I.E.S. EL MAJUELO	
Boletín 8 ***Programación Modular***	***CURSO ACADÉMICO 2022-2023*** ***NIVEL C.F.G.S. D.A.M*** ***CURSO 1º (MÓDULO PROGRAMACIÓN)*** ***DEPARTAMENTO : Informática***

Introducción

Una función es una colección de sentencias que ejecutan una tarea específica . En un programa C se distinguen dos clases de funciones, las funciones definidas por el usuario y las funciones de biblioteca.
En C, la definición de una función nunca puede contener a la definición de otra función

Paso de argumentos a las funciones

Cuando se llama a una función , el valor del primer parámetro actual es pasado al primer parámetro formal, el valor del segundo parámetro actual es pasado al segundo parámetro formal y así sucesivamente. Por defecto, todos los argumentos, excepto los arrays, son pasados por valor . Esto significa que a la función se pasa una copia del valor del argumento, no su dirección. Esto hace que la función invocada, no pueda alterar las variables que contienen los valores pasados.

Ejemplo.

```
#include <stdio.h>
void intercambio (int , int);
void main ()
{
      int a=20 , b=30;
      intercambio (a,b); // a y b son pasado por valor
      printf(" En la funcion main a es %d y b es %d \n " , a , b);
}
void intercambio (int x , int y)
{
      int z =x;
      x=y;
      y=z;
       printf("En la funcion intercambio a es %d y b es %d \n " , x , y);

}
```

En este ejemplo, observamos que la función main llama a la función intercambio y le pasa los argumentos a y b. La función intercambio almacena en x el valor de a y en y el valor de b. Esto significa que los datos a y b se han duplicado.

Los parámetros formales x e y son variables locales a la función intercambio, por lo tanto sólo son accesibles dentro de la propia función. Esto significa que las variables locales se crean cuando se ejecuta la función y se destruyen cuando finaliza la ejecución de la función.

*Después de ejecutarse la función intercambio el valor de **y** ha sido copiado en **x** y el valor de **x** ha sido copiado en **y.** Lo importante es darse cuenta que estas operaciones no afectan a los valores de a y de b.*

*Si lo que se desea es alterar los contenidos de los argumentos especificados en la llamada , entonces hay que pasar dicho argumento por **referencia**. Esto es, **a la función se le pasa la dirección de cada argumento** y no su valor, lo que exige que los parámetros formales correspondientes sean punteros. Para pasar la dirección de un argumento, utilizamos el operador &.*

```
#include <stdio.h>
#include <stdlib.h>
void intercambio(int * , int *);
void main()
{
        int a=20 , b=30;
        intercambio (&a , &b); // a y b son pasado por referencia
        printf("El valor dentro del main a es %d y b es %d \n " , a , b);
}

void intercambio (int *x , int *y)
{
        int z =*x;
        *x=*y;
        *y=z;
      printf("El valor dentro de la funcion intercambio a es %d y b es %d \n " , *x , *y);
}
```

Programa C formado por múltiples ficheros

Un programa C es un conjunto de funciones que se llaman entre sí. Lo que no debemos pensar es que todo el programa tiene que estar escrito en un único fichero .c. De hecho no es así, ya que además del fichero .c, intervienen uno o más ficheros de cabecera.

Nosotros podemos hacer lo mismo; esto es, podemos optar por escribir las funciones que nos interesen en uno o más ficheros separados y utilizar para las declaraciones y/o definiciones uno o más ficheros de cabecera.

Un fichero fuente puede contener cualquier combinación de directrices para el compilador, declaraciones y definiciones. Pero, una función o una estructura, no puede ser dividida entre dos ficheros fuente. Por otra parte, un fichero fuente no necesita contener sentencias ejecutables; esto es, un fichero fuente puede estar formado, por ejemplo, solamente por definiciones de variables que son referenciadas desde otros ficheros fuentes.

