Sesión 5. Recursividad

Competencias previas

- Crear, importar, exportar, compilar y ejecutar un proyecto en Eclipse.
- Escribir pequeños programas en C++ con variables de tipos simples, operadores e instrucciones de lectura, escritura, asignación, alternativa (if) y bucles (while).
- Definir y probar módulos simples con parámetros de entrada y un valor devuelto.
- Poder incluir en programas funciones de las librerías predefinidas de C++.

Objetivos

- Usar las herramientas básicas de depuración de Eclipse
- Implementación de módulos usando la recursividad.

Actividad 1: Depuración de módulos recursivos

En la sesión anterior ya hemos utilizado el depurador para facilitar la detección de errores y ver cómo evolucionaba el valor de las variables al ejecutar nuestros programas línea a línea. En esta actividad aprenderemos a definir puntos de parada (*breakpoints*) para forzar al depurador a detenerse al llegar a ellos. También veremos cómo, al ejecutar un módulo recursivo en modo depuración, las sucesivas llamadas al propio módulo se van apilando y cómo se van devolviendo los resultados al ir terminando (cuando se alcanza el caso base).

- Importa el proyecto **combinatorio.tar.gz**, incluido en la carpeta de la sesión.
- Añade un punto de parada en la segunda línea del módulo **main** (muestra el resultado de factorial (4) por pantalla). Para ello, selecciona la línea y pulsa en el menú la opción: *Run → Toggle Break Point*, o pulsa la combinación de teclas: *Shift + Ctrl + B*. Verás que aparece un punto azul a la izquierda del número de la línea seleccionada.
- Ejecuta el programa en modo depuración pulsando la opción: *Run → Debug (F11)*. La ejecución arrancará en la primera línea del módulo main (llamada al módulo de pruebas).
- Pulsa la opción *Run → Resume (F8)* para ejecutar el código sin parar hasta alcanzar el *breakpoint* previamente definido o, si no hay ninguno, hasta terminar la ejecución. Como hemos definido un breakpoint en la llamada al módulo factorial(4), se detendrá en ella.
- Para entrar en el módulo recursivo y ejecutarlo paso a paso, pulsa la opción *Run → Step Into (F5).* Continúa pulsando esta opción y comprueba cómo, en cada nueva llamada recursiva que realiza, el módulo se va añadiendo a la lista de llamadas pendientes de atender en la ventana *Debug*. Observe cómo evolucionan los valores de las variables en cada llamada y cómo, al llegar al caso base, se van devolviendo los resultados calculados hacia atrás y las llamadas resueltas van desapareciendo de la lista de pendientes.

IMPORTANTE: Para todos los módulos desarrollados como parte de los siguientes ejercicios, recuerda:
(1) definir su especificación, incluyendo la pre- y post- condición;
(2) diseñar al menos 3 casos de prueba e implementar el módulo de prueba correspondiente;
(3) definir las variables requeridas e incluir sus roles y el de los parámetros;
(4) implementar el módulo correspondiente;
(5) ejecutar el módulo de prueba para verificar que su implementación sea correcta; en caso de que no lo sea, modifícalo hasta que todas las pruebas pasen con éxito; y,
(6) después de completar la implementación (solo para módulos iterativos), también debes incluir el tamaño del problema y la complejidad como parte de la especificación del módulo.

Actividad 2. Persistencia (Noviembre 2012)

El matemático Neil Sloane estudió la **persistencia** de los números enteros.

Dado un número entero no negativo ***n1***, se multiplican todas sus cifras para obtener un nuevo número ***n2***. Con ese valor ***n2*** se vuelve a hacer lo mismo hasta conseguir un número de una cifra. El número de pasos necesarios para conseguir esto es la persistencia del número n.

La persistencia del 253 es 2: 253 → 30 (2*5*3) → 0 (3*0)

La persistencia del 88 es 3: 88 → 64 (8*8) → 24 (6*4) → 8 (2*4)

La persistencia del 5 es 0: 5

Se cree que ningún número tiene una persistencia superior a 11.

