UNIDAD 3.- FUNCIONES.

OBJETIVOS:

- ✓ Que el alumno aprenda a realizar correctamente las funciones en un lenguaje de programación.
- ✓ Que el alumno utilice adecuadamente los tipos de funciones en lenguaje C.

TEMAS:

3.1. Definición de función.
3.2. Variables globales, variables locales, ámbito de validez de cada una de ellas.
3.3. Tipos de funciones. Cabeceras, parámetros, sentencias de llamada.
3.4. Recursividad.
3.5. Implementación de funciones en lenguaje C.

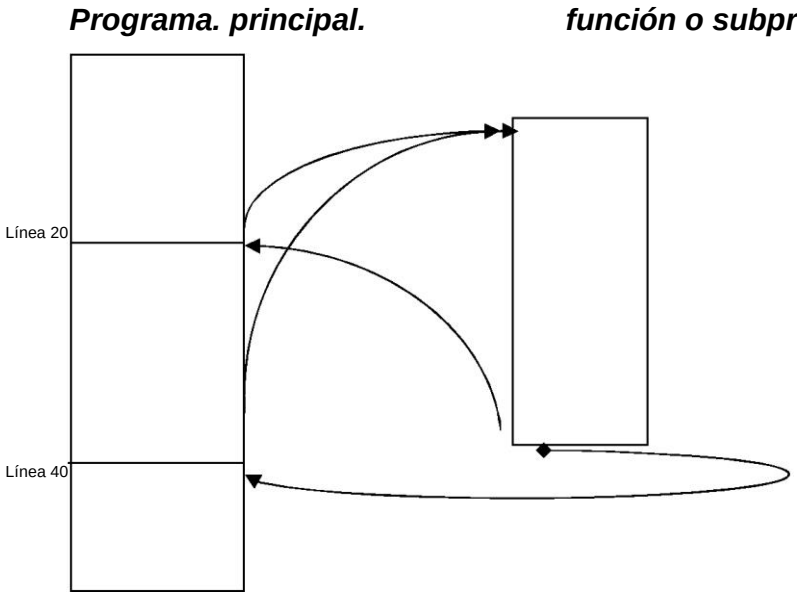
3.1. DEFINICIÓN DE FUNCIÓN

Frecuentemente, los programadores se encuentran frente al problema de construir un algoritmo el cual tiene características especiales: por ejemplo, la reiteración de determinados cálculos, los cuales son a veces tediosos en su escritura o complejos en la tarea que realizan.

Los lenguajes de alto nivel incorporan lo que se denominan **funciones** o **subprogramas** que realizan trabajos o tareas para facilitar la confección del código de un programa o para una mejor legibilidad del mismo.

Definiremos entonces, el término **función** para cualquier tipo de lenguaje:

Se denomina **función**, a un conjunto de tareas (representadas por líneas de programa) que se ejecutan cuando el programador determina (llamada a la función, incluida dentro del programa principal), y que no forma parte del programa principal. Luego de la ejecución de las mismas, el control del programa vuelve al programa principal a continuar con el resto de las acciones especificadas en el mismo.



- Se supone la existencia de un programa principal, en el cual, en la línea 20 está escrita la directiva para que se inicie la ejecución de la función o subprograma (también llamado **subrutina)**.
- Se dice que el programa principal ***llama o invoca*** al subprograma.
- En ese momento, el control del programa abandona al Programa Principal y va a ejecutar la misma.
- Terminada la ejecución de la función, el control del programa vuelve a la línea **21**, que es la siguiente a aquélla desde donde se indicó que la función debía ejecutarse.
- Se repite el llamado en la línea 40, realizando el mismo procedimiento que en el caso anterior.

Las acciones que lleva a cabo la función es lo que se denomina **definición de la función.**

3.2. VARIABLES GLOBALES, VARIABLES LOCALES, ÁMBITO DE VALIDEZ

Según el lugar dónde se declaren las variables, las mismas pueden ser globales o locales teniendo cada una de ellas un distinto ámbito de validez.

Cuando construimos una función, en el código de ésta pueden o no declararse variables ya que es un programa similar a main ().

VARIABLES GLOBALES Y ÁMBITO DE VALIDEZ

Las variables declaradas FUERA DE TODAS LAS FUNCIONES (incluida main ()) se denominan ***VARIABLES GLOBALES*** y pueden accederse o ser usadas en cualquier función incluida main ().