Ejemplo

Vamos a escribir un programa C que nos dé como resultado el mayor de tres valores dados. Para ello, escribiremos una función max que devuelva el mayor de dos valores pasados como argumentos en la llamada. Esta función será invocada dos veces por la función main; la primera para calcular el mayor de los dos primeros valores, y la segunda para calcular el mayor del resultado anterior y del tercer valor. Los tres valores serán introducidos por el teclado. El código correspondiente lo escribiremos en dos ficheros:

- El fichero modulo01.c contendrá la función main, además de otras declaraciones.
- El fichero modulo02.c contendrá la función max.

```
/************************ modulo01.c ***************************
          Fichero fuente 1 - función principal
*************************************************************/
```

```
#include <stdio.h>

/* Declaración de funciones */
int max(int x, int y);

void main () /* función principal */
{
  int a = 0, b = 0, c = 0; /* definición de variables */
  int mayor = 0;

  printf("Valores a, b y c: ");
  scanf("%d %d %d", &a, &b, &c);

  mayor = max(a, b);     /* mayor de a y b */
  mayor = max(mayor, c); /* mayor del resultado anterior y de c */
  printf("%d\n", mayor);
}
```

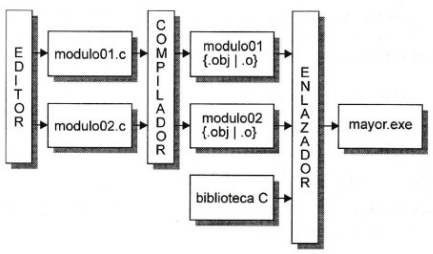
```
/************************ modulo02.c ***************************
            Fichero fuente 2 - función max
****************************************************************/
/* Función max. Toma dos valores, x e y, y devuelve el mayor */
int max(int x, int y)
{
  int z = 0;
  z = (x > y) ? x : y;
  return z;
}
```

Para compilar un programa formado por varios ficheros, se deben compilar por separado cada uno de los ficheros y, a continuación, enlazarlos para formar un único fichero ejecutable.
Por ejemplo, para compilar y enlazar los ficheros modulo0l.c y modulo02.c, utilizaremos alguna de las órdenes siguientes:

```
c1 modulo01.c  modulo02.c / mayor.exe  (en MS-DOS Microsoft C)

cc modulo01.c  modulo02.c -o mayor.exe  (en linux)
```

El resultado es el programa ejecutable mayor.exe. La siguiente figura muestra paso a paso la forma de obtener el fichero ejecutable.



I.E.S. EL MAJUELO	
Boletín 8 ***Programación Modular***	***CURSO ACADÉMICO 2022-2023*** ***NIVEL C.F.G.S. D.A.M*** ***CURSO 1º (MÓDULO PROGRAMACIÓN)*** ***DEPARTAMENTO : Informática***

Accesibilidad de variables

Se denomina ámbito de una variable a la parte de un programa donde dicha variable puede ser referenciada por su nombre . ***Una variable puede ser limitada a un bloque, a un fichero, a una función, o a una declaración de una función.***

Variables globales y locales

Cuando una variable se declara fuera de todo bloque en un programa, es accesible desde su punto de definición o declaración hasta el final del fichero fuente. Esta variable recibe el calificativo de global.

Una variable global existe y tiene valor desde el principio hasta el final de la ejecución del programa. ***Las funciones tienen todas carácter global.***

Si la declaración de una variable se hace dentro de un bloque, el acceso a dicha variable queda limitado a ese bloque y a los bloques contenidos dentro de éste por debajo de su punto de declaración. En este caso, la variable recibe el calificativo de local o automática.

Una variable local existe y tiene valor desde su punto de declaración hasta el final del bloque donde está definida. Cada vez que se ejecuta el bloque que la contiene. la variable local es nuevamente definida, y cuando finaliza la ejecución del mismo, 1a variable local deja de existir. Un elemento con carácter local es accesible solamente dentro del bloque al que pertenece.