Importa el proyecto **persistencia.tar.gz**, incluido en la carpeta de la sesión, en el que encontrarás una versión iterativa de dos módulos: uno que calcula el producto de los dígitos de un número entero y otro que calcula la persistencia de un número entero. Se pide:

1. **Implementar un módulo recursivo que, dado un entero no negativo n, calcule el producto de todas sus cifras (versión recursiva del módulo `productoDigitos`).**

   Por ejemplo: si n vale 253, debería devolver 30
   si n vale 64, debería devolver 24.
   si n vale 3, debería devolver 3.

2. **Implementar un módulo recursivo que, dado un entero no negativo n, calcule la persistencia de ese número (versión recursiva del módulo `persistencia`).**

   Por ejemplo, si n vale 253, la persistencia es 2.
   si n vale 88, la persistencia es 3
   si n vale 5, la persistencia es 0.
   si n vale 22, la persistencia es 1.

3. **Incluir en cada módulo su especificación (pre- y post-condición).**

Actividad 3. Matrícula

Una escuela de idiomas tiene un sistema de enseñanza basado en niveles (del 1 al 6). Un estudiante hace una prueba de acceso y se le sitúa en su nivel inicial. Después, el alumno decide hasta qué nivel quiere llegar. El estudiante paga por los niveles de los que se quiere matricular, sabiendo que:

p1 = 200 euros (el precio del primer nivel)

pn = 0.9 * pn-1 (el precio del n-ésimo nivel es el 90% del nivel anterior)

Así, por ejemplo, si un alumno se matricula desde el 2º al 4º nivel, pagará:

p2 + p3 + p4 = 180 + 162 + 145.8 = 487.8 euros

Escribir un programa en C++ que permita realizar estos cálculos, implementando los siguientes módulos:

- precio (recursivo): dado un nivel, calcula el precio para ese nivel.
- matricula (iterativo y recursivo): dado el nivel inicial y el final, calcula el precio total de la matrícula de un alumno.

Ejercicios adicionales.

Invertir los dígitos de un número

Se pide implementar una versión recursiva y otra iterativa de un módulo que, dado un número entero positivo de dos dígitos o más, muestre por pantalla sus dígitos en orden inverso. Así, si el módulo recibe el número 12345, deberá mostrar por pantalla el número 54321. ¿Cómo probaría estos módulos?

Nota: observa que si vamos dividiendo sucesivamente el número de entrada por 10, los restos de estas divisiones van conformando el número solicitado.

Valor de entrada (n) = 12345

n = 12345 → 12345/ 10 = 1234; mostrar resto = 5

n = 1234 → 1234/ 10 = 123; mostrar resto = 4

n = 123 → 123/ 10 = 12; mostrar resto = 3

n = 12 → 12/10 = 1; mostrar resto = 2

n = 1 → 1/10 = 0; mostrar resto = 1

Salida por pantalla = 54321

Capicúa

Un número se dice que es capicúa si se lee igual de derecha a izquierda. Por ejemplo son números capicúas el 161, 2992, 3003 etc. Se pide:

1. Implementar un módulo en C++ en el que dado un número entero positivo, indique si es capicúa, de decir, si el número no cambia al darle la vuelta.

2. Implementar su correspondiente módulo de pruebas

Fibonacci

La Sucesión de Fibonacci es una secuencia de números naturales, que empieza con 0 y 1, y cuyos siguientes términos están definidos como la suma de los dos anteriores:
0, 1, 1, 2, 3, 5, 8, 13, 21, 34, 55, 89, 144, ...

Podemos definir la función de Fibonacci de manera recursiva del siguiente modo:

Fibonacci(n):

- 0, si n = 0
- 1, si n = 1
- Fibonacci(n-2) + Fibonacci(n-1), si n > 1

A partir de esta definición se pide implementar una función recursiva y otra iterativa que genere el n-ésimo término de la sucesión.