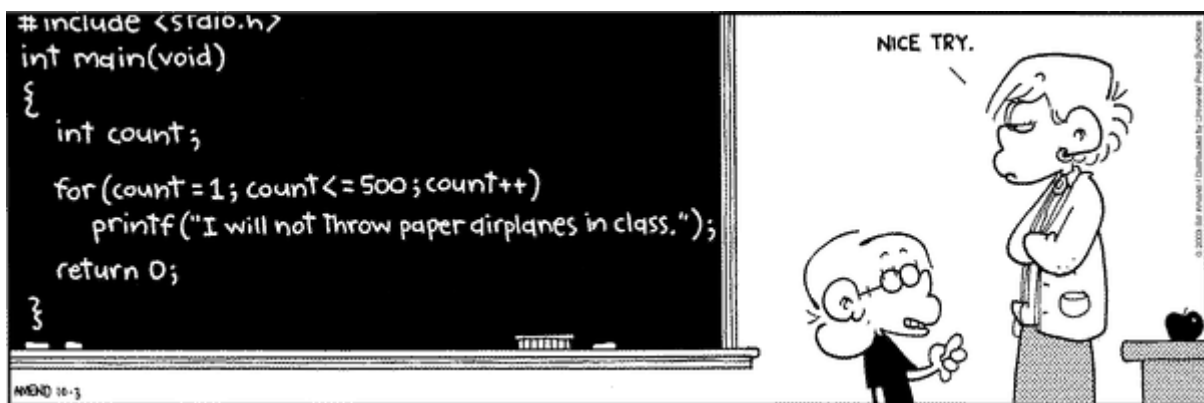


Fundamentos de programación

Ejercicios



Curso 2017/18

Ejercicios de Fundamentos de Programación - revisión 01/05/2017

Copyright © Alejandro Castán Salinas

Se otorga el permiso para copiar, distribuir y/o modificar este documento bajo los términos de la licencia de documentación libre GNU, versión 1.3 o cualquier otra versión posterior publicada por la *Free Software Foundation*.

Puedes consultar dicha licencia en <http://www.gnu.org/copyleft/fdl.html>.

El contenido de este documento puede cambiar debido a ampliaciones y correcciones enviadas por los lectores. Encontrarás siempre la última versión del documento en <http://www.xtec.net/~acastan/textos/>.

Índice de contenido

Práctica 1: Arquitectura del computador.....	5
Práctica 2: Codificación de la información.....	7
Práctica 3: Lenguajes de programación, compiladores e intérpretes, y entornos de desarrollo.....	10
Práctica 4: Algoritmos secuenciales.....	20
Práctica 5: Estructuras de control alternativas.....	22
Práctica 6: Estructuras de control iterativas.....	26
Práctica 7: Estructuras de almacenamiento homogéneas unidimensionales (vectores y strings).....	30
Práctica 8: Estructuras de almacenamiento homogéneas multidimensionales (matrices).....	34
Práctica 9: Estructuras de almacenamiento heterogéneas (registros).....	37
Práctica 10: Funciones y modularidad.....	42
Práctica 11: Almacenamiento en ficheros.....	52
Práctica 12: Apuntadores y estructuras dinámicas de datos.....	62
Trabajos finales.....	69
¡Fin!.....	72

Práctica 1: Arquitectura del computador

Objetivos de la práctica

- Introducción a la arquitectura y funcionamiento interno del ordenador.
- Conocimiento de los diferentes dispositivos de la máquina.
- Comprensión de los términos: CPU, memoria, bus, periférico, sistema operativo, fichero.
- Identificación y solución de pequeños problemas de hardware y software.

Ejercicios Obligatorios

1.1 Escribe en una hoja en blanco dos o tres términos relacionados con el mundo de la informática que no llegues a entender bien. Durante cinco minutos busca información en Internet o entre los compañeros sobre uno de estos términos. A continuación haz una puesta en común de los resultados obtenidos, intentando resolver las dudas de los compañeros (y pidiendo su ayuda y la del profesor en las dudas que todavía tengas).

