ANÁLISIS Y DISEÑO DE DATOS Y ALGORITMOS -
ESTRUCTURA DE DATOS Y ALGORITMOS - CURSO 2016/2017

PRÁCTICA 1: RECURSIVIDAD EN C

OBJETIVOS

- Recordar el uso del IDE de eclipse para C ya visto en primero.
- Implementar en C algunos ejercicios de recursividad vistos en clase de teoría y problemas.

PRERREQUISITOS

Cree una carpeta llamada `WS<usuario>Pract01`, donde <usuario> es su nombre. Posteriormente usará esta carpeta como espacio de trabajo (workspace), es decir, todos los proyectos que cree con la herramienta del IDE de eclipse deberán almacenarse dentro de esta carpeta.

Si no tiene Eclipse para C instalado puede descargarlo en http://www.eclipse.org/downloads/packages/eclipse-ide-cc-developers/neonr.

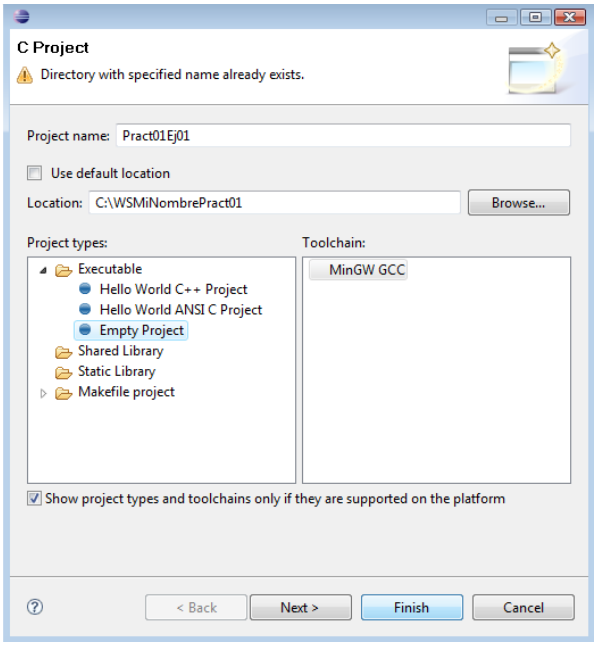
Finalmente para poder compilar y enlazar los programas C se hace necesario instalar gcc. Para los usuarios de Windows se puede instalar MinGW. Puede descargarlo en el enlace: http://sourceforge.net/projects/tdm-gcc/?source=dlp. Al instalar este programa las variables de entorno del sistema operativo se configuran automáticamente. Si usara otro gcc tenga en cuenta que ha de configurar la variable *Path* de las variables del sistema e indicar el directorio del gcc.

EJERCICIOS PROPUESTOS

EJERCICIO 1 – FACTORIAL RECURSIVO

Implemente el algoritmo recursivo de factorial para números enteros. Para ello, siga los siguientes pasos.

1) Cree un nuevo proyecto haciendo clic en **File – New – C Project**. Cree un nuevo proyecto de nombre *Pract01Ej01 y* seleccione como Tipo de Proyecto: **Executable (empty project)** y posteriormente **Toolchains** (herramientas): **MinGW GCC** y finalmente pulse sobre Finish.