Ejemplo: Si las variables **a** y **b** son declaradas antes de main (), ellas pueden ser ingresadas o mostradas en la función **main ()** y en toda otra función que dependan de ella.

```
#include <stdio.h>

int a,b

main ()
{
   .....;
}
```

VARIABLES GLOBALES

VARIABLES LOCALES Y ÁMBITO DE VALIDEZ

Las variables declaradas en cada función, se denominan ***VARIABLES LOCALES***, y sólo son válidas en la misma, es decir no pueden ser accedidas o usadas en otras funciones, ni tampoco en main (), debido al "ámbito de validez de una variable".

```
#include <stdio.h>
main ()
{
      int a, b
      ……;
}
```

VARIABLES LOCALES

3.3. TIPOS DE FUNCIONES. CABECERAS, PARÁMETROS, SENTENCIAS DE LLAMADA.

En el caso del lenguaje **C** tenemos dos grandes tipos de funciones (o subprogramas): las **funciones standards** y las **funciones declaradas por el usuario**.

Las **funciones standard** en **C** son las que el lenguaje ya tiene incorporadas.

FUNCIONES STANDARD

Todas las instrucciones codificadas dentro de main (), corresponden a lo que se denomina programa principal, llamada función principal. En consecuencia, todo algoritmo en C debe ser codificado dentro de la función principal o main ().

Otras funciones **standard** que se usaron son:

- ✓ **scanf (…)**
- ✓ **printf(…)**.

Ambas directivas son realmente funciones, las cuales poseen argumentos los cuales están encerrados entre paréntesis. Casi todas las directivas de ejecución de acciones en C, corresponden a funciones standard debidamente establecidas por el compilador.

FUNCIONES CREADAS O DECLARADAS POR EL USUARIO

Las funciones creadas por el usuario no están incorporadas en los compiladores (como lo están las standard), y es el usuario quien debe escribirlas de acuerdo con las reglas de sintaxis que el lenguaje impone.

NOMBRE DE LA FUNCIÓN

Se trata de un identificador válido en **C,** es decir que no corresponda a palabras reservadas.

Generalmente se recomienda dar nombres relacionados con lo que la función lleva a cabo (suma, cálculo, promedio, etc.).

La definición de la función se acostumbra a realizarla después del programa principal o main().

El bloque de escritura de la función es similar en consecuencia a main (), salvo su nombre, el valor devuelto y sus argumentos.

PARÁMETROS

Son los valores que la función recibe desde el programa principal.
En una función, los datos de entrada vienen del programa principal y los resultados son enviados al mismo (que produjo la llamada). Para producir este intercambio de información, se utilizan las variables de enlace o parámetros.

TIPOS DE PARÁMETROS DE UNA FUNCIÓN

Los parámetros que figuran en:

- ✓ la ***llamada*** de una función se denominan parámetros **actuales**,

- ✓ los que están como argumentos en la cabecera de la función**,** se denominan parámetros **formales.**

El tipo y número de datos enunciado en el prototipo o **declaración** de la función se corresponden con los del **llamado** a la función. Esta correspondencia se realiza por posición y naturaleza.

Existen dos tipos de parámetros:

- Parámetros por valor.
- Parámetros por referencia.

DEFINICIÓN DE LA FUNCIÓN

En esta sección, se especifica que acciones realiza la función. Es decir, se escriben las instrucciones que realizaran la tarea o cálculo para lo cual fue escrita la misma.

Las funciones creadas por el usuario son de dos tipos:

1. Funciones SIN tipo.
2. Funciones CON tipo.

FUNCIONES SIN TIPO

Toda función cuando es ejecutada **puede o no devolver un valor** (resultante del cálculo o proceso que dicha función lleva a cabo). Dicho valor puede ser cualquier tipo de dato de los ya estudiados (entero, flotante, carácter, etc.) como también acciones que el usuario establezca. En el caso de no retornar ningún valor, se dice que dicha función es una función **SIN TIPO.**

Este tipo de funciones se utiliza para:

- Realizar un trabajo. Ej. Imprimir títulos.
- Calcular o trabajar con un conjunto de valores. Ej. Leer un vector.
- Modularizar la solución.

FUNCIONES SIN TIPO: CABECERA

En las funciones sin tipo la cabecera (en lenguaje C) debe contener la palabra void, el nombre de la función y la lista de parámetros, indicando de qué tipo es cada uno de ellos.

void nombre de la función (lista de parámetros);

Ej.:

void mayor (int a1, int b1, int c2)

FUNCIONES SIN TIPO: LLAMADA

En un diagrama de flujo la llamada e indica con el siguiente bloque:

	NOMBRE DE LA FUNCIÓN (LISTA DE PARÁMETROS)	