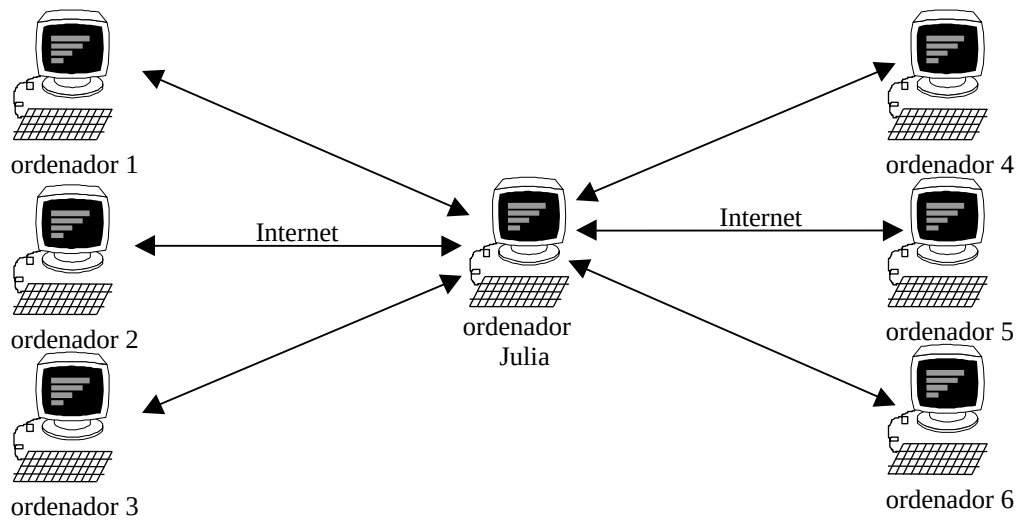
1.2 Haz un dibujo o esquema de la arquitectura de un ordenador. A continuación, en grupos de tres personas poned en común vuestros esbozos y realizad un esquema final conjunto, donde debe constar:

- Nombre de los diferentes componentes.
- Función de cada componente.
- Qué componentes son imprescindibles y cuáles no.

Por último, abrid uno o más ordenadores, e identificad las partes que aparecen en el esquema anterior.

1.3 Imagina que eres vendedor/a de una tienda de informática. Aconseja a los siguientes tres clientes que llegan a la vuestra tienda sobre cual es el ordenador que se ajusta a sus necesidades, es decir, qué componentes y accesorios necesitaran, como deben ser éstos y el porqué. Como lista de componentes tenemos: microprocesador, memoria RAM, disco duro, módem, tarjeta de red, tarjeta de vídeo, tarjeta de sonido, caja y s.a.i.

- Cliente 1: Javi es un chico que ya tiene un ordenador. Lo utiliza sobretodo para jugar: es un apasionado de los videojuegos. El problema es que los últimos juegos que ha comprado ya van un poco lentos en su ordenador. ¿Qué componentes de su ordenador cambiaríais para mejorar el rendimiento en los juegos? Además Javi también utiliza el ordenador como a asistente a la hora de componer música.
- Cliente 2: Julia es una ingeniera. En el trabajo necesita un ordenador para hacer simulaciones de dinámicas de fluidos (cálculos muy costosos y muchísimos datos). El resto de ordenadores del trabajo accederán constantemente a este ordenador mediante su red local para consultar los resultados de las simulaciones. Imagina:



- Cliente 3: Juan trabaja en una oficina. Quiere un ordenador para poder escribir en casa informes y llevárselo del trabajo a casa.
- ¿Y tú? ¿Cómo es el ordenador que necesitas?

Ejercicios de intensificación

- 1.4 Busca información y haz un breve resumen sobre cuáles han sido los cambios más importantes en la historia de los microprocesadores de la familia Intel 80x86, desde el 8086 hasta el i7.
- 1.5 ¿Qué características diferencian los sistemas operativos Windows y Linux?

Práctica 2: Codificación de la información

Objetivos de la práctica

- Cambios de base (binario, hexadecimal y decimal).
- Codificación de números enteros con signo.
- Aritmética binaria.
- Codificación de números reales en simple y doble precisión.
- Memoria y precisión.
- Códigos detectores y correctores de errores. Capacidad del código.

Introducción

El ordenador debe trabajar con información de diferentes tipos: instrucciones de programas, números, letras y símbolos, etc. Las diferentes tecnologías que utiliza el ordenador (almacenamiento óptico en DVD, almacenamiento magnético en discos duros, almacenamiento eléctrico en transistores, etc.) hacen que dicha información sólo se pueda almacenar utilizando secuencias de dos estados (bits), a los que llamamos estado 0 y estado 1. Por lo tanto, debemos encontrar la manera de codificar la información como secuencias de ceros y unos.

Además, en las tecnologías que almacenan o transportan la información se pueden producir errores que alteren dicha información. Para subsanar este problema, a los datos a almacenar o transportar se les añade un pequeño porcentaje de información redundante que permite detectar si se ha producido error, e incluso corregirlo. Estamos hablando de los códigos detectores y correctores de errores.

En las siguientes direcciones podemos aprender estos conceptos (aviso: la información en la Wikipedia inglesa actualmente es más completa):

- http://es.wikipedia.org/wiki/Sistema_de_numeración
- http://es.wikipedia.org/wiki/Tipo_de_dato_entero
- http://es.wikipedia.org/wiki/IEEE_punto_flotante
- http://es.wikipedia.org/wiki/Codificación_de_caracteres
- http://es.wikipedia.org/wiki/Detección_de_errores



Cambios de base

“Sólo existen 10 tipos de personas: las que entienden binario y las que no.”

