Apunte 1: FUNCIONES EN C

El lenguaje C tiene como bloque básico la función main(). Main() es una función, printf() otra, y hay muchas más funciones predefinidas, pero nosotros mismos también podemos definir nuestras propias funciones. De hecho, es fundamental hacerlo.

Podemos definir una función cualquiera de la misma manera en que definimos la función main(). Basta con poner su tipo, su nombre, sus argumentos entre paréntesis y luego, entre llaves, su código:

```
/* Inclusión de archivos */
#include <stdio.h>

void holamundo(void) /*Función donde se ejecuta la lógica del programa*/
{
	printf("Hola Mundo\n"); /*imprime la cadena*/
	return; /*sale de la función*/
}

int main(void) /*Función principal del programa*/
{
	holamundo(); /*llamada a la función que lleva el peso*/
	return 0; /*sale del programa: correcto*/
}
```

Este código nos muestra cómo escribir y cómo utilizar una función. Y además nos muestra un principio de buena programación: meter las sentencias que "hacen el trabajo" en otras funciones específicas para sacarlas de main(), dejando en ésta tan sólo un guión general de lo que hace el programa, no las órdenes específicas. De esta manera se facilita la comprensión del programa, y por tanto el futuro trabajo de modificarlo.

De la misma manera que tenemos que declarar una variable antes de utilizarla, no es indiferente el orden en que se sitúen las diferentes funciones en el fichero: las funciones deben declararse antes de ser llamadas.

Igualmente vemos que, para una función void, la sentencia de control return no puede llamarse como pseudofunción, porque en dicho caso la función void (en nuestro casoholamundo()) devolvería un valor, cosa que su definición no permite.

Las funciones también permiten recibir tipos de datos, así pues las funciones nos sirven para hacer de un gran problema pequeñas partes de un problema o sea dividir un gran problema en diferentes problemas más pequeños. Así que las funciones también pueden retornar un tipo de dato que hemos definido dentro de la misma.

La definición de una función para sumar dos números sería de la siguiente manera:

```
/*  suma.c */
```

```c
#include <stdio.h>

int sumar(int numero1, int numero2); /* prototipo de la función */

int main(void)
{
	int suma; /* definimos una variable*/

	suma = sumar(5, 3);
	/* la variable obtiene el valor retornado de sumar
	 * puede verse que entre los paréntesis se mandan los valores
	 * de los números que se desean sumar en este caso 5 y 3
	 * pueden haberse declarados estos dos números en otras variables
	 * int dato = 4, valor = 3;
	 * suma = sumar(dato, valor);
	 */

	/* imprimimos la suma de los dos números */
	printf("La suma es: %d ", suma);

	return 0;
}

int sumar(int numero1, int numero2)
{
	int retsuma; /* creamos la variable a retornar*/

	retsuma = numero1 + numero2; /* asignamos a esa variable la suma de número 1 y 2*/

	return retsuma; /* retornamos la suma de los números */
}
```

PASO DE PARÁMETROS

Las funciones pueden recibir datos como lo hemos observado, pero existen dos formas importantes de enviar los datos hacia una función **por valor** y **por referencia**, con las cuales observarse que son muy diferentes y que una puede ser muy diferente de la otra, haciendo modificaciones en nuestro programa.

Por Valor

El paso por valor envía una copia de los parámetros a la función por lo tanto los cambios que se hagan en ella no son tomados en cuenta dentro de la función main(). Ejemplo:

```c
/** por_valor.c */

#include <stdio.h>
```

```
void sumar_valor(int numero); /* prototipo de la función */

int main(void)
{
	int numero = 57; /* definimos numero con valor de 57*/

	sumar_valor(numero); /* enviamos numero a la función */

	printf("Valor de numero dentro de main() es: %d\n", numero);
	/* podemos notar que el valor de numero se modifica
	 * sólo dentro de la función sumar_valor pero en la principal
	 * número sigue valiendo 57
	 */

	return 0;
}

void sumar_valor(int numero)
{
	numero++; /* le sumamos 1 al numero */

	/* el valor de número recibido se aumenta en 1
	 * y se modifica dentro de la función sumar_valor()
	 */
	printf("Valor de numero dentro sumar_valor() es: %d\n", numero);

	return;
}
```