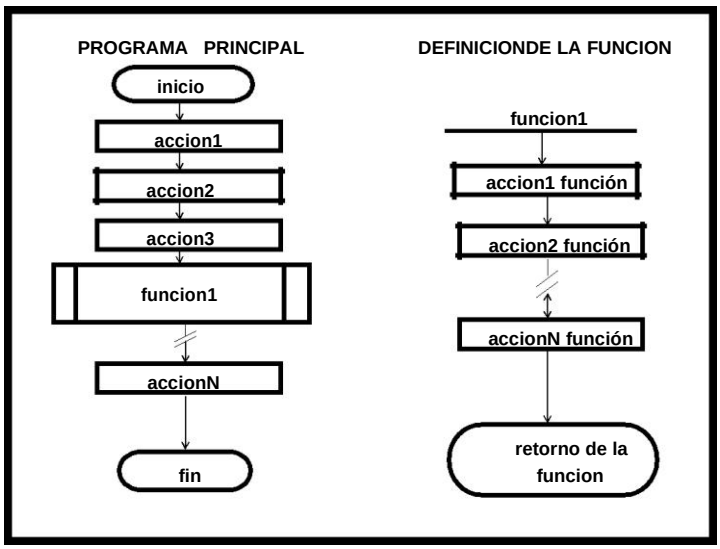
En la codificación la llamada a una función es una INSTRUCCIÓN en lenguaje C, compuesta por el nombre de la función y la lista de parámetros (si los hubiere), encerrados entre paréntesis.

Nombre de la función (lista de parámetros);

Ej:

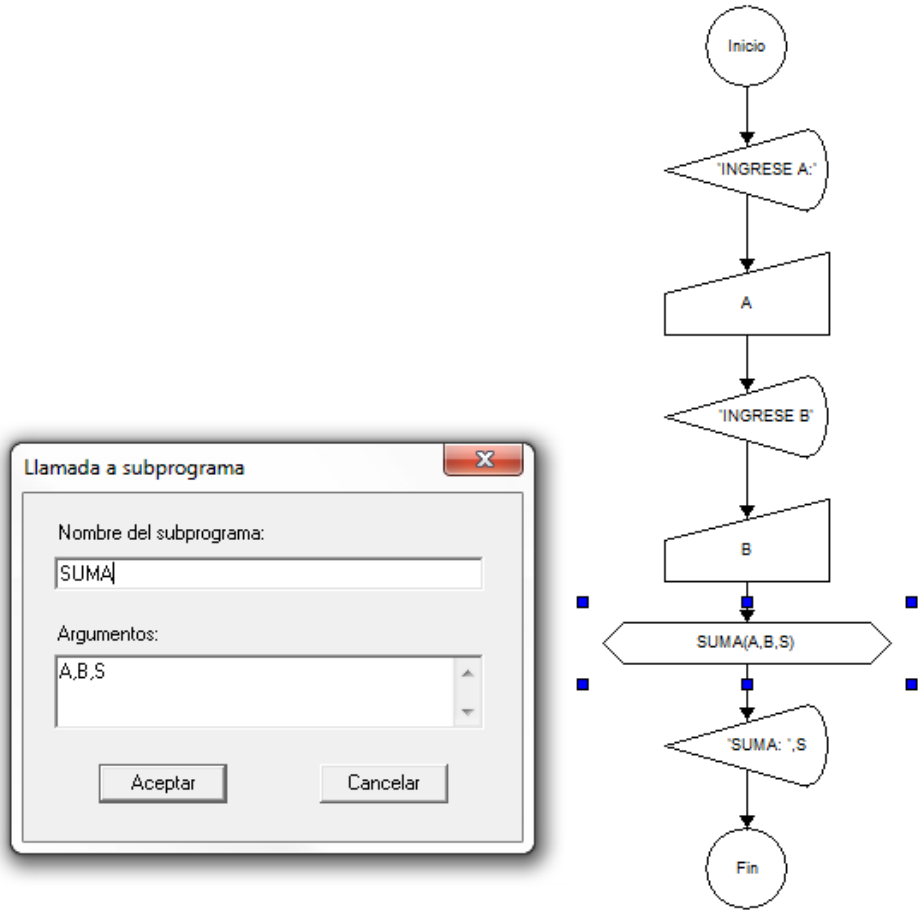
mayor(a,b,c);