“¿Por qué los/as programadores/as confunden Halloween con Navidad? porque $31_{(OCT)} = 25_{(DEC)}$ ”

2.1 Completa la siguiente tabla:

Binario	Hexadecimal	Decimal
		128.75
10011101.11001		
	BE.A7	

Codificación de números enteros con signo

2.2 Representa los números 223 y -223 en binario en los diferentes códigos para la representación de enteros con signo:

		Binario
Binario natural	223	
	-223	
Signo y magnitud	223	
	-223	
Complemento a 2	223	
	-223	
Exceso (256)	223	
	-223	

2.3 Calcula el valor decimal del número binario 10100111 representado en los diferentes códigos.

		Decimal
Binario natural	10100111 ₂	
Signo y magnitud	10100111 ₂	
Complemento a 2	10100111 ₂	
Exceso (128)	10100111 ₂	

Aritmética binaria

2.4 Realiza las siguientes sumas en 8 bits y complemento a 2, dando el valor del resultado en decimal.

$$26 + 24 = \quad -15 + 82 = \quad 84 + 69 = \quad -46 + (-10) =$$

Codificación de números reales

2.5 Representa 123.12 según la norma IEEE 754 de simple precisión. Utilizar truncado o redondeo si fuera necesario.

Memoria y precisión

- 2.6 En un microprocesador de 16 bits deseamos ejecutar un programa que ocupa 20000 posiciones de memoria y que además necesita espacio para: 10000 números enteros (16 bits), 5000 números reales de simple precisión (32 bits), 2000 números reales de doble precisión (64 bits) y 1000 alarmas binarias (1 bit). Indica la capacidad mínima de la memoria necesaria en posiciones de memoria, bits, bytes y kilobytes.

Códigos detectores y correctores de errores

- 2.7 Queremos enviar por una línea telefónica una información en paquetes de 8+1 bits. Aplica un código de paridad par y otro impar para enviar la cadena de caracteres “Alumno” (sin las comillas) en código ASCII de 8 bits. Discute la posibilidad de detectar transmisiones erróneas.

Práctica 3: Lenguajes de programación, compiladores e intérpretes, y entornos de desarrollo

Objetivos de la práctica

- Conocer la historia de los lenguajes de programación más importantes.
- Entender los conceptos de código fuente, código máquina, compilador e intérprete.
- Familiarización con un entorno integrado de desarrollo.
- Familiarización con la estructura de un programa en lenguaje C.
- Familiarización con las funciones de entrada y salida del lenguaje C.
- Familiarización con los tipos de datos en C y comprensión de los errores debidos al rango de precisión de los tipos numéricos.

Algorítmica

Algoritmo es la exposición, paso a paso, de la secuencia de instrucciones que se ha de seguir para resolver un determinado problema.

Estructura de datos es una representación en forma lógica de la información, una manera de organizar una determinada información.

Lenguaje de programación son el conjunto de instrucciones que permiten controlar una máquina. Un programa es la descripción de un algoritmo en un determinado lenguaje de programación.

- <http://es.wikipedia.org/wiki/Algoritmo>
- http://es.wikipedia.org/wiki/Estructura_de_datos
- http://es.wikipedia.org/wiki/Lenguaje_de_programación



Diagramas de flujo

El diagrama de flujo es un lenguaje visual de descripción de algoritmos. La representación gráfica nos permite entender fácilmente el proceso, aunque para procesos muy complejos los diagramas de flujo se vuelven demasiado extensos y, por lo tanto, intratables.

- http://es.wikipedia.org/wiki/Diagrama_de_flujo
- <http://code.google.com/p/freedfd/>

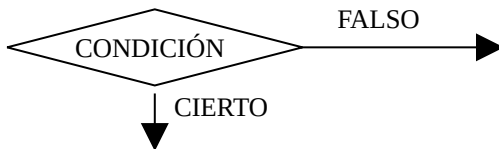
- <http://www.youtube.com/watch?v=VvUuey811PU>
- <http://drakon-editor.sourceforge.net/>



Las cajas con forma de elipse indican el inicio y el final del algoritmo



Las cajas con forma de rectángulo indican una operación, cálculos en general.

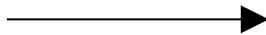


Las cajas con forma de rombos indican una decisión. Según sea el resultado de evaluar la expresión lógica el código se bifurcará hacia un camino o hacia otro.