Por Referencia

El paso por referencia se hace utilizando apuntadores. Se envía la dirección de memoria de la variable, por lo tanto los cambios que haga la función si afectan el valor de la variable. Ejemplo:

```
/** por_referencia.c */

#include <stdio.h>

void sumar_referencia(int *numero); /* prototipo de la función */

int main(void)
{
	int numero = 57; /* definimos numero con valor de 57*/

	sumar_referencia(&numero); /* enviamos numero a la función */

	printf("\nValor de numero dentro de main() es: %d ", numero);
	/* podemos notar que el valor de numero se modifica
	 * y que ahora dentro de main() también se ha modificado
	 * aunque la función no haya retornado ningún valor.	*/
```

```
	return 0;
}

void sumar_referencia(int *numero)
{
	*numero += 1; /* le sumamos 1 al numero */

	/* el valor de numero recibido se aumenta en 1
	 * y se modifica dentro de la función
	 */
	printf("\nValor de numero dentro sumar_referencia() es: %d", *numero);

	return;
}
```

VARIABLES LOCALES Y GLOBALES

Además de pasar valores a una función, también se pueden declarar tipos de datos dentro de las funciones, estos tipos de datos declarados dentro de una función solo son accesibles dentro de esta misma función y se les conocen como variables locales, así pues podemos definir los mismos nombres de variables en diferentes funciones, ya que estas variables solo son accesibles dentro de esas funciones. Ejemplo:

```
/** locales.c */

#include <stdio.h>

void funcion1()
{
	int dato = 53; /* definimos dato en 53*/
	char num1 = 'a'; /* num1 vale a */

	/* imprimimos */
	printf("Funcion1, dato=%d, num1=%c\n", dato, num1);

	return;
}

void funcion2()
{
	int dato = 25; /* definimos dato en 25*/
	char num2 = 'z'; /* num2 vale z*/

	/* imprimimos */
	printf("Funcion2, dato=%d, num2=%c\n", dato, num2);

	return;
```

```
}

int main(void)
{
	funcion1(); /* llamamos a funcion1() */

	funcion2(); /* llamamos a funcion2() */

	return 0;
}
```

En este caso la variable dato, esta definida dentro de cada una de las funciones y son totalmente distinta una de otra y no se puede utilizar fuera de esta, así pues num2 no puede ser utilizada por la funcion1() y num1 tampoco puede ser utilizada por funcion2().

Existen pues variables que se definen fuera de la función principal main() y fuera de cualquier otra función creada por nosotros, estas variables se les conoce con el nombre de Variables Globales ya que se pueden utilizar dentro de main() y dentro de cualquier función creada por nosotros. Ejemplo:

```
/** global.c */

#include <stdio.h>

int variable_global = 99; /* inicializamos la variable global */

void funcion();

int main(void)
{
	/* imprimimos el valor*/
	printf("main(), acceso a variable_global %d\n", variable_global);

	/* llamamos a la función */
	funcion();

	return 0;
}

void funcion()
{
	/* imprimimos el valor*/
	printf("funcion(), acceso a variable_global %d\n", variable_global);

	return;
}
```

FUNCIONES RECURSIVAS

La recursividad (recursión) es la propiedad por la cual una función se llama a sí misma **directa** o **indirectamente**. La recursión indirecta implica utilizar más de una función.

Se puede considerar la recursividad como una alternativa a la iteración. La recursión permite especificar soluciones naturales, sencillas, que serían, en caso contrario, difíciles de resolver. Toda función recursiva debe contemplar un **caso base** o **condición de salida**, para terminar, o la recursividad no podrá terminar nunca.

Una función recursiva podría definirse así:

```
/* tipo */ funcion_recursiva( /* parámetros recibidos por la función
*/ )
{
        /* Código */
        funcion_recursiva( ); /* llamada a la función misma */
        /* Código */
}
```

Uno de los ejemplos más representativos en la recursividad es el factorial de un numero (n!):

$$n! = \prod_{k=1}^{n} k \qquad \forall n \in \mathbb{N}$$

la definición de recursividad del factorial es:

$$n! = \begin{cases} 1 & \text{Si } n = 0 \\ n(n-1)! & \text{Si } n > 0 \end{cases} \qquad \forall n \in \mathbb{N}.$$

En esta definición, n = 0, es nuestro caso base, que le da fin a la recursividad.

