modificar valores de las v. usadas como parámetro actuales y obtener mas de un valor de retorno: **POR REFERENCIA**

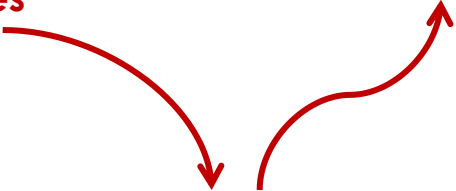
Pasaje por Referencia - Arreglos

Cuando el nombre de un arreglo se emplea como argumento, el valor que se pasa a la función es la dirección de memoria de la 1^{ra} componente del arreglo, se trabaja sobre la dirección original.

Pasaje por Referencia - Arreglos

**Ingresa con sus
valores iniciales**

Sale modificado



```
Void Inicializar(int arre[] , int n)
{
    int i;
    For(i=0; i<n; i++)
    {
        arre[i]=0;
    }
}
```

Cuando se llama la función de nombre “inicializar” se le envían 2 parámetros para que pueda proceder. Un entero que corresponde al tamaño del arreglo, (pasaje por valor), y también se envía un arreglo.

Cuando enviamos el nombre del arreglo, lo que enviamos en realidad es la dirección de memoria del primer elemento del arreglo. Es por este motivo que decimos que al pasar un arreglo a una función utilizamos pasaje por referencia.

Asignación de memoria dinámica

Una característica esencial del lenguaje C es la capacidad de requerir bloques de memoria variable durante la ejecución del programa.

Hasta ahora hemos visto que se reserva espacio para variables en el momento que las definimos. Por ejemplo:

```
int x; // reservamos espacio para almacenar un entero
float z; //reservamos espacio para almacenar un valor real
char car; //reservamos espacio para almacenar un caracter
int arreglo[10]; //reservamos espacio para almacenar 10 enteros
int *puntero; //reservamos espacio para almacenar un puntero
```

Asignación de memoria dinámica

El lenguaje C nos permite en tiempo de ejecución solicitar espacio mediante la función **malloc** (**m**emory **al**locate = Asignar memoria) y luego de usarla en forma obligada debemos devolverla llamando a la función **free**.

Ambas funciones se encuentra dentro de la librería: **<stdlib.h>**

```

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

int main()
{
    int *puntero;
    int tam;
    int indice;

    printf("Cuantos elementos tendra el arreglo:");
    scanf("%d",&tam);

    puntero=malloc(tam*sizeof(int));

    for(indice=0;indice<tam;indice++)
    {
        printf("Ingrese elemento:");
        scanf("%d",&puntero[indice]);
    }

    printf("Contenido del vector dinamico:");
    for(indice=0;indice<tam;indice++)
    {
        printf("%d ",*puntero);
        puntero++;
    }

    free(puntero);

    return 0;
}

```

Asignación de memoria dinámica

Trabajemos con el siguiente ejemplo:

Crear un arreglo en forma dinámica, cargar e imprimir sus datos.

Asignación de memoria dinámica

Expliquemos el paso a paso:

- Para trabajar con la memoria dinámica en el lenguaje C es obligatorio trabajar con punteros. Lo primero que hacemos es definir un puntero a entero:

```
int *puntero;
```

- Solicitamos al usuario de nuestro programa que ingrese un entero que representa la cantidad de elementos que tendrá el arreglo:

```
printf("Cuantos elementos tendra el arreglo:");  
scanf("%d",&tam);
```

- No esta permitido indicar en el índice de un arreglo con una variable: ~~int arre[tam];~~
- Pero este problema lo podemos resolver solicitando espacio mediante la función malloc, debemos indicar el número de bytes a reservar:

```
puntero=malloc(tam*sizeof(int));
```

El operador sizeof cuando le pasamos como parámetro un tipo de dato lo que nos devuelve es la cantidad de bytes que se requieren para almacenar dicho tipo de dato.

Asignación de memoria dinámica

- Una vez que hemos reservado espacio, recorremos y cargamos el arreglo como siempre:

```
for(indice=0;indice<tam;indice++)
{
    printf("Ingrese elemento:");
    scanf("%d",&puntero[indice]);
}

printf("Contenido del vector dinamico:");
for(indice=0;indice<tam;indice++)
{
    printf("%d ",*puntero);
    puntero++;
}
```

- Finalmente, cuando no necesitamos más dicho arreglo procedemos a liberar el espacio llamando a la función **free** y pasando el nombre del puntero.

```
free(puntero);
```

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

```
int main()
```

```
{
```

```
    printf("Hasta la próxima clase!!\n");
```

```
    return 0;
```

```
}
```