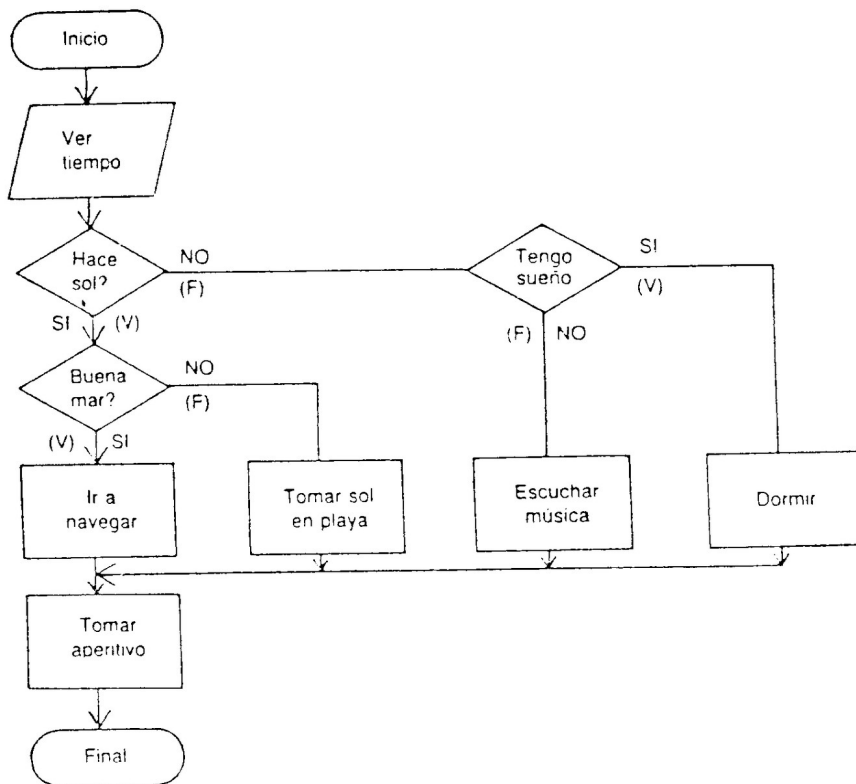
Las cajas con forma de trapezoides indican una petición de datos o una salida de resultados.

ENLAZAR
OPERACIONES

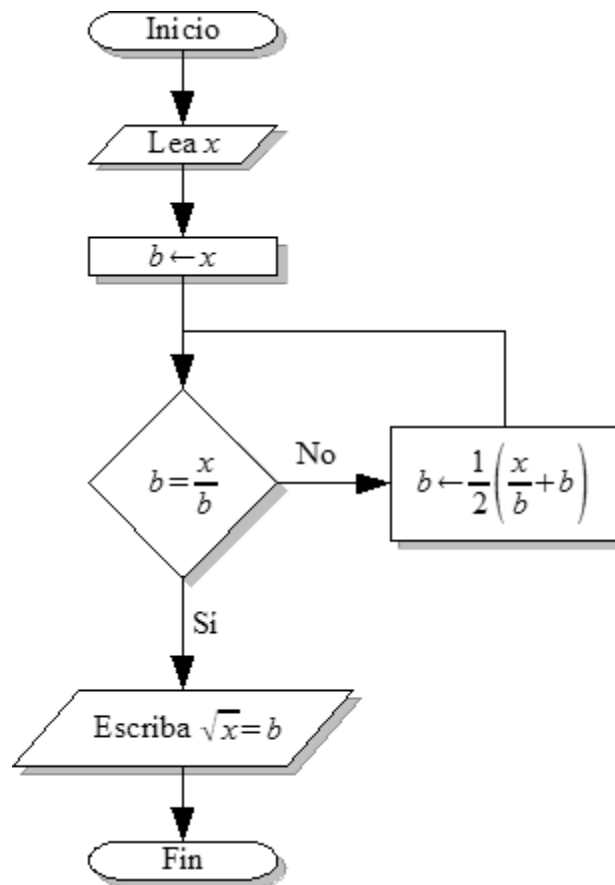


Las flechas enlazan los diferentes pasos del algoritmo y nos indican el orden de estos.

Ejemplo: descripción mediante diagrama de flujo del proceso a seguir un domingo por la mañana.



Ejemplo: descripción mediante diagrama de flujo del proceso a seguir para calcular una raíz cuadrada.



Pseudocódigo

Pseudocódigo es un lenguaje escrito de descripción de algoritmos, con una sintaxis más coloquial y menos rígida que la de los lenguajes de programación.

- <http://es.wikipedia.org/wiki/Pseudocódigo>
- <http://pseint.sourceforge.net/>

Ejemplo: descripción mediante pseudocódigo del proceso a seguir para calcular una raíz cuadrada.

```
programa RaizCuadrada
  pedir número x
  sea b = x
  mientras x/b sea muy diferente de b, hacer lo siguiente:
    actualizar el nuevo valor de b = (b + x/b) / 2
  fin del mientras
  decir b es la raíz cuadrada de x
fin del programa
```

