

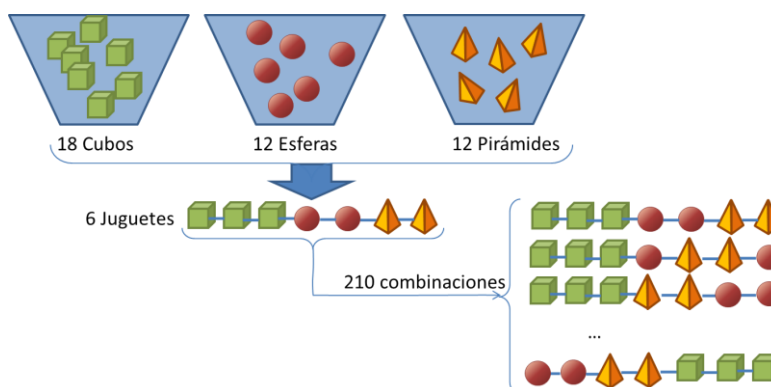


Guía de Prácticas

ASIGNATURA: Informática Industrial y Comunicaciones
CENTRO: Escuela Politécnica de Ingeniería de Gijón
ESTUDIOS: Grado en Ingeniería Electrónica y Automática
CURSO: 3º CUATRIMESTRE: 1
CARÁCTER: Obligatoria CRÉDITOS ECTS: 6
PROFESORADO: Ignacio Alvarez, José M^a Enguita, Borja Millán, Carlos Gómez-Aleixandre

PRACTICA 03: Algoritmos y funciones

- Se desea fabricar un juguete infantil con cubos, esferas y pirámides colocados en fila. Para ello, se dispone de 18 cubos, 12 esferas y 12 pirámides. Se quieren realizar el máximo número de juguetes iguales sin que sobre ninguna pieza. El fabricante quiere conocer cuántos juguetes podrá fabricar, de cuántas piezas cada uno, y cuántas combinaciones posibles puede realizar (con posiciones diferentes de cubos, esferas y pirámides).



Resultado esperado por pantalla:

```
Introduzca num cubos, esferas, pirámides: 18 12 12 [↵]
Se pueden hacer 6 juguetes iguales, cada uno con:
 3 cubos
 2 esferas
 2 pirámides

Cada juguete puede tener 210 combinaciones diferentes
```

- Para solucionar el problema, realizar planteamiento de programa completo: pide por teclado 3 enteros (número total de cubos, esferas, pirámides), a continuación calcula el máximo común divisor (m.c.d.=6 en este caso) para saber el n° máximo de juguetes iguales sin que sobre ninguna pieza, y con ello sabe el n° de cubos_por_juguete ($cpj=18/6=3$), esferas_por_juguete ($epj=12/6=2$), pirámides_por_juguete ($ppj=12/6=2$). Las combinaciones posibles son las permutaciones con repetición:

$$PR_{npj}^{cpj,epj,ppj} = \frac{npj!}{cpj! \cdot epj! \cdot ppj!}, \text{ donde } npj = cpj + epj + ppj$$

- Planteamiento de `main()`: variables básicas necesarias, y pseudo-código que no tiene en cuenta cómo se hacen los detalles (m.c.d., PR). Los detalles serán realizados mediante funciones, pero a `main()` le da igual cómo vayan a estar hechas internamente.
- Planteamiento down-top para programas “complejos”: ir haciendo función por función (plantear argumentos con sus tipos, salida con su tipo, crear vbles locales y desarrollar algoritmo, probar con un `main()` sencillo).

- Realizar función `Factorial(n)` y probar en un `main()` específico, pidiendo el valor de `n` por teclado y escribiendo el resultado (o, mejor, usando depurador).
- Realizar función `PermutacionesRepeticion(a,b,c)` y probar de la misma manera.
- Realizar función `mcd(a,b,c)` y probar de la misma manera, teniendo en cuenta que:

$mcd(a,b,c) = mcd(mcd(a,b),c) \rightarrow$ Hacer y probar primero `mcd(x,y)`.

