

Una **Función** o un **Procedimiento** es un bloque de código dentro del programa que se encarga de realizar una tarea determinada.

En el caso del lenguaje c ambos conceptos se denominan funciones.

Por lo tanto un programa en **c** debe constar de una o más **funciones**, y por supuesto no puede faltar la **función** principal main().

De todas maneras, haremos diferencia entre cada uno de estos conceptos.

¿Para qué se usan las funciones y procedimientos?

<u>Grupo de sentencias que se repiten.</u>	<u>Jerarquización o modularización.</u>
Si en un programa hay varias sentencias que se repiten, estas pueden aislarse en una función o procedimiento y luego se invoca al mismo. Simplemente con una sentencia de invocación, se ahorran líneas de código.	Se pueden agrupar sentencias que cumplen un objetivo particular y luego se los invoca.  En este caso, no se ahorran líneas de código pero se ordena y se le asigna una jerarquía al grupo total de sentencias.

**Recordemos cómo se arma un programa en C:**

<pre>#include &lt;stdio.h&gt; #include &lt;stdlib.h&gt; //----- int main() {int x,y, z; x=5; y=7; z=x+y; printf("\n La suma de %d y %d es %d ",x,y, z); printf("\n"); system("PAUSE"); return 0; } //-----</pre>	<div>} Declaración de librerías</div> <div>→ Programa principal → Declaración de variables</div> <div>} Sentencias del programa principal</div>
--	---

## ¿Qué diferencia existe entre las funciones y los procedimientos?

Las **funciones** ejecutan un grupo de sentencias y *devuelven un resultado* en su nombre.

Los **procedimientos** también ejecutan un grupo de sentencias pero en su nombre no devuelven nada, lo hacen a través sus parámetros. Pueden *devolver un resultado, varios o ninguno*.

Vamos a explicar esto a través de ejemplos.

### Procedimientos.

```
void pausar()  
{ printf("\n");  
  system("PAUSE");  
}
```

Para escribir un procedimiento debemos colocar void delante del nombre del mismo; luego entre paréntesis el o los parámetros o ninguno, y finalmente entre llaves el cuerpo de sus sentencias.

Pausar es un procedimiento que solo ejecuta dos sentencias.

Para invocar a un procedimiento, se coloca simplemente su nombre seguido de sus parámetros o ninguno, como en este caso. La invocación es una sentencia, por lo tanto demos poner ; al final.

En el main quedará

pausar();  
En nuestro ejemplo, quedará:

```
#include <stdio.h>  
#include <stdlib.h>  
//-----  
void pausar()  
{ printf("\n");  
  system("PAUSE");  
}  
//-----
```

```

int main()
{int x,y, z;
x=5;
y=7;
z=x+y;
printf("\n La suma de %d y %d es %d ",x,y, z);

pausar( );
return 0;
}
//-----

```

Observar que el procedimiento se escribió luego de los include y antes del main.

Otra forma:

```

#include <stdio.h>
#include <stdlib.h>
//-----
void pausar();
//-----

int main()
{int x,y, z;
x=5;
y=7;
z=x+y;
printf("\n La suma de %d y %d es %d ",x,y, z);
pausar( );
return 0;
}
//-----

void pausar()
{ printf("\n");
  system("PAUSE");
}
//-----

```

Observar que el procedimiento se escribió abajo del del main y se debe escribir solo la parte declarativa arriba del main.

Cualquiera de las dos forma es correcta, solo debemos elegir una y luego sostener la elección. Simplemente por mantener un estilo de programación.

Ahora escribiremos un procedimiento con tres parámetros.

```
void mostrar(int a, int b, int c )
```

```
{ printf("\n La suma de %d y %d es %d ",x,y, z);  
}
```

Los parámetros son a, b y c. Son los datos que necesita el procedimiento para ejecutarse. En este caso son parámetros de entrada.

Finalmente quedará:

```
#include <stdio.h>  
#include <stdlib.h>  
//-----  
void pausar();  
void mostrar(int a, int b, int c);  
//-----  
  
int main()  
{int x,y, z;  
x=5;  
y=7;  
z=x+y;  
mostrar(x,y,z);  
pausar( );  
return 0;  
}  
//-----  
void pausar()  
{ printf("\n");  
  system("PAUSE");  
}  
//-----  
void mostrar(int a, int b, int c )  
{ printf("\n La suma de %d y %d es %d ",a,b,c);  
}  
//-----
```

En este caso, desde el programa main, los valores de las variables x, y, z se copian en a, b, c en ese mismo orden.

## Funciones.

Una función la constituyen un grupo de sentencias con tareas específicas que devuelven un solo resultado. Ese único resultado la función lo devuelve en su nombre.

Vamos a explicar esto a través de ejemplos.

<pre>int suma(int a, int b) {int res; res=a+b; return(res); };</pre>	<table border="0"><tr><td>}</td><td>Parte declarativa de la funcion</td></tr><tr><td>}</td><td>Cuerpo de la función (entre paréntesis)</td></tr></table>	}	Parte declarativa de la funcion	}	Cuerpo de la función (entre paréntesis)	<div style="border: 1px solid black; padding: 10px;"><p>suma es el <i>nombre</i> de la función</p><p>a y b son <i>parámetros de entrada</i> de la función</p><p>res es <i>variable local</i> a la función</p><p>sentencia return para <i>devolver el resultado</i></p></div>
}	Parte declarativa de la funcion					
}	Cuerpo de la función (entre paréntesis)					

La función suma recibe dos valores enteros a y b los cuales se suman en el interior de la función. Dicho resultado se devuelve con la sentencia return.

Desde el main se invoca a la función de la siguiente manera:

```
x=2;
```

```
y=4;
```

```
z=suma(x,y);
```

```
printf("la suma de %d y %d es %d", x, y, z);
```

```
o
```

```
printf("la suma de %d y %d es %d", x, y, suma(x,y));
```

```
o
```

```
if (suma(x,y) >0)
```

```
    printf("la suma es positiva");
```

```
else
```

```
    printf("la suma es negativa o cero");
```

Finalmente quedará:

```
#include <stdio.h>
#include <stdlib.h>
//-----
void pausar();
void mostrar(int a, int b, int c);
int suma(int a, int b);
//-----

int main()
{int x,y, z;
x=5;
y=7;
z=x+y;
mostrar(x,y,z);
x=2;
y=4;
z=suma(x,y);
printf("la suma de %d y %d es %d", x, y, z);
printf("la suma de %d y %d es %d", x, y, suma(x,y));
pausar( );
return 0;
}
//-----
void pausar()
{ printf("\n");
  system("PAUSE");
}
//-----
void mostrar(int a, int b, int c )
{ printf("\n La suma de %d y %d es %d ",a,b,c);
}
//-----
int suma(int a, int b)
{int res;
res=a+b;
return(res);
};
```

Irso. Estructuras de Datos. Taller II. Lic. Paula Di Rocco.