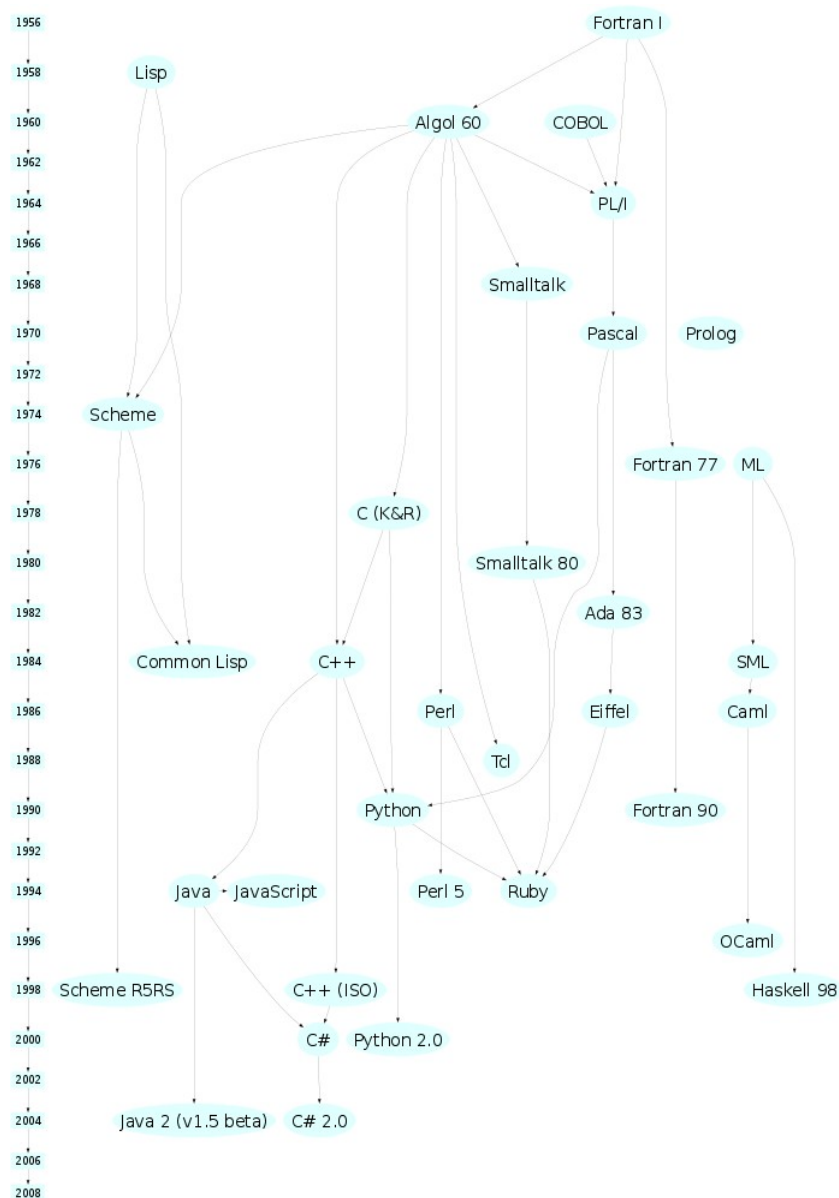
Lenguaje de programación

Los lenguajes de programación han evolucionado con el tiempo. Para hacernos una idea de dicha evolución y de las características que han ido incorporando podemos consultar las siguientes direcciones:

- http://es.wikipedia.org/wiki/Lenguaje_de_programación
- <http://www.digibarn.com/collections/posters/tongues/>
- http://cdn.oreillystatic.com/news/graphics/prog_lang_poster.pdf
- <http://www.levenez.com/lang/history.html>

Como curiosidad, en las siguientes direcciones podéis encontrar un mismo programa (“Hello, world!” y “99 Bottles of Beer”) escrito en centenares de lenguajes de programación diferentes:

- http://en.wikibooks.org/wiki/List_of_hello_world_programs
- <http://99-bottles-of-beer.net/>



Un compilador es el programa encargado en traducir un programa escrito en lenguaje de programación de alto nivel (código fuente) al lenguaje que es capaz de ejecutar un ordenador (código máquina). Para aprender más podemos consultar las siguientes direcciones:

- <http://es.wikipedia.org/wiki/Compilador>
- [http://es.wikipedia.org/wiki/Intérprete informático](http://es.wikipedia.org/wiki/Intérprete_informático)

Los ejercicios de este documento se pueden resolver en el lenguaje de programación que se desee. Para dar soluciones y alguna pequeña explicación yo he escogido lenguaje C. Para aprenderlo podéis encontrar innumerables cursos de lenguaje C en Internet. Ahí van tres direcciones para comenzar:

- Historia de C: [http://es.wikipedia.org/wiki/Lenguaje de programación C](http://es.wikipedia.org/wiki/Lenguaje_de_programación_C)
- Curso de C: [http://es.wikibooks.org/wiki/Programación en C](http://es.wikibooks.org/wiki/Programación_en_C)
- Normas de estilo de programación en C: [http://es.wikipedia.org/wiki/Estilo de programación](http://es.wikipedia.org/wiki/Estilo_de_programación)

Para trabajar vamos a utilizar un Entorno Integrado de Desarrollo (IDE), donde podemos escribir nuestro programa, compilarlo y ejecutarlo, recibir ayuda sobre errores y sintaxis, y ejecutar instrucción a instrucción visualizando los valores que toman las variables (“depuración”).

Existen IDEs ligeros, libres y gratuitos tanto para Linux como para Windows y MacOS. Por ejemplo, yo os recomiendo Code::Blocks. Existen muchos otros IDEs más pesados y completos, también libres y gratuitos: Eclipse, Netbeans, Kdevelop, ...