E1 siguiente ejemplo muestra el ámbito de las variables, dependiendo de si están definidas en un bloque o fuera de todo bloque. En este ejemplo, al tratar de definir el ámbito de una variable distinguimos cuatro niveles:

- **E1 nivel externo (fuera de todo bloque).**
  Las variables definidas en este nivel, son accesibles desde el punto de definición hasta el final de1 programa.
- **El nivel del bloque de ia función main.**
  Las variables definidas en este nivel, solamente son accesibles desde la propia función main y, por lo tanto, son accesibles en los bloques 1 y 2.
- **El nivel del bloque 1 (sentencia compuesta).**
  Las variables definidas en este nivel, solamente son accesibles en el interior del bloque 1 y, por 1o tanto, en el bloque 2.
- **El nivel del bloque 2 (sentencia compuesta).**
  Las variables definidas en este nivel, solamente son accesibles en el interior del bloque 2.

```
/* vars01.c - Variables globales y locales
 */
#include <stdio.h>

/* Definición de var1 como variable GLOBAL */
int var1 = 50;

void main()
{ /* COMIENZO DE main Y DEL PROGRAMA */

  printf("%d\n", var1); /* se escribe 50 */

  { /* COMIENZO DEL BLOQUE 1 */
    /* Definición de var1 y var2 como variables
       LOCALES en el BLOQUE 1 y en el BLOQUE 2 */

    int var1 = 100, var2 = 200;

    printf("%d %d\n", var1, var2);
    /* escribe 100 y 200 */

    { /* COMIENZO DEL BLOQUE 2 */
      /* Redefinición de la variable LOCAL var1 */
      int var1 = 0;

      printf("%d %d\n", var1, var2);
      /* escribe 0 y 200 */

    } /* FINAL DEL BLOQUE 2 */

    printf("%d\n", var1); /* se escribe 100 */

  } /* FINAL DEL BLOQUE 1 */

  printf("%d\n", var1); /* se escribe 50 */

} /* FINAL DE main Y DEL PROGRAMA */
```

En el ejemplo anterior se observa que una variable global y otra local pueden tener el mismo nombre, pero no guardan relación una con otra, lo cual da lugar a que la variable global quede anulada en el ámbito de accesibilidad de la local del mismo nombre. Como ejemplo observar lo que ocurre en el programa anterior con var1.

Los parámetros declarados en la lista de parámetros de la declaración de una función o prototipo de la función, tienen un ámbito restringido a la propia declaración de la función.

int max(int x, int y); /* declaración de 1a función max *,/

Los parámetros x e y en la declaración de la función max están restringidos a la propia declaración. Por esta razón se pueden omitir; esto es, la siguiente línea sería equivalente a la anterior:

int max(int, int); /* decfaración de la función max */

Los **parámetros formales** declarados en la lista de parámetros de la definición de una función, son locales a la función.

```
int max(int x, int y)   /* definición de 1a función max */
{
        int z=0;
        z = (x > y) ? x
        return z;
}
```

Los parámetros x e y en La definición de la función max son locales a la función; esto es, x e y se crean cuando se llama a la función para su ejecución y dejan de existir cuando finaliza la ejecución de la función.

Esta es la razón por la que x e y sólo son accesibles dentro de la propia función.

Clase de almacenamiento

Por defecto, todas las variables llevan asociada una clase de almacenamiento que determina su accesibilidad y existencia. Los conceptos de accesibilidad y de existencia tanto para las variables como para las funciones, pueden alterarse por los calificadores:

auto	almacenamiento automático
register	almacenamiento en un registro
static	almacenamiento estático
extern	almacenamiento externo

Los calificadores **auto** o **register** pueden ser utilizados solamente con variables locales; el calificador **extern** puede ser utilizado solamente con variables globales o funciones; y el calificador **static** puede ser utilizado con variables locales, globales o funciones.

Variables declaradas a nivel externo

En una variable declarada a nivel externo, esto es, fuera de toda definición de función, se pueden utilizar los calificadores **static** o **extern,** o bien omitir el calificador, en cuyo caso es como si se hubiera especificado **extern**. A nivel externo no se pueden utilizar los calificadores **auto** o **register**.

