Campus Virtual
UAIOnline

Fac. de Tecnología Informática
Analista Programador

PROGRAMACIÓN ESTRUCTURADA

UNIDAD 5
FUNCIONES

CLASE 10 Y 11:
SUBRUTINAS

FUNCIONES POR PASAJE DE VALOR

FUNCIONES POR PASAJE DE REFERENCIA

UAI
Universidad Abierta Interamericana
El futuro sos vos.

PRESENTACIÓN

En esta unidad estudiaremos una de las piedras angulares de la programación en C, las funciones.

Las funciones son microprogramas dentro de un programa. Están conformadas por un conjunto de sentencias o rutinas que pueden ser llamadas desde distintos lugares del programa para resolver problemas puntuales.

Usted ya ha estudiado que existen funciones previamente definidas en las librerías de C. En esta unidad, usted aprenderá a generar sus propias funciones específicas para la resolución del problema que se le presente.

Recuerde que le propusimos la lectura simultánea de los contenidos para la resolución de las Actividades para la Facilitación de los Aprendizajes que le ofrecimos con el estudio de las estructuras de control. Si no lo hizo en ese momento le solicitamos que, luego del estudio de estas clases vuelva a aquellos ejercicios e intente resolverlos para lograr la integración de los contenidos abordados.

La temática de las funciones se inscribe dentro de una discusión entre los estudiosos de temas de tecnología informática. Mientras que algunos dicen que en el lenguaje C todo es una función, otros establecen la diferencia entre función y procedimiento.

Sostienen que la función hace referencia a una rutina que luego de procesar datos devuelve un resultado, en cambio, los procedimientos pueden ser llamados pero no devuelven algo sino que lo hacen internamente durante su ejecución. Este es un tema no resuelto pero que destaca la importancia de las funciones en el diseño eficiente de programas.

Así mismo las funciones necesitan de parámetros que reciben, mientras que los procedimientos utilizan variables generalmente definidas en forma global

Para que dimensione la utilidad de las funciones le presentamos un ejemplo simple.

Suponga dos ejercicios:

1. Calcular el sueldo de un empleado donde sueldo = cantidad de horas x valor de la hora.

2. Calcular la superficie del rectángulo.

En ambos casos se deberá pasar a la función dos datos:

1. Valor de la hora, cantidad de horas trabajadas.

2. Valor del lado 1, valor del lado 2.

¿Qué hará la función en ambos casos? ¡Multiplicarlos! Es la misma operación.

1. Sueldo = cálculo (cantidad de horas, valor de la hora)

2. Superficie = cálculo (lado1, lado2)

La función calcular escrita ligeramente podría ser rta = dato1 x dato2

Es decir, escribiendo una misma rutina puedo solucionar distintos problemas que en el fondo son iguales. Ahorrar espacio, reducir repeticiones, facilitar la programación son algunas de las ventajas que hacen de las funciones un contenido clave para el programador.

Las funciones que usted estudiará se dividen en dos categorías: **funciones por pasaje de valor**, aquellas que realizan una operación y devuelven un resultado pero no modifican los datos originales cuando devuelven el control al programa principal y las **funciones por referencia** donde sí pueden volver modificados al programa principal los datos ingresados a la función como parámetros.

Se pueden definir dos tipos de variables: las globales y las locales. Las **variables globales** se definen antes de la función *main* y son visibles para todas las funciones de un programa en todo momento. Mientras tanto las **variables locales**, definidas dentro de la función *main* o de cualquier función generada por nosotros, sólo existen en el momento que esa función es invocada, liberando el resto del tiempo los recursos de memoria que ocupan.

En esta unidad estudiará cómo **generar funciones propias** y la manera de definir y manejar las variables locales dentro de ellas.

> Como usted ya sabe, el libro es el insumo fundamental y la fuente bibliográfica irremplazable para el estudio de los contenidos de esta asignatura. Por ello, en este Orientador del Aprendizaje le ofreceremos una selección de ideas principales y algunos ejemplos que usted deberá ampliar con la lectura del capítulo correspondiente a Funciones.

Esperamos que través del estudio de esta unidad, adquiera capacidad para:

- Comprender y valorar la utilidad del uso de las funciones.
- Identificar funciones y utilizarlas para el diseño de programas.
- Generar funciones de cualquier tipo que optimicen un programa mediante la reutilización de código.

Funciones

¿Qué es una función?, podemos decir que es una parte de mi programa que realiza una determinada acción sea lo sea que recibe, por ejemplo, A=B*C, sin importar que le enviemos los lados de un cuadrado para calcular la superficie o el valor de la hora y cantidad de hs para calcular el sueldo, solo sabe que debe realizar una multiplicación

Las funciones en el lenguaje C y varios más que heredan sus conceptos se pueden dividir en 2 de acuerdo a como se pasan los parámetros de las mismas

Como está compuesta una función

Que devuelve NOMBRE (parámetros que recibe)

¿Que devuelve? un dato de tipo int, float, char, etc o nada void

¿Nombre?, cualquiera, que sea indicativo de lo que hace, siempre sin espacios

¿Qué parámetros recibe?, serían los datos que yo le envío a la función para que realice una determinada acción y me devuelva un resultado