Si trabajas en un ordenador que no permite instalar software, tienes un IDE online al que puedes acceder mediante navegador en <https://www.onlinegdb.com/>

Ejercicios Obligatorios

3.1 Copia el siguiente programa en el entorno de desarrollo y guárdalo con el nombre `ex_3_1.c`:

```
#include <stdio.h>

int main(void) {
    int i, j, k;

    printf("Suma de dos enteros.\n");
    printf("Primer número: ");
    scanf("%d", &i);
    printf("Segundo número: ");
    scanf("%d", &j);
    k = i + j;
    printf("Resultado = %d\n", k);

    return 0;
}
```

Compila el programa y, si la compilación no da ningún error, ejecútalo. Prueba el programa como mínimo con los siguientes valores, comprobando los resultados y averiguando que sucedió en caso que dichos resultados no sean correctos:

200 +	350
250 +	-300
20000 +	30000

$$\begin{array}{rcl} 40000 & + & -10 \\ 2000000000 & + & 2000000000 \end{array}$$

3.2 Modifica el programa anterior y guárdalo con el nombre *ex_3_2.c*:

```
#include <stdio.h>

int main(void) {

    float i, j, k;

    printf("Suma de dos reales.\n");
    printf("Primer número: ");
    scanf("%f", &i);
    printf("Segundo número: ");
    scanf("%f", &j);
    k = i + j;
    printf("Resultado = %f\n", k);

    return 0;
}
```

Compila el programa y, si la compilación no da ningún error, ejecútalo. Prueba el programa como mínimo con los siguientes valores, comprobando los resultados y averiguando que sucedió en caso que dichos resultados no sean correctos:

$$\begin{array}{rcl} 250.30 & + & 300.50 \\ 1E12 & + & 1.0 \\ 333333333333333.333 & + & 6.667 \\ 999999999999999.0 & + & 999999999999999.0 \\ 123456789876543 & + & 0 \end{array}$$

Pista: para saber qué ha pasado en los dos programas anteriores realiza una ejecución paso a paso con seguimiento de variables ...

- Abre con el entorno de desarrollo cualquiera de los dos programas anteriores.
- Abre una ventana para visualizar el valor de la variable *i*. Repite este proceso para visualizar también el valor de las variables *j* y *k*. Si fuera necesario cambia el tamaño de las ventanas para poder visualizar el código del programa y el seguimiento de las variables a la vez.
- Prueba el programa ejecutándolo paso a paso con los valores proporcionados anteriormente, estando atento del momento en que se producen desbordamientos y pérdidas de precisión.

3.3 Copia el siguiente programa en el entorno de desarrollo y guárdalo con el nombre *ex_3_3.c*:

```
#include <stdio.h>

int main(void) {

    /* Definición de variables */
    int  dato1;
    float dato2;
    char dato3 , dato4;

    printf("Pruebas de formatos de impresión\n");
    printf("-----\n\n");

    /* Inicializamos las variables */
    dato1 = 205;
```

```

dato2 = 205.5;
dato3 = 'a';
dato4 = 'b';

/* Pruebas */
printf("Entero 205 sin formato 2 veces : %i %i\n" , dato1 , dato1);
printf("Entero 205 con formato (6) : %6i\n" , dato1);
printf("Real 205.5 sin formato : %f\n" , dato2);
printf("Real 205.5 con formato (exp) : %e\n" , dato2);
printf("Real 205.5 con formato (12) : %12f\n" , dato2);
printf("Real 205.5 con formato (12.0) : %12.0f\n" , dato2);
printf("Real 205.5 con formato (12.2) : %12.2f\n\n" , dato2);

printf("Char 'a' y 'b' sin formato : %c %c\n" , dato3 , dato4);
printf("Char 'a' y 'b' con formato (6) : %6c %6c\n" , dato3 , dato4);
printf("Palabras sin formato : | %s | %s |\n" , "mesa" , "silla");
printf("Palabras alineadas derecha (6) : | %6s | %6s |\n" , "mesa" , "silla");
printf("Palabras alineadas izquierda (6) : | %-6s | %-6s |\n" , "mesa" , "silla");

printf("\nLínea completa con entero 205(6), real 205.5(8.2) y 'mesa'(8)\n");
printf("%6i %8.2f %8s\n\n" , dato1 , dato2 , "mesa");

return 0;
}

```

Compila el programa y ejecútalo. Observa como los parámetros proporcionados a la función *printf* alteran el formato de la impresión.

3.4 ¿Cuál es la diferencia más destacable entre un compilador y un intérprete? Nombra tres lenguajes de programación compilados y tres más interpretados.

3.5 ¿Qué valor se almacena en las variables *y* (de tipo int) y *x* (de tipo float) al ejecutar cada una de estas sentencias?