Una variable declarada a nivel externo es una definición de la variable o una referencia a una variable definida en otra parte. Esto quiere decir que la declaración de una variable externa inicializa la variable a cero (valor por defecto) o a un valor especificado.

A una variable definida a nivel externo, se la puede hacer accesible antes de su definición o en otro fichero fuente, utilizando el calificador **extern**. *Esto quiere decir, que la utilización del calificador **extern** tiene sentido cuando la variable ha sido definida a nivel externo, una vez, y solamente una, en cualquier parte del programa y queremos tener acceso a ella en otra parte donde no es visible.*

El siguiente ejemplo, formado por dos ficheros fuente, muestra lo expuesto con claridad. El fichero fuente UNO.C define la variable var y le asigna el valor 5.

Así mismo, para hacer visible var antes de su definición utiliza la declaración extern int var.

```
/*************************************************************
Fichero fuente UNO.C
* * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * */
#include <stdio.h>
void funcion_1 ( ) ;
void funcion_2( );

extern int var ; * declaración  de var referencia a la variable var definida a continuación *

void main (){
        var=var+1;
        printf ("%d \n", var); // se escribe 6
        funcion_1();
}

int var = 5; //definición de var.

void funcion_1 ( )
{
        var=var+1;
        printf ("%d \n", var); // se escribe 7
        funcion_2();
}
```

El fichero fuente DOS.c utiliza la declaración **extern int var** para poder acceder a la variable var definida en el fichero fuente UNO.C.

```
/*************************************************************
                Fichero fuente   DOS.C
*************************************************************/

#include < stdio.h>

extern int var;  * declaracion de var . Referncia a la variable var definida en el fichero UNO.c*/

void funcion_2 ( )
{
        var=var+1;
        printf ("%d \n", var); // se escribe 7
}
```

observar que en el programa anterior formado por los ficheros fuente UNO.c y DOS.C:

1. Existen tres declaraciones externas de var.
2. La variable var, se define e inicializa a nivel externo una sola vez.
3. La declaración **extern** en el fichero UNO.C, permite acceder a la variable var, antes de su definición. Sin la declaración extern, la variable global var no sería accesible en la función main.
4. La declaración extern en el fichero DOS.C, permite acceder a la variable var en este fichero.

5. Si la variable var no hubiera sido inicializada explícitamente, C le asignaría automáticamente el valor 0 por ser global.

Si se utiliza el calificador static en la declaración de una variable a nivel externo. ésta solamente es accesible dentro de su propio fichero fuente. Esto permite declarar otras variables static con el mismo nombre en otros ficheros correspondientes al mismo programa.

Como ejemplo, sustituimos en los ficheros UNO.c y DOS.c del programa anterior. el calificador extern por static. Si ahora ejecutamos el programa observamos que la solución es 6, 7 y 1 en lugar de 6,7 y 8,lo que demuestra que el calificador static restringe el acceso a la variable, al propio fichero fuente.

Variables declaradas a nivel interno.

En una variable declarada a nivel interno, esto es, dentro de un bloque, se pueden utilizar cualquiera de los cuatro calificadores. u omitir el calificador, el calificador se considera la variable como **auto** (local o automática).

Una variable declarada como **auto** solamente es visible dentro del bloque donde está definida. Este tipo de variables no son inicializadas automáticamente, *por lo que hay que inicializarlas explícitamente, cuando sea necesario.*

Una variable declarada a nivel interno como **static**, solamente es visible dentro del bloque donde está definida; pero, a diferencia de las automáticas, su existencia es permanente, en lugar de aparecer y desaparecer al iniciar y finalizar la ejecución del bloque que la contiene.

Una variable declarada **static** es inicializada solamente una vez, cuando comienza la ejecución del programa. No es reinicializada cada vez que se ejecuta el bloque que la contiene. Si la variable no es inicializada explícitamente, C la inicializa automáticamente a 0.