Entonces nuestro programa que calcula el factorial es:

```
/* factorial.c */

#include <stdio.h>

long factorial(int n)
{
        if (n == 0) /* caso base */
                return 1; /* como 0! = 1, se retorna 1*/
        else
                return n * factorial (n - 1); /* llamada a esta misma
función */
}

int main(void)
{
```

```
        /* en este caso se llama a la función y se imprime
directamente*/
        printf("%ld ", factorial(5));

        return 0;
}
```

También existen otros tipos de funciones recursivas como lo es el producto de dos números. El producto de **a** ✕ **b**, donde **a** y **b** son números enteros positivos seria:

Solución iterativa:

$$a \times b = \underbrace{a + a + \cdots + a}_{b\ veces} = \sum_{i=1}^{b} a$$

Solución recursiva:

$$a \times b = \begin{cases} 0 & \text{Si } b = 0 \\ a + a \times (b-1) & \text{Si } b > 0 \end{cases}$$

Así pues 7×3 es:

$$7 \times 3 = 7 + 7 \times 2 = 7 + 7 + 7 \times 1 = 7 + 7 + 7 + 0 = 21$$

Podemos ver que la multiplicación de dos números a, b se puede transformar en otro problema más pequeño multiplicar **a** por **(b-1)**, el caso base se produce cuando **b = 0** y el producto es **0**. Ejemplo:

```
/* producto.c */

#include <stdio.h>

int producto(int a, int b)
{
        if (b == 0) /* caso base */
                return 0; /* como b = 0, se retorna 0*/
        else
                return a + producto (a, b - 1); /* llamada a esta
misma función */
}

int main(void)
{
        /* en este caso se llama a la función y se imprime
directamente*/
        printf("%i ", producto( 7, 3));

        return 0;
}
```

Ejemplos de funciones:

Ejemplo:

Diseñe un programa que dados dos números enteros determine la suma y cual de ellos es mayor, usando dos funciones diferentes.

```
#include <stdio.h>
#include <conio.h>
void suma (int a, int b);   /*Declaraci¢n de la funci¢n*/
void mayor (int a, int b);  /*Tipo de dato, nombre de la funci¢n y el tipo y nombre de los
argumentos*/
main()
  {
     int a, b;
     printf("Ingrese el valor de a:\n");
     scanf("%d", &a);
     printf("Ingrese el valor de b:\n");
     scanf("%d", &b);
     suma(a,b);  /*Llamado de la funci¢n*/
     mayor(a,b); /*Unicamente el nombre de la funci¢n y de los par metros*/
     getch();
     return 0;
  }
  void suma(int a, int b) /*Definici¢n de la funci¢n*/
  {                  /*Abrimos llaves al inicio de la definici¢n*/
    int sum;           /*Declaraci¢n de las variables locales*/
    sum=a+b;
    printf("El valor de la suma es %d:\n\n", sum);
  }                 /*Fin de la funci¢n suma*/
  void mayor(int a, int b)
  {
    if(a==b)
    printf("Son iguales\n\n");
    else
    {
          if(a>b)
            printf("El valor de a es mayor que el de b\n\n");
          else
            printf("El valor de b es mayor que el de a\n\n");
    }
  }
```

Ejemplo:

Diseñe un Programa en C, que Dado un número entero y mayor que cero, Determine si es o no un número Primo. Ojo, los números primos sólo son divisibles por el mismo y por la unidad (1).

```
#include <stdio.h>
#include <conio.h>
void primo (int numero);
main()
 {
    int numero, ban=1;
    clrscr();
    while(ban==1)
    {
    printf("Introduzca el n£mero por favor:\n");
    scanf("%d", &numero);
    while(numero<0)
      {
```

```
            printf("ERROR, el valor del n£mero debe ser mayor que cero\n");
            scanf("%d", &numero);

      }

          primo(numero);

    printf("¨Otro n£mero (si=1 y No=0)?\n");
    scanf("%d", &ban);
    }
    getch();
    return 0;
 }
 void primo (int numero)
 {
   int div, primo=1;
   for(div=2; div<numero; div++)
   {
      if(numero%div==0)
      {
           primo=0;
           printf("%d NO es primo\n\n\n", numero);
           return 0;
      }
      else
           primo=1;
   }
    if(primo!=0)
       printf("%d es primo\n\n\n", numero);
 }
```

Ejemplo:

Diseñe un programa, que dada una cifra entera y mayor que cero, sea elevada a una potencia introducida por el usuario, la cual. (Ejemplo: 5^2=25).

```
#include <stdio.h>
#include <conio.h>
long int potencia (int base, int exponente);
main()
 {
    int base, exponente;
    clrscr();
    printf("La Base es:\n");
    scanf("%d", &base);
    while (base<0)
                                                {
                                printf("ERROR, el dato debe ser mayor que cero:\n");
                                            scanf("%d", &base);
                                                }
    printf("El Exponente es:\n");
    scanf("%d", &exponente);
    printf("%d ^ %d es %ld\n\n", base, exponente, potencia(base,exponente));
    getch();
    return 0;
 }
 long int potencia (int base, int exponente)
 {
    long int sum=0, i,x;
    for(i=1; i<exponente; i++)
                                                {
```

```
                                        x=base*base;
                                        sum=sum+x;
                                            }
    return (sum);
}
```