- a) *y* = 2;
- b) *y* = 1 / 2;
- c) *y* = 2 / 4;
- d) *y* = 2.0 / 4;
- e) *x* = 2.0 / 4.0;
- f) *x* = 2.0 / 4;
- g) *x* = 2 / 4;
- h) *x* = 1 / 2;

3.6 ¿Qué valor se almacena en las variables *y* (de tipo int) y *x* (de tipo float) al ejecutar cada una de estas sentencias?

- a) *y* = (float) 2;
- b) *y* = 1 / (float) 2;
- c) *y* = (int) (2 / 4);
- d) *y* = (int) 2. / (float) 4;

- e) `x = 2.0 / (int) 4.0;`
- f) `x = (int) 2.0 / 4;`
- g) `x = (int) (2.0 / 4);`
- h) `x = 2 / (float) 4;`

3.7 Copia el siguiente programa en el entorno de desarrollo y guárdalo con el nombre `ex_3_7.c`:

```
#include <stdio.h>

typedef char string[100];

int main(void) {
    int edad;
    string alumno;

    printf("Dime tu edad: ");
    scanf("%d", &edad);
    getchar();
    printf("Dime tu nombre y apellidos: ");
    gets(alumno);
    printf("¡Bienvenido %s!\n", alumno);

    return 0;
}
```

- a) ¿Por qué añadimos la función `getchar()`? ¿Qué pasa si no está?
- b) ¿Por qué tenemos la función `gets(...)` en lugar de la función `scanf("%s", ...)`?
- c) ¿Por qué la compilación da un “warning”? ¿Cómo podemos solucionarlo?

3.8 Si estás en clase, vas a jugar con los compañeros. Forma un grupo de dos a cuatro personas y toma el código de un ejercicio anterior. Mientras una persona no mira, el resto añaden un error al programa. La persona que no miraba debe encontrar dicho error. Si lo encuentra a simple vista suma dos puntos, o si lo encuentra usando el compilador suma un punto, pero si no lo encuentra será la persona que puso el error la que suma un punto.

Anexo: C, C++ y Java

La estructura de un programa en C es:

```
/* (opcional) Incluir bibliotecas del sistema y propias que utilizará el programa */

#include <librería.h>
#include "mi_librería.h"

/* (opcional) Declaración de contantes */

#define NOMBRE_CONSTANTE valor_constante

/* (opcional) Declaración de estructuras y de nuevos tipos de datos */

typedef declaración_nuevo_tipo nombre_nuevo_tipo;

struct nombre_struct { declaración_campos_struct };
```

```

/* (opcional) Declaración de variables globales */

tipo_datos nombre_variable;

/* (opcional) Declaración de funciones */

valor_retorno nombre_función(definición_parámetros) {
    [tipo_datos nombre_variable1; ...;]           // Declaración de variable locales
    instrucción1; [instrucción2; ...;]           // Bloque de instrucciones
}

/* (obligatorio) Programa principal , que es una función más */

int main(void) {
    [tipo_datos nombre_variable1;]
    [tipo_datos nombre_variable2; ...;]
    instrucción1;
    [instrucción2; ...;]
    return 0;
}

```

El siguiente programa en C:

```

void main (void) {
    int a, b, suma;

    printf("Suma de dos enteros\n");
    printf("Entre el dato 1:");
    scanf("%i", &a);
    printf("Entre el dato 2:");
    scanf("%i", &b);
    suma = a + b;
    printf("La suma vale: %6i\n", suma);
}

```

En C++ sería:

```

#include <iostream.h>
#include <iomanip.h>

void main (void) {
    int a, b, suma;

    cout << "Suma de dos enteros" << endl;
    cout << "Entre el dato 1:";
    cin >> a;
    cout << "Entre el dato 2:";
    cin >> b;
    suma = a + b;
    cout << "La suma vale: " << setw(6) << suma << endl;
}

```

En Java sería:

```

import java.io.*;

class Suma {
    public static void main(String[] args) {
        int a, b, suma;
        Scanner teclado = new Scanner( System.in );

        System.out.println("Suma de dos enteros");
        System.out.print("Entre el dato 1:");
    }
}

```

```
a = teclado.nextInt();  
System.out.print("Entre el dato 2:");  
b = teclado.nextInt();  
suma = a + b;  
System.out.println("La suma vale: " + suma);  
}  
}
```

Práctica 4: Algoritmos secuenciales

Objetivos de la práctica

- Trabajo con expresiones aritméticas.
- Trabajo con expresiones lógicas.
- Trabajo con asignaciones.
- Trabajo con operaciones de entrada y salida.