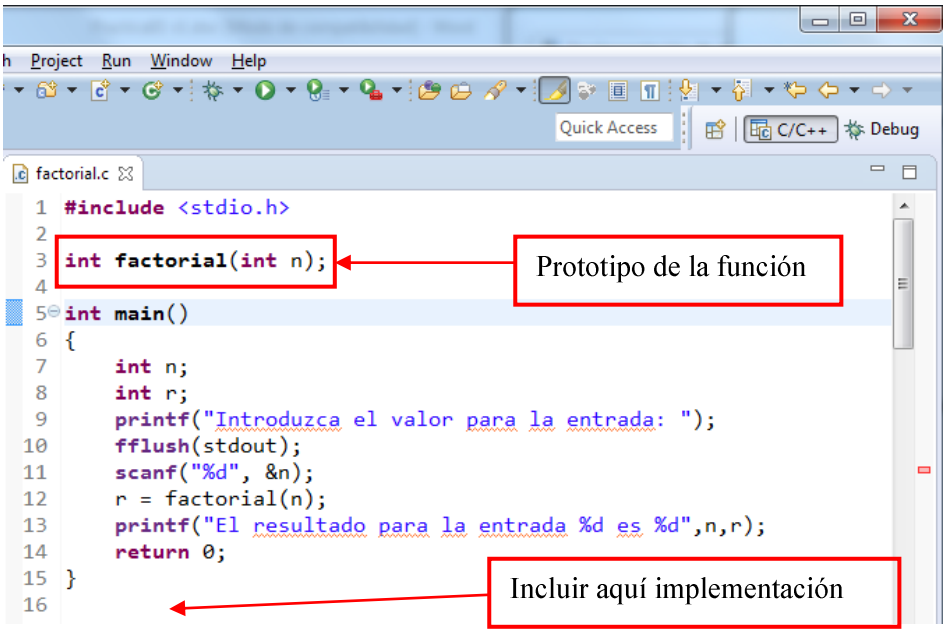


2) Incorpore al proyecto el archivo *factorial.c* suministrado con la práctica en la carpeta de proyecto (puede hacerlo copiando y pegando el archivo o arrastrándolo).

3) Implemente, dentro de ese archivo, la función *factorial* vista en teoría:

$$f(n) = \begin{cases} 1 & \text{si } n = 0 \\ n \cdot f(n-1) & \text{si } n > 0 \end{cases}$$

Nota: Recuerde incluir la definición del prototipo de la función entre las importaciones y la función *main*.



4) Compile y ejecute el proyecto y verifique que funciona correctamente.
   a. Recuerde que para compilar y enlazar el proyecto tiene que pulsar en la barra de menú: **Project → Build All.** En caso de errores en la compilación, es recomendable hacer previamente un Project → Clean Project, y luego repetir el paso de Build All.
   b. Se ejecuta el proyecto eligiendo en la barra de menú: **Run As →** *Local C/C++ Applications* **,** o simplemente **Run.**
   c. En la consola se visualizará el resultado.

5) ¿Es recursividad simple o múltiple? ¿Es final o no final?

6) Implemente dentro de *factoria.c* la función *factorialFinal* con la versión recursiva final, usando los esquemas y plantillas vistos en teoría.

7) A partir de la versión recursiva final deduzca la versión iterativa, y realice su implementación incorporando una nueva función *factorialIterativa* dentro de *factorial.c.*

8) Compruebe el correcto funcionamiento de las tres implementaciones para varios casos de prueba y compare los resultados obtenidos.

EJERCICIO 2 – POTENCIA RECURSIVA

Implemente el algoritmo recursivo de la potencia para números enteros. Para ello siga los siguientes pasos.

1) Cree un nuevo proyecto llamado *Pract01Ej02* de la misma manera que lo hizo en el ejercicio anterior.

2) Incorpore al proyecto el archivo *potencia.c* suministrado con la práctica en la carpeta de proyecto e implemente la definición recursiva de potencia:

$$a^n = \begin{cases} 1 & \text{si } n = 0 \\ a \cdot a^{n-1} & \text{si } n > 0 \end{cases}$$

3) Compile y ejecute el programa y verifique que funciona correctamente.

4) ¿Es recursividad simple o múltiple? ¿Es final o no final?

5) Implemente una nueva función *potenciaFinal* en *potencia.c* a partir de la solución anterior y los esquemas recursivos visto en teoría.

6) Implemente una nueva función *potenciaIterativa* en *potencia.c* a partir de la solución anterior y los esquemas visto en teoría.

7) Compruebe el correcto funcionamiento de las tres implementaciones para varios casos de prueba y compare los resultados obtenidos.

EJERCICIO 3 – SUMA RECURSIVA

Implemente el algoritmo recursivo de la suma de elementos de un vector de números enteros. Para ello siga los siguientes pasos.

1) Cree un nuevo proyecto llamado *Pract01Ej03*.

2) Incorpore al proyecto el archivo s*uma.c* proporcionado e implemente la definición recursiva siguiente:

$$sum(i) = \begin{cases} a[i] & \text{si i} = 0 \\ a[i] + sum(i-1) & \text{si i} > 0 \end{cases}$$

   a. Por simplicidad, defina al principio del archivo *suma.c* la tabla de números a sumar. Utilice una tabla con los números del 0 al 9.
   b. Defina también una constante con el tamaño de la tabla llamada *TAM*. En este caso la tabla tendrá 10 elementos.

3) La función *suma* debe implementarse siguiendo el esquema general visto en teoría. Indique que esquemas recursivos de los visto en clase de teoría servirían para solucionar este problema.

8) Compile y ejecute el programa y verifique que funciona correctamente.

9) ¿Es recursividad simple o múltiple? ¿Es final o no final?

10) Implemente dentro de *suma.c* la función *sumaFinal* con la versión recursiva final, usando los esquemas y plantillas vistos en teoría.

11) Implemente dentro de *suma.c* la función *sumaIterativo* con la versión iterativa, usando los esquemas y plantillas vistos en teoría.