Primer tipo de funciones, donde sus parámetros se pasan por pasaje de valor

Que significa esto que los datos que yo les pase a la función podrán ser modificados en la misma por distintas operaciones, pero cuando la función termina de ejecutarse y vuelve el control al main, esas variables vuelven a su valor original

Antes del main la "declaro", digo que existirá algo nuevo para que el programa sepa de qué se trata lo que luego llamare desde el main

Int calcular (int, int); acá estoy diciendo que existirá una función calcular que devolverá un entero y que para funcionar necesita que le envié dos variables enteras

```
Int main()
{
Int sdo, ch, vh;
Sdo= calcular (ch, vh);          acá la llamo, la invoco
Printf ("el sueldo es %d", sdo);
}
```

Ahora luego del main la "defino", digo que hace

```
Int calcular (int a, int b)      fíjense que no lleva "; "
    {
    int c;                       estas variables son locales de la función
    c = a * b;                   solo existen al llamar a la función
    return c;                    mientras no ocupan lugar
    }
```

Como se ve este tipo de funciones sirve para multiplicar, dividir, alguna operación matemática, no tanto para sumas o restas

¿porque?, porque en general en esas funciones una de las variables va a cambiar de valor y necesito quedarme con ese Nuevo valor, entonces uso las otras funciones llamadas por referencia

¿Cuál es la diferencia fundamental? que yo no le paso una copia del valor de la variable, sino que por medio de un puntero le paso la dirección de memoria de la variable a modificar

```
Void sumar( int* , int);         declaro
Int main()
    {
    Int total,sueldo;
    Sumar(&total,sueldo);        invoco
```

```
        Printf ("%d", total);
        }
Void sumar (int * a, int b)          defino
        {
        *a = *a + b;
        }
```

Otro uso es por ejemplo en el cálculo del máximo o mínimo dentro del ciclo

```
Void máximo (int* , float* , int, float,int *);          declaro
Int main()
        {
        Int legajo, aux, con=0 ;
        Float sueldo,max ;
        Máximo(&aux,&max, legajo,sueldo,&con);          invoco
        Printf("el sueldo maximo es %5.2f del legajo %d", max, aux);
        }
Void máximo (int* a , float* b , int c, float d ,int * e)    defino
        {
If ( *e == 0 )
        {
        *e=1;
        *b=sueldo;
        *a = legajo;
```

```
        }
If( d > *b )
        {
        *b= sueldo;
        *a= legajo;
        }
}
```

Yo podría nombrar a las variables locales de las funciones con el mismo nombre que las variables que vienen del main, no habria problema, pues al ser locales son otras variables, aunque su nombre sea exactamente el mismo

Veamos ahora el caso de los vectores y matrices

Por definición del lenguaje ambas estructuras siempre se pasan por referencia, o sea que sus valores pueden ser cambiados en la función y no hace falta pasarlos con un puntero &

Seria, por ejemplo

```
Void calcular (int [ ],int [ ][10 ])    declare un vector y una matriz, obligatorio
                                        poner la cantidad de columnas
int main()
        {
        Calcular ( vector, matriz );            invoco
        }
Void calcular ( int a[15],b[15][10])                    define que hace la función
        {
        }
```

Por ultimo veamos los apuntes que si hacemos funciones para cargar datos no debemos poner & en el scanf porque se produciría la llamada de un puntero a otro puntero

Como consejo les diría inicialmente escribir el programa entero y luego de a una ir hacienda las funciones para saber por si falla, donde está el error

El ejercicio que a continuación les pongo, NO hace nada serio ni quizás lógico, pero lo que pretendo mostrarles es como se declaran, invocan y definen las funciones

```
#include<stdio.h>
#include<conio.h>
#include<stdlib.h>
#include<iostream>
void compara(int *,int*,int,int *);
int multiplicar(int,int);
void sumame(int*,int);
void cargarvector(int[],int);

int main()
{
	int vcp[11]={0};
	int np,cat,I,hs=10,tot,max,aux,cont=1;
	printf("ingrese el nro de publicacion  ");
	scanf("%d",&np);
	while(np!=0)
		{
		printf("ingrese la categoria  ");
		scanf("%d",&cat);
		tot= multiplicar(cat, hs);  //por pasaje de vaslor
		sumame(&tot,hs);                    // referencia y pasaje de valor
		compara(&max,&aux,tot,&cont);
		cargarvector(vcp,cat);              // referencia de vectores

		printf("ingrese el nro de publicacion  ");
		scanf("%d",&np);
		}
	//comienzo a responder
	system("cls");
	for(I=1;I<=10;I++)
		{
			if(vcp[I]!=0)
			{
			printf("la categoria %d, publico %d avisos \n",I,vcp[I]);
			}
		}
}
```

```
void compara(int *max,int*aux,int tot,int * cont)
	{
		if(*cont==1)
			{
			*max=tot;
			*aux=*cont;
			*cont=*cont+1;
			}
		if(tot>*max)
			{
			*max=tot;
			*aux=*cont;
			}
	}
void cargarvector(int vcp[11],int cat )
		{
			vcp[cat]=vcp[cat]+1;
		}

int multiplicar(int a,int b)
		{
		int c;
	   c= a*b;
	   return c;
		}

void sumame(int* a,int b)
		{
			*a=*a+b;
		}
```