Una declaración **register** indica al compilador que la variable será almacenada, si es posible, en un registro de la máquina, lo que producirá programas más cortos y más rápidos. El número de registros utilizables para este tipo de variables, depende de la máquina. Si no es posible almacenar una variable register en un registro, se le da el tratamiento de automática. Este tipo de declaración es válido para variables de tipo **int** y de tipo **puntero**, debido al tamaño del registro.

Una variable declarada como **register** solamente es visible dentro del bloque donde está definida. Este tipo de variables no son inicializadas automáticamente, por lo que hay que inicializarlas explícitamente, si es necesario.

Una variable declarada **extern,** referencia a una variable definida con el mismo nombre a nivel externo en cualquier parte del programa. *La declaración **extern** a nivel interno es utilizada para hacer accesible una variable externa, en una función o módulo en el cual no lo es.*

Ejemplo

```
/* vars03.c - Variables declar:adas a nivel interno */
#include <stdio. h>

void funcion_1 ( );
```

```
void main ( )
{
        /* se hace referencia a la variable var1 */
        extern int var1;
        /* var2 es accesible solamente dentro de main. Su valor inicial es 0 */

        static int var2;

        /* var3 es almacenada en un registro si es posible */
        register int var3 - 0;

        /* var4 es declarada auto , por defecto */
        int var4=0;

        var1=var1+2;

        /* se escriben los valores 7 , 0 , 0 , 0 */
        printf(" %d  %d  %d  %d  \n" , var1 , var2, var3, var4);
        funcion_1 ( );
}

int var1=5;

void funcion_1 ()
{
        //Se define la variable local var1
        int var1 = 15;

        //var2 es accesible solamente dentro de funcion_1( )
        static var2=5;
        var2 =var2+5;
        // Se escriben los valores 15 , 10
        printf ("%d  %d\n", var1 , var2);
}
```

En este ejemplo, la variable **var1** está definida a nivel externo. En la función **main** se utiliza una declaración extern, para hacer accesible dentro de ésta, la variable **var1**. La variable **var2** declarada **static** es inicializada, por defecto, a 0.

En la función ***funcion_1 ()*** se define la variable local **var1**, anulando así a la variable externa **var1**.La variable **var2**, declarada **static**, es inicializada a 5. Esta definición no entra en conflicto con la variable **var2** de la función **main**, ya que las variables **static** a nivel interno son visibles solamente dentro del bloque donde están declaradas. A continuación la variable **var2** es incrementada en 5, de tal forma que si **funcion_1** fuera llamada otra vez, el valor inicial para esta variable sería de 10, ya que las variables internas declaradas **static**, conservan los valores adquiridos durante la última ejecución.

Declaración de funciones a nivel interno y a nivel externo

Una función declarada **static** es accesible solamente dentro del fichero fuente en el que está definida.

Una función declarada **extern** es accesible desde todos los ficheros fuentes que componen un programa.

La declaración de una función **static** o **extern**, puede hacerla en la función prototipo o en la definición de la función. *Una función es declarada por defecto **extern***. Por ejemplo, la siguiente declaración indica que funcion_1 va a ser static.

```
static void funcion_1( );
```

EJERCICIO 1

Realiza un programa que lea un número por teclado y calcule el factorial de ese número. Debe de contener la función main , la función factorial y la función leer número.

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


// PROTOTIPOS
int factorial (int numero);
int leer_numero();