12) Compruebe el correcto funcionamiento de las tres implementaciones para varios casos de prueba y compare los resultados obtenidos.

Nota: como se ha comentado, definir la tabla a sumar en el archivo de funciones simplifica el código pero, a cambio lo hace mucho menos reutilizable. Para aumentar la reusabilidad de esta solución seria más adecuado poder indicar con algún parámetro el cuál es la tabla a sumar y su tamaño. Como ejercicio alternativo puede intentar plantear cómo sería el diseño de esta solución.

EJERCICIO 4 – ALGORITMO DE EUCLIDES

Implemente el algoritmo recursivo del algoritmo de Euclides para el cálculo del máximo común divisor. Para ello siga los siguientes pasos.

1) Cree un nuevo proyecto llamado *Pract01Ej04*.

2) Incorpore al proyecto el archivo *euclides.c* proporcionado e implemente la definición recursiva del problema :

$$mcd(a,b) = \begin{cases} a & \text{si } b = 0 \\ mcd(b,a) & \text{si a < b} \\ mcd(b, a\%b) & \text{si a > b} \end{cases}$$

3) Compile y ejecute el programa y verifique que funciona correctamente.

4) ¿Es recursividad simple o múltiple? ¿Es final o no final?

5) Implemente dentro de *euclides.c* la función *mcdIterativo* con la versión iterativa, usando los esquemas y plantillas vistos en teoría.

6) Compruebe el correcto funcionamiento de las dos implementaciones para varios casos de prueba y compare los resultados obtenidos.

EJERCICIO 5 – NÚMEROS DE FIBONACCI

Implemente el algoritmo recursivo del cálculo de los números de Fibonacci. Para ello siga los siguientes pasos.

1) Cree un nuevo proyecto llamado *Pract01Ej05*.

2) Incorpore al proyecto el archivo *fibonacci.c* suministrado con la práctica e implemente la definición recursiva del problema:

$$fib(n) = \begin{cases} n & \text{si } 0 \le n \le 1 \\ fib(n-1) + fib(n-2) & \text{si } n > 1 \end{cases}$$

3) Compile y ejecute el programa y verifique que funciona correctamente.

4) ¿Es recursividad simple o múltiple? ¿Es final o no final?

EJERCICIO 6 – COMBINACIONES SIN REPETICIÓN

Implemente el algoritmo recursivo de las combinaciones sin repetición de n objetos tomados de k en k. Para ello siga los siguientes pasos.

1) Cree un nuevo proyecto llamado *Pract01Ej06*.

2) Incorpore al proyecto el archivo *repeticiones.c* proporcionado con la práctica al proyecto e implemente la definición recursiva del problema:

$$nc(n,k) \begin{cases} 0 & \text{si } k > n \\ 1 & \text{si } k = 0 \text{ ó } k = n \\ nc(n-1, k) + nc(n-1, k-1) & \text{en otro caso} \end{cases}$$

3) Compile y ejecute el programa y verifique que funciona correctamente.

4) ¿Es recursividad simple o múltiple? ¿Es final o no final?

EJERCICIO 7 – DIVISIÓN RECURSIVA

En este ejercicio se implementará el algoritmo recursivo de la operación de la división aplicando la operación resta de manera recursiva. Así, si se desea realizar A / B y obtener el resultado S y el resto R, el proceso consistirá en restar a A la cantidad B hasta que el resultado de la resta sea menor que el propio B. S será el número de operaciones de resta realizadas y R el resultado de la última resta.

La función recursiva necesita conocer más de un parámetro. La función recursiva necesita saber cuál es el A actual, cuál es el B y cuál es el número de restas que se han realizado hasta el momento (S en el párrafo anterior).

1) Realice la definición recursiva de la función que se propone.

2) Cree un nuevo proyecto llamado *Pract01Ej07*.

3) Incorpore al proyecto el archivo *division.c* que se suministra con la práctica e implemente la función que se propone.

4) Solicite por consola numerador y denominador y muestre el resultado de la división y el resto.

5) ¿Es recursividad simple o múltiple? ¿Es final o no final?

6) Implemente dentro de *division.c* la función *divisionIterativo* con la versión iterativa, usando los esquemas y plantillas vistos en teoría.

7) Compruebe el correcto funcionamiento de las tres implementaciones para varios casos de prueba y compare los resultados obtenidos.

Nota: tenga en cuenta que para usar parámetros de entrada y salida en las funciones en C debe usar puntero, es decir, * en su prototipo. Ejemplo, int *s como parámetro significa parámetro de entrada salida de la función.