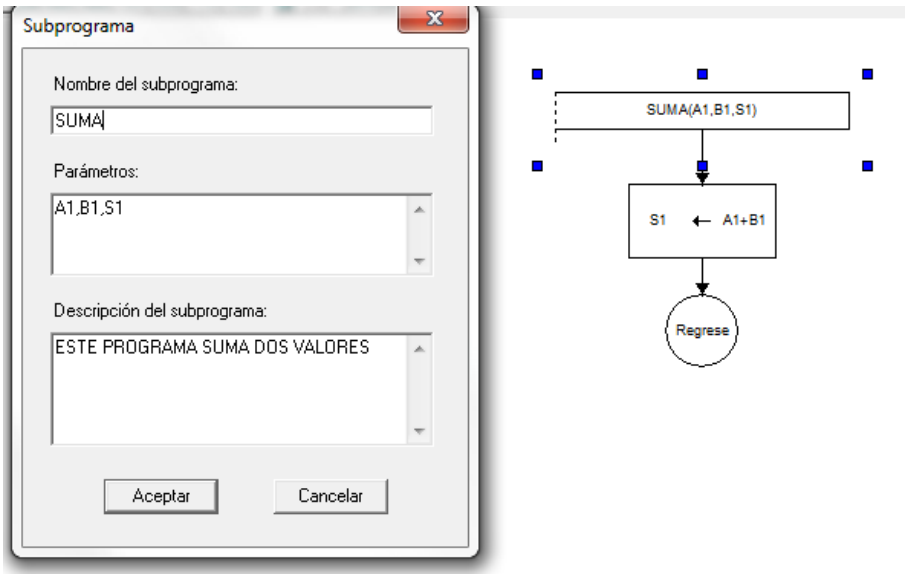
Ejemplo de un programa principal y el desarrollo de una función SIN tipo.



Ejemplo

Ingresar dos valores a y b, a través de una función sin tipo encontrar el valor de la suma y mostrar el resultado en la función principal.





La codificación en lenguaje C:

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

void suma(int a1, int b1, int &s1)
{
        s1=a1+b1;
}

main ()
{
        int a,b,s;
        printf("Ingrese el valor de a: ");
        scanf("%d",&a);
        printf("Ingrese el valor de b: ");
        scanf("%d",&b);
        suma(a,b,s);
        printf("Suma: %d",s);
        getch();
}
```

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

void suma(int a1, int b1, int &s1);


main ()
{
        int a,b,s;
        printf("Ingrese el valor de a: ");
        scanf("%d",&a);
        printf("Ingrese el valor de b: ");
        scanf("%d",&b);
        suma(a,b,s);
        printf("Suma: %d",s);
        getch();
}

void suma(int a1, int b1, int &s1)
{
        s1=a1+b1;
}
```

FUNCIONES CON TIPO

Cuando una función **devuelve un valor** (resultante del cálculo o proceso que dicha función lleva a cabo), se dice que es una función **CON TIPO.** Dicho valor puede ser cualquier tipo de dato de los ya estudiados (entero, flotante, carácter, etc.)

Este tipo de función se utiliza para:

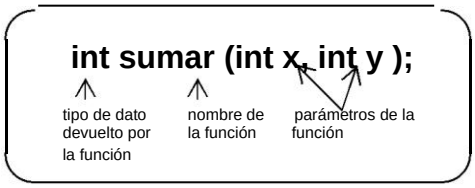
- Calcular un único valor.

FUNCIONES CON TIPO: CABECERA

El formato general de la declaración de una función con tipo es:

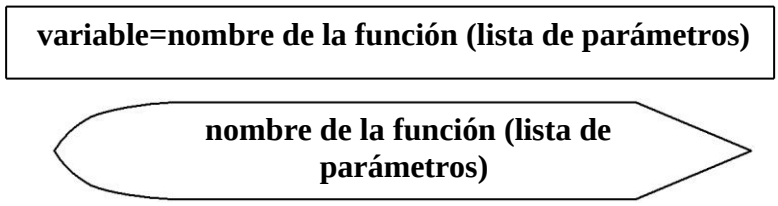
***<tipo de dato devuelto por la función> <nombre de la función>* (*parámetros*);**

Ejemplo: Escribir una función que reciba dos argumentos enteros, los sume y devuelva el valor de dicha suma



FUNCIONES CON TIPO: LLAMADA

En un diagrama de flujo la llamada puede formar parte de un bloque de asignación o de salida:



En la codificación la llamada a una función es una PARTE de una INSTRUCCIÓN en lenguaje C; puede formar parte de una instrucción de asignación o de salida.

