Práctica 5: Funciones. Paso por referencia. Recursividad.

Objetivos y descripción general.

- Aplicar el método de diseño descendente para obtener programas modulares.
- Adquirir destreza en el paso de parámetros a subprogramas.
- Reforzar el concepto de ámbito de las variables al aplicarlo en el diseño de subprogramas.
- Saber diferenciar entre los tipos de paso de parámetros (por valor y por referencia) y distinguir cuándo es conveniente uno u otro.
- Aprender a usar el paso por referencia como datos de entrada y/o datos de salida.
- Aprender a diseñar e implementar algoritmos recursivos.

Conceptos Básicos.

Paso de parámetros por valor y paso por referencia.

Cuando en la llamada a un subprograma pasamos un **parámetro por valor**, estamos **pasando una copia del valor del parámetro real de la llamada al parámetro formal de la definición de la función**. El subprograma reserva memoria para el parámetro formal y copia en este espacio el valor del parámetro real. Una vez acabado el subprograma se libera la memoria. Cualquier cambio en el valor del parámetro formal de **no influye** en el parámetro real.

En el **paso por referencia pasamos una referencia al parámetro real** (la dirección en memoria de la variable). De esta manera, se trabaja dentro del subprograma con la posición de memoria reservada para el parámetro real. Cualquier **modificación** del valor del parámetro formal afecta al valor del parámetro real pasado en la llamada.

Para distinguir un paso por valor de un paso por referencia, en C++ se coloca el **símbolo &** delante del nombre del parámetro formal en la cabecera de la función. También, se coloca en el prototipo. Viendo la llamada al subprograma no se puede diferenciar entre uno u otro. Por ejemplo:

`void paso_parametros(int param1, int & param2)`
`{`
`  param1 = 5;`
`  param2 = 7;`
`}`
	El parámetro param1 es pasado por valor y param2 por referencia.

`int main()`
`{`
`  int x, y;`
 	
`  x = 8;`
`  y = 9;`
`  paso_parametros(x, y);`
`  cout << “x = “ << x << “, y = “ << y << endl;`
 	
`  return 0;`
`}`
	La salida por pantalla es:
	x = 8, y = 7

Referencias en C++.

Además de las variables "normales", todas las que hemos visto hasta ahora, que almacenan valores, C++ soporta otro tipo de variables que son las variables referencia. Las referencias son variables que contienen una dirección de memoria.

Se usan como alias (una etiqueta) a una variable ya existente. Cuando se modifica el valor de la referencia se modifica el valor de la variable a la que hace referencia. En consecuencia, un parámetro pasado por referencia actúa como un alias del parámetro real de la llamada, y de esta forma en el cuerpo de instrucciones de la función se puede modificar su valor. Las referencias se declaran e inicializan a la vez de la forma:

```
Tipo_Dato & nombre_ref = nombre_var;
```

Se lee de derecha a izquierda, `nombre_ref` es una referencia a `Tipo_Dato`. Una vez inicializada, una referencia no puede cambiar para referenciar a otra variable. No existen referencias a `void` ni referencias a referencias. Las referencias en C++ son muy útiles porque permiten:

- Que las funciones devuelvan más de un resultado.
- Pasar objetos muy grandes a las funciones evitando el coste en tiempo y memoria que supone tener que hacer una copia.
- Sobrecargar operadores. Los operadores (y su sintaxis) para manipular tipos de datos definidos por el programador son los mismos que para los tipos predefinidos en lenguaje.

Funciones Recursivas.

Dentro del código de una función recursiva siempre hay una sentencia o expresión donde aparecerá la **llamada a la misma función**. Para no generar infinitas llamadas, y provocar un desbordamiento de la pila de llamadas a funciones, se necesita tener en cuenta que:

- Una función recursiva resuelve un problema de talla N, asumiendo resuelto el problema de talla N – 1. Así, los parámetros de la función deben variar su valor de una llamada a otra.
- Debe implementarse una condición de parada (caso base) de forma que no se produzcan más llamadas recursivas.

Recuerda:
Las funciones no leen datos de entrada desde teclado ni muestran resultados por pantalla. Las funciones tienen parámetros (de entrada y/o salida) y devuelven resultados. No tiene sentido usar dentro de las funciones los operadores de entrada y de salida de flujo: >> y <<.
Excepción:
La tarea a realizar por la propia función es la lectura de datos desde teclado o la impresión de resultados por pantalla.