/* ******************************************
************* PROGRAMA PRINCIPAL ***************
****************************************** */
void main()
{
	int m, n;

	m = leer_numero();


	printf("\n\nEl factorial del número : %d es  %d\n\n",m , factorial (m));
}


int leer_numero()
{
	int m;

	do{
		printf("Introduzca un numero positivo para calcular su factorial ");
		scanf("%d", &m);
		fflush(stdin);
	}while(m <= 0);

	return m;
```

```
}

int factorial (int numero)
{
        int factorial = 1;
        int contador;

        for (contador = 1; contador <= numero; contador++)
                factorial = factorial * contador;

        return factorial;
}
```

EJERCICIO 2

Realizar un programa que calcule el total de combinaciones (sin repetición) de m elementos tomados de n en n.

C(m,n)=(factorial(m)) / (factorial(n) * factorial(m-n));

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

// PROTOTIPOS

int combinaciones (int m, int n);
int leer_total_elementos();
int factorial(int numero);

/* ******************************************
********** FUNCIONES ********************
****************************************** */

int combinaciones (int m, int n)
{
   int z=factorial(m)/(factorial(n) * factorial(m-n));
   return z;
}


int leer_total_elementos()
{
        int m;

        do{
                printf("Introduzca el numero total de elementos: ");
                scanf("%d", &m);
```

```
		fflush(stdin);
	}while(m <= 0);

	return m;
}
int factorial (int numero)
{
	int factorial = 1;
	int contador;

	for (contador = 1; contador <= numero; contador++)
		factorial = factorial * contador;

	return factorial;
}

/* ******************************************
************* PROGRAMA PRINCIPAL **************
****************************************** */
void main()
{
	int m, n;

	printf("Introducir el valor de m");
	m = leer_total_elementos();

	printf("Introducir el valor de n");
	n = leer_total_elementos();

	printf("\n\nNumero total de combinaciones: %d\n\n",combinaciones(m, n));
}
```

EJERCICIO 3

Utilizar esa función en un programa que pida al usuario el precio de un producto y el porcentaje de descuento que se le va a aplicar, y que imprima por pantalla el precio resultante tras aplicarle dicho descuento. Este proceso deberá repetirse mientras el usuario no indique lo contrario.

Las funciones que debe de contener son las siguientes:

```
// PROTOTIPOS
float calcular_porcentaje (int numero, int porcentaje);
int leer_precio();
int leer_porcentaje();
```

I.E.S. EL MAJUELO	
Boletín 8 ***Programación Modular***	***CURSO ACADÉMICO 2022-2023*** ***NIVEL C.F.G.S. D.A.M*** ***CURSO 1º (MÓDULO PROGRAMACIÓN)*** ***DEPARTAMENTO : Informática***

EJERCICIO 4

Realizar una función que reciba un número entero positivo y devuelva si dicho número es primo o no. Utilizar dicha función en un programa que intente imprimir por pantalla los números primos que existen dentro de un rango escogido por el usuario.

EJERCICIO 5

1) Realizar una función que dado n devuelva el valor de f(n):

$$f(n)=(1! + 2! + 3! + \ldots.. n!) / n$$

EJERCICIO 6

Realizar un programa que lea una cantidad de números a introducir. Para cada uno de los números introducidos debe indicar si el número es o no perfecto (un nº es perfecto cuando es igual a la suma de sus divisores excepto el mismo).

Funciones que debe de contener:

```
// PROTOTIPOS
char leer_opcion ();
int leer_numero ();
int es_divisor (int, int);
int es_perfecto (int);
void mostrar_resultado (int);
```

EJERCICIO 7

Realizar un programa que presente el siguiente menú:

1) Suma de números naturales
2) Factorial
3) Suma de los inversos
4) Fin

Después de pulsar una opción deberá pedir un número N.
Si la opción es la 1 se calculará la suma 1+2+3+...N.
Si la opción es la 2 se calculará 1*2*3*...N.
Si la opción seleccionada es la 3 se calculará 1/1 + 1/2 + 1/3 +...+ 1/N.

```
// PROTOTIPOS
int leer_opcion();
void mostrar_menu();
void tratar_opcion (int op);
float suma_naturales();
float suma_inversos();
```

```
float producto_naturales();
int leer_numero();
void mostrar_resultado (int op, float res);
```

EJERCICIO 8

Se desea diseñar un programa que obtenga el salario neto semanal de n trabajadores de acuerdo a los siguientes puntos:

- Las 38 primeras horas semanales se cobran a la tarifa ordinaria.
- Cualquier hora extra a 1,5 veces la tarifa ordinaria.
- Los 200 primeros euros están libres de impuestos; los siguientes 300 tienen un impuesto del 25 % y los restantes del 45 %.
- La tarifa ordinaria es de 12 euros /hora.