Ej:

```
s = sumar (a, b);
printf("la suma de los valores es %d",s);
```

Ejemplo:

Ingresar dos números**,** y calcular la suma de los mismos mediante una función con tipo, mostrar el resultado en main ()

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

int suma(int a1, int b1)
{
        int s1;
        s1=a1+b1;
        return s1;
}


main ()
{
        int a,b,s;
        printf("Ingrese el valor de a: ");
        scanf("%d",&a);
        printf("Ingrese el valor de b: ");
        scanf("%d",&b);
        s=suma(a,b);
        printf("Suma: %d",s);
        getch();
}
```

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

int suma(int a1, int b1);

main ()
{
        int a,b;
        printf("Ingrese el valor de a: ");
        scanf("%d",&a);
        printf("Ingrese el valor de b: ");
        scanf("%d",&b);
        printf("Suma: %d",suma(a,b));
        getch();
}

int  suma(int a1, int b1)
{
        int s1;
        s1=a1+b1;
        return s1;
}
```

PARÁMETROS POR VALOR – PARÁMETROS POR REFERENCIA

PARÁMETROS POR VALOR

Los parámetros por valor (un tipo de parámetros formales) realizan duplicación de memoria, ya que definen nuevas localizaciones de memoria y realizan una transferencia unidireccional de los valores que se encuentran en los parámetros actuales. Esto implica que los cambios de las variables en la función no producen alteración en las variables de donde recibieron su valor.

PARÁMETROS POR REFERENCIA

Los parámetros por referencia (otro de los tipos de los parámetros formales) utilizan las mismas localizaciones de memoria que los parámetros actuales. Esto implica que los cambios de

las variables en la función producen alteración en las variables de donde recibieron su valor.

Son aquellos que se incluyen como argumento de funciones y cuyo valor cambia, si cambia el valor de la variable correspondiente dentro de la función.

Para usar parámetros por referencia, se debe:

- En la declaración de la función, anteponer el operador **&** a las variables que serán consideradas como parámetros por referencia en lenguaje C.

FUNCIONES RECURSIVAS

La recursividad forma parte del repertorio para resolver problemas en Computación y es de los métodos más poderosos y usados.

Los algoritmos recursivos ofrecen soluciones estructuradas, modulares y elegantemente simples.

La recursividad es un concepto fundamental en matemáticas y en computación. Una definición recursiva dice cómo obtener conceptos nuevos empleando el mismo concepto que intenta describir.

En toda situación en la cual la respuesta pueda ser expresada como una secuencia de movimientos, pasos o transformaciones gobernadas por un conjunto de reglas no ambiguas, la fórmula recursiva es un buen candidato para resolver el problema.

Los razonamientos recursivos se encuentran en la base misma de las matemáticas porque son necesarios para describir conceptos centrales como el de número:

Uno de los ejemplos más clásicos es el **factorial de un número**. Para cualquier entero positivo N, el factorial de N (que se expresa como N!) es el producto (multiplicación) de todos los enteros menores a él:

- 1! = 1
- 2! = 1 x 2 = 2
- 3! = 1 x 2 x 3 = 6
- 4! = 1 x 2 x 3 x 4 = 24
- 5! = 1 x 2 x 3 x 4 x 5 = 120
- 6! = 1 x 2 x 3 x 4 x 5 x 6 = 720

Por lo tanto: 2! es 1! x 2 3! es 2! x 3

Y así sucesivamente. Para cualquier entero N mayor a 1, podemos decir que el factorial de N es igual al factorial del número anterior a N multiplicado por N.

La fórmula ….. N! = (N-1)! x N.

Las funciones recursivas son aquellas en las cuales, dentro del código de la función, existe una llamada a la misma función.

Para que una función recursiva esté completamente definida es necesario:

- tener un **caso base** que no se calcule utilizando casos anteriores y
- que la división del problema converja a ese **caso base.**

La recursividad es una técnica muy útil que puede utilizarse en lugar de las repeticiones, dando una visión diferente de las iteraciones.

La capacidad de una función para llamarse a sí misma, se conoce como **recursividad,** y a los algoritmos así implementados se los conoce como **algoritmos recursivos.**

En un algoritmo recursivo distinguimos como mínimo 2 partes:

a) Caso trivial, base o de fin de recursión:
Es un caso donde el problema puede resolverse sin tener que hacer uso de una nueva llamada a sí mismo. Evita la continuación indefinida de las partes recursivas.

b) Parte puramente recursiva:
Relaciona el resultado del algoritmo con resultados de casos más simples. Se hacen nuevas llamadas a la función, pero están más próximas al caso base.

EJEMPLO

Escribir un programa para sumar los números consecutivos entre **0** y **n,** siendo n un número ingresado por el usuario. Mostrar el resultado en la función main ().

Codificación en C

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

int suma(int n1);

main()
{
        int n,s;
        printf("Valor n: ");
        scanf("%d",&n);
        s=suma(n);
        printf("Suma: %d",s);
        getch();
}

int suma(int n1)
{
   int s;
   if (n1==0)
      s=0;
   else
      s=n1+suma(n1-1);
   return s;
}
```