`mcd(x,y)` : Calcular primero *min*=mínimo de los dos valores `x,y`. A continuación:

Desde $i=min$ hasta 1:

Comprobar si i es divisor de x y también de y (usar operador %)

Si i no es divisor de ambos, se continúa en el bucle.

Al terminar, el bucle, el valor resultante de i es el `mcd`.

(No es un algoritmo óptimo, pero después se podría refinar, a la función llamadora le da igual cómo esté hecho).

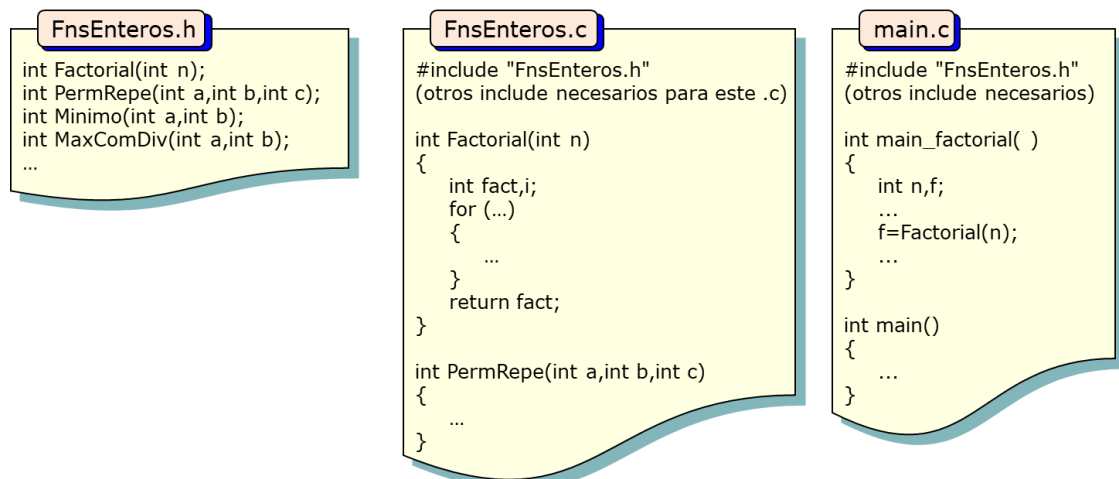
Ejemplo:

```
mcd(x=18,y=12) → result=12 → ¿es 12 divisor de x(18) Y TAMBIÉN de y(12) ? → NO → result=result-1
                result=11 → ¿es 11 divisor de x(18) Y TAMBIÉN de y(12) ? → NO → result=result-1
                result=10 → ¿es 10 divisor de x(18) Y TAMBIÉN de y(12) ? → NO → result=result-1
                ...
                result=6 → ¿es 6 divisor de x(18) Y TAMBIÉN de y(12) ? → SI → return result;
```

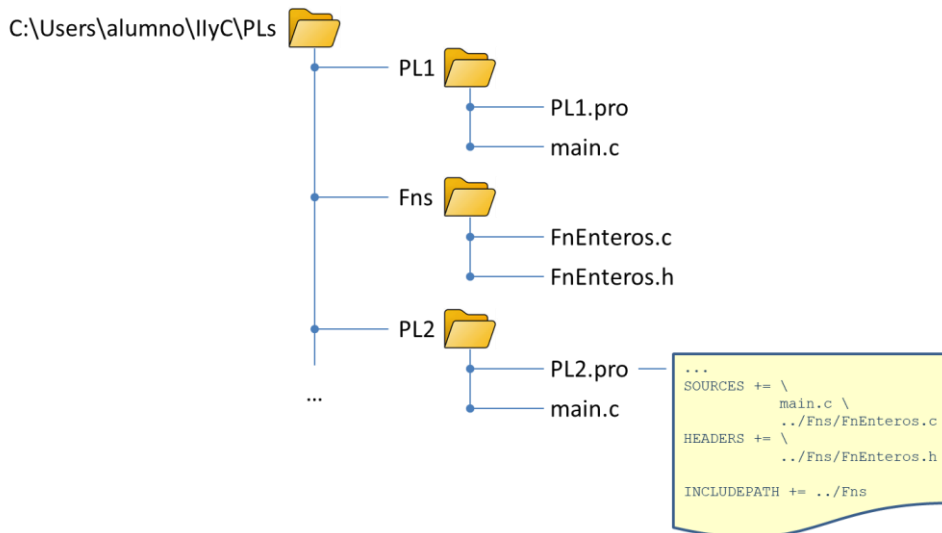
- Sólo cuando todas las funciones estén realizadas y probadas, se realiza el `main()` definitivo.