El programa deberá pedir el código del empleado y el número de horas que ha trabajado, y a continuación deberá mostrar de forma detallada el salario neto semanal.

El usuario decidirá tras el cálculo para un trabajador si desea continuar con el programa o si desea salir del mismo.

```
//PROTOTIPOS
float calcula_impuestos (float);
float calcula_salario_horas (int);
float calcular_salario (int,float *);
void escribir_resultados (int,int,float,float);
void leer_horas (int *);
void leer_opcion (char *);
void leer_codigo (int *);
```

Ejemplo de salida

```
Introduzca el código del empleado: 99
Introduzca el número de horas: 42
Salario bruto (42 horas) = 528 euros
Impuestos: 87.60000 euros
Salario neto del empleado 99: 440.4 euros
¿Desea continuar con el siguiente empleado (s/n)?: N
```

```
//
**********************************************************************************
* //
// NOMBRE: calcula_impuestos
// DESCRIPCIÓN: Calcula la cantidad a retener al empleado a cuenta de impuestos.
// PARÁMETROS: sal ==> Salario bruto del empleado (E).
```

```
// DEVUELVE: Cantidad a retener.
//
**********************************************************************************
* //

//
**********************************************************************************
* //
// NOMBRE: calcula_salario_horas
// DESCRIPCIÓN: Calcula el salario semanal del empleado correspondiente a las
//horas  que ha trabajado.
// PARÁMETROS: h ==> Número de horas (E).
// DEVUELVE: Salario bruto del empleado.
//
**********************************************************************************
* //


//
**********************************************************************************
* //
// NOMBRE: calcular_salario
// DESCRIPCIÓN: Calcula el salario semanal del empleado.
// PARAMETROS: h ==> Número de horas (E).
//             i ==> Retención de impuestos (E/S).
// DEVUELVE: Salario bruto del empleado.
//
**********************************************************************************
* //

//
**********************************************************************************
* //
// NOMBRE: escribir_resultados
// DESCRIPCIÓN: Imprime los resultados del programa.
// PARÁMETROS: cod   ==> Código del empleado (E).
//             horas ==> Número de horas (E).
//             sal   ==> Salario bruto (E).
//             imp   ==> Retención de impuestos (E).
// DEVUELVE: -
//
**********************************************************************************
* //

//
**********************************************************************************
* //
// NOMBRE: leer_horas
// DESCRIPCIÓN: Lee el n?mero de horas semanales que ha trabajado el empleado.
```

```
// PARÁMETROS: h ==> Número de horas (E/S).
// DEVUELVE: -
//
**********************************************************************************
* //

//
**********************************************************************************
* //
// NOMBRE: leer_opcion
// DESCRIPCIÓN: Lee si el usuario desea continuar con otro empleado o salir.
// PARÁMETROS: o ==> S si desea continuar, N si desea salir.
// DEVUELVE: -
//
**********************************************************************************
* //


//
**********************************************************************************
* //
// NOMBRE: leer_codigo
// DESCRIPCIÓN: Lee el código del empleado.
// PARÁMETROS: c ==> Códiigo del empleado (E/S).
// DEVUELVE: -
//
**********************************************************************************
* //
```

EJERCICIO 9

Realizar un programa que sume dos fracciones y de cómo resultado una fracción irreducible.

```
//PROTOTIPOS
void intercambia (int *, int *);
int maximo_comun_divisor (int, int);
int abs (int);
void calcular_fraccion_final (int,int,int *,int *);
void calcular_suma (int,int,int,int,int *,int *);
void leer_fraccion (int *n, int *d, int numero);
void imprimir_resultado (int,int,int,int);
```

Ejemplo de salida

Fracción 1
Introduzca el numerador: 6
Introduzca el denominador: 4
Fracción 2
Introduzca el numerador: 7

Introduzca el denominador: 5
La suma de ambas fracciones vale 58 / 20
La suma simplificada es de 29 / 10

EJERCICIO 10

Lee del usuario una fecha en formato dd/mm/aaaa e imprime si es correcta o no.