BLOQUE DE EJERCICIOS

Ejercicio 1 (ejercicio1.cpp): Analiza el siguiente programa y modifícalo hasta que funcione correctamente. El programa calcula las dos soluciones de una ecuación de segundo grado.

```
#include <iostream>
#include <cmath>

using namespace std;

void Raices (float, float); //prototipo, modificar

int main()
{
  float c0, c1, c2, r1, r2; //coeficientes y raíces de la ecuación

  cout << "Introduce los coeficiente del polinomio de segundo grado: ";
  cin >> c0 >> c1 >> c2;

  Raices (r1, r2); //llamada, modificar

  cout << "Las raíces son: " << r1 << " y " << r2 << endl;

  return 0;
}

void Raices (float raiz1, float raiz2) //cabecera, modificar
{
   float discriminante;

   discriminante = sqrt ( b * b – 4.0 * a * c );
   raiz1 = ( -b + discriminante ) / ( 2.0 * a );
   raiz2 = ( -b - discriminante ) / ( 2.0 * a );
}
```

Ejercicio 2 (ejercicio2.cpp): Abre el fichero ejercicio2.cpp. Compílalo y ejecútalo. ¿Por qué en la línea 18 la variable y contiene el valor 500 en vez de 50? Añade puntos de ruptura en las líneas 19, 24 y 36. Usa los comandos `print var_name`, `info locals`, `info args`, `print &var_name` (`imprime la dirección de var_name`) para inspeccionar el contenido de las variables y sus direcciones en memoria.

Rellena la tabla siguiente con la información que muestra el depurador en la ventana de comandos y justifica e interpreta los resultados.

	info locals	info args	print &a	print &y
Línea 19				
Línea 36				
Línea 24				

Ejercicio 3 (ejercicio3.cpp): Escribe una función, **ConvertirTiempo,** que reciba como entrada una cantidad de tiempo en segundos y la devuelva descompuesta en horas, minutos y segundos. **Justifica** para cada uno los parámetros de la función el tipo de paso de parámetros elegido e indica si se trata de un parámetro de entrada y/o salida.

```
void ConvertirTiempo (unsigned int, unsigned int &, unsigned int &, unsigned int &);
```

Escribe un programa en C++ que pida al usuario una cantidad de tiempo en segundos y, con ayuda de la anterior función, calcule y muestre por pantalla el equivalente en formato h:m:s. Ejemplo:

```
Introduce la cantidad de tiempo (s): 2115
2115s = 0h:35m:15s
```

Ejercicio 4 (ejercicio4.cpp): Implementa un procedimiento **recursivo**, **ImprimirBinario,** que se le pase como dato de entrada un número entero en base decimal e imprima por pantalla su codificación en binario. **Justifica** para cada uno los parámetros del procedimiento el tipo de paso de parámetros elegido e indica si se trata de un parámetro de entrada y/o salida.

Implementa un programa en C++ que lea un número entero desde el teclado y llame al procedimiento para mostrar su equivalente en binario. El proceso se debe repetir mientras el usuario desee. El algoritmo para convertir un número decimal a binario es el siguiente:

```
ALGORITMO DecimalAbinario
Datos Entrada: num, numero entero en codificación decimal
Datos de Salida: número en codificación binaria
INICIO
SI num >= 2 ENTONCES
   DecimalABinario(num/2)
   Escribir(num%2)
SINO
   Escribir (num)
FINSI
FIN
```

Por ejemplo, 23 corresponde con 10111.

n	n%2	n/2
23	1	11
11	1	5
5	1	2
2	0	1
1	1	0

Ejercicio 5 (ejercicio5.cpp): Implementa un procedimiento recursivo, **MostrarSec**, que reciba como parámetro de entrada un número entero n, y muestre por pantalla una secuencia formada por los números incluidos en el intervalo `[1, n]`, impresos de forma que aparezcan primero los números pares en orden decreciente, seguidos de los impares en orden creciente. **Justifica** para cada uno los parámetros del procedimiento el tipo de paso de parámetros elegido e indica si se trata de un parámetro de entrada y/o salida.

Por ejemplo, si n = 10, se muestra por pantalla la secuencia:
`10 8 6 4 2 1 3 5 7 9`
Escribe un programa en C++ que pida al usuario dos valores enteros y muestre por pantalla la secuencia del primer número, del segundo y de la suma.

Ayuda: Cuestión 4, documento de prerrequisitos.

IMPORTANTE:

- Todos los programas deben seguir la Guía de Estilo e incluir los comandos de Doxygen para generar la documentación.
- Subir al Aula Virtual los archivos .cpp de uno en uno (enunciado y adicionales). Sólo los sube un miembro de la pareja.
- Fecha de entrega: durante la sesión de vuestro grupo de lab.