NOTA: Cuando se desean realizar varios programas distintos en un mismo proyecto, se sugiere la siguiente organización (ver ampliación de esta NOTA al final).

- Tener todas las funciones (excepto `main`) en un archivo `.c` separado (ejemplo `funciones.c`), que se añade al proyecto. Añadir un archivo de cabecera `.h` (`funciones.h`) con la declaración de prototipos de las funciones (cabecera de la función terminada en punto y coma, sin el código interno).
- Crear el `main()` de la primera prueba parcial en `main.c` o similar. `main.c` debe incluir “`funciones.h`” (entre “ ”, no entre < >) para poder utilizar las funciones. También `funciones.c` debe incluir a “`funciones.h`”
- Para pasar a una 2ª versión de prueba, sin eliminar la anterior, renombrar el `main()` (por ejemplo, `main_factorial()`) y hacer nuevo `main()` en `main.c`. `main_factorial()` está pero no se usa.



La organización más adecuada para compartir las mismas funciones entre varias PLs es la siguiente:



6. Depuración: diferencia entre Step Over (Paso a paso por procedimientos [F10]) y Step Into (Paso a paso por Instrucciones [F11]). Observar pila de llamadas y variables accesibles en función de la función seleccionada.

7. Ampliación propuesta: escribir el código necesario para imprimir en pantalla todas las permutaciones posibles. Algoritmo sugerido:

a) Asignar un dígito (1,2,3) a cada tipo de elemento (1=cubo, 2=esfera, 3=pirámide). Utilizar #define para facilitar la asignación de estas constantes.

b) Componer un n° entero con los dígitos siguientes: $\text{num} = \overbrace{33\dots3}^{ppj} \overbrace{22\dots2}^{epj} \overbrace{211\dots1}^{cpj}$
 Para ello, empezar con num=0, mult=1 y repetir cpj veces: num=num+1*mult, mult=mult*10
 A continuación, repetir epj veces: num=num+2*mult, mult=mult*10
 Por último, repetir ppj veces: num=num+3*mult, mult=mult*10

c) Desde i=1 hasta num, calcular el n° de dígitos de i que están a 1, a 2 y a 3. Si son iguales a ppj, epj, cpj, es una permutación válida y se escribe el valor de i.

d) Para calcular el n° de dígitos de un número entero iguales a uno dado: ir obteniendo el resto de dividir por 10 y comparando, incrementando una cuenta si es igual; el número se divide entre 10 y se repite hasta que el n° sea 0. Al acabar el bucle, la cuenta es el n° de dígitos iguales a ese.

Realizar la ampliación paso a paso:

- Desarrollar función `int CalcNumDigitos(int valor,int digito)` para apartado d) y probar con números cualquiera introducidos por teclado.
- Desarrollar función `int CrearNumConDigitos(int n_1s,int n_2s,int n_3s)` para apartado b) y probar de la misma forma.
- Desarrollar función `void EscribirCombinaciones(int n_1s,int n_2s,int n_3s)` y probar. Esta función llamará a las anteriores.
- Añadir funciones al programa escrito anteriormente.

Resultado esperado:

```

Introduzca num cubos, esferas, pirámides: 18 12 12 ↵
Info resultados: juguetes y combinaciones

Combinaciones: (1)=Cubo (2)=Esfera (3)=Pirámide
3322111
3321112
3321121
3321211
...
1112233

```