VARIABLES: "dia", "mes", "anio" ==> Componentes de la fecha leída del usuario.
"fecha_correcta" ==> Flag utilizado para almacenar si la fecha es correcta o no.

```
//PROTOTIPOS
int bisiesto (int);
int dias_mes (int,int);
int comprueba_dia (int,int,int);
int comprueba_mes (int);
int comprueba_anio (int);
int comprueba_fecha (int,int,int);
void leer_fecha (int *,int *,int *);
void escribir_resultado (int);

// ********************************************************************************
//
// NOMBRE: bisiesto
// DESCRIPCIÓN: Recibe un año verifica si es bisiesto.
// PARÁMETROS: anio ==> Año (E).
// DEVUELVE: 1 si es bisiesto, 0 si no lo es.
// ********************************************************************************
//

// ********************************************************************************
//
// NOMBRE: dias_mes
// DESCRIPCIÓN: Recibe un mes y un a?o y devuelve los días que tiene ese mes.
// PARÁMETROS: mes  ==> Mes (E).
//             anio ==> Año (E).
// DEVUELVE: Número de días del mes.
// ********************************************************************************
//

// ********************************************************************************
//
// NOMBRE: comprueba_dia
// DESCRIPCIÓN: Recibe un dia de un mes y de un año y verifica si es correcto.
// PARÁMETROS: d  ==> Día (E).
//             m  ==> Mes (E).
//             a  ==> Año (E).
// DEVUELVE: 1 si es válido, 0 si no lo es.
```

```
// ********************************************************************************
//

// ********************************************************************************
//
// NOMBRE: comprueba_mes
// DESCRIPCIÓN: Recibe un mes y devuelve si es válido.
// PARÁMETROS: m  ==> Mes (E).
// DEVUELVE: 1 si es válido, 0 si no lo es.
// ********************************************************************************
//

// ********************************************************************************
//
// NOMBRE: comprueba_anio
// DESCRIPCIÓN: Recibe un año y devuelve si es válido.
// PARÁMETROS: a  ==> Año (E).
// DEVUELVE: 1 si es válido, 0 si no lo es.
// ********************************************************************************
//

// ********************************************************************************
//
// NOMBRE: comprueba_fecha
// DESCRIPCIÓN: Recibe una fecha y devuelve si es válida o no.
// PARÁMETROS: dd   ==> Día de la fecha (E).
//             mm   ==> Mes de la fecha (E).
//             aaaa ==> Año de la fecha (E).
// DEVUELVE: 1 si es válida, 0 si no lo es.
// ********************************************************************************
//

// ********************************************************************************
//
// NOMBRE: leer_fecha
// DESCRIPCIÓN: Lee una fecha en formato dd/mm/aaaa.
// PARÁMETROS: dd   ==> Día de la fecha (E/S).
//             mm   ==> Mes de la fecha (E/S).
//             aaaa ==> Año de la fecha (E/S).
// DEVUELVE: -
// ********************************************************************************
//

// ********************************************************************************
//
// NOMBRE: escribir_resultado
// DESCRIPCIÓN: Imprime por pantalla si la fecha es correcta o no.
// PARÁMETROS: valida ==> Fecha correcta o no, V o F respectivamente (E).
// DEVUELVE: -
```

I.E.S. EL MAJUELO	
Boletín 8 ***Programación Modular***	***CURSO ACADÉMICO 2022-2023*** ***NIVEL C.F.G.S. D.A.M*** ***CURSO 1º (MÓDULO PROGRAMACIÓN)*** ***DEPARTAMENTO : Informática***

// **********
//