Repaso previo

- http://es.wikibooks.org/wiki/Programación_en_C/Fundamentos_de_programación
- http://es.wikibooks.org/wiki/Programación_en_C/Historia_de_C
- http://es.wikibooks.org/wiki/Programación_en_C/Primer_programa_en_C
- http://es.wikibooks.org/wiki/Programación_en_C/Tipos_de_datos
- http://es.wikibooks.org/wiki/Programación_en_C/Expresiones
- http://es.wikibooks.org/wiki/Programación_en_C/Interacción_con_el_usuario

Ejercicios Obligatorios

- 4.1 Observa el siguiente programa en C llamado *ex_4_1*, que implementa el cálculo del área y el perímetro de un círculo, dado un radio r , según las fórmulas $\text{área} = \pi * r^2$ y $\text{perímetro} = 2 * \pi * r$

```
#include <stdio.h>

#define PI 3.1416

int main(void) {

    float r, a, p;

    /* Pedimos el radio */
    printf("Introduce radio del círculo: ");
    scanf("%f", &r);

    /* Calculamos el área y perímetro */
    a = PI * r * r;
    p = 2 * PI * r;

    /* Damos los resultados */
    printf("Área = %f\n", a);
    printf("Perímetro = %f\n", p);

    return 0;
}
```

¿Qué variables son de entrada, qué variables son de salida, y cuáles auxiliares?

¿Por qué PI no lo declaramos como una variable?

¿Por qué ponemos un mensaje (“printf”) delante de la entrada de datos (“scanf”)? ¿Es obligatorio hacerlo así? ¿Qué pasaría si no?

4.2 Crea un programa llamado *ex_4_2*, que pida tres notas y calcule la media.

4.3 Crea un programa llamado *ex_4_3*, que pida dos puntos del espacio bidimensional y calcule el punto medio según la fórmula:

$$\text{Sean los puntos } \vec{a} = (a_x, a_y) \text{ y } \vec{b} = (b_x, b_y) \text{ entonces } \vec{m} = \vec{a} + \vec{b} = \left(\frac{a_x + b_x}{2}, \frac{a_y + b_y}{2} \right)$$

Ejercicios Adicionales

4.4 Crea un programa llamado *ex_4_4*, que dado un número entero que designa un periodo de tiempo expresado en segundos, imprima el equivalente en días, horas, minutos y segundos.

Por ejemplo: 24000 segundos serán 0 días, 6 horas, 40 minutos y 0 segundos.

Por ejemplo: 7400 segundos serán 0 días, 2 horas, 3 minutos y 20 segundos.

Práctica 5: Estructuras de control alternativas

Objetivos de la práctica

- Trabajo con condiciones y expresiones lógicas: `&&`, `||`, `!`, `<`, `<=`, `>`, `>=`, `==`, `!=`
- Trabajo con la estructura de control alternativa simple: *if ...*
- Trabajo con la estructura de control alternativa doble: *if ... else ...*
- Trabajo con estructuras de control alternativas anidadas: *... else if ...*
- Trabajo con estructuras de control alternativas múltiples: *switch ... case ...*

Repaso previo

- http://es.wikibooks.org/wiki/Programación_en_C/Instrucciones_de_control

Ejercicios Obligatorios

- 5.1 Sean *A*, *B* y *C* tres variables enteras que representan las ventas de tres productos *A*, *B* y *C*, respectivamente. Utilizando dichas variables, escribe las expresiones que representen cada una de las siguientes afirmaciones:
- a) Las ventas del producto *A* son las más elevadas.
 - b) Ningún producto tiene unas ventas inferiores a 200.
 - c) Algún producto tiene unas ventas superiores a 400.
 - d) La media de ventas es superior a 500.
 - e) El producto *B* no es el más vendido.
 - f) El total de ventas esta entre 500 y 1000.
- 5.2 Dada una variable *c* de tipo carácter, consultando una tabla ASCII escribe las expresiones que representen las siguientes afirmaciones:
- a) *c* es una vocal.
 - b) *c* es una letra minúscula.
 - c) *c* es un símbolo del alfabeto.
- 5.3 Crea un programa llamado *ex_5_3*, que pida una contraseña numérica por teclado e indique si es correcta o incorrecta. La contraseña correcta es 123456.
- 5.4 Crea un programa llamado *ex_5_4*, que pida un número por teclado e indique si es positivo, negativo o cero. Intenta hacerlo con el mínimo número de comparaciones.

- 5.5 Observa el siguiente programa en C llamado *ex_5_5*, que introducidos tres números cualquiera por teclado, calcula el mínimo y el máximo de los tres y muestra el resultado por pantalla:

```
#include <stdio.h>

int main(void) {

    float a, b, c;    /* los valores introducidos */
    float min, max;   /* el mínimo y el máximo */

    printf("Introduce el primer valor: ");
    scanf("%f", &a);
    min = a;
    max = a;

    printf("Introduce el segundo valor: ");
    scanf("%f", &b);
    if (b < min)
        min = b;
    else
        max = b;

    printf("Introduce el tercer valor: ");
    scanf("%f", &c);
    if (c < min)
        min = c;
    else
        if (c > max)
            max = c;

    printf("El mínimo es %f\n", min);
    printf("El máximo es %f\n", max);

    return 0;
}
```

¿Por qué se utilizan estructuras alternativas anidadas?

¿Este programa tiene comparaciones repetidas o innecesarias? ¿Se podría hacer de otra manera con menos comparaciones?

- 5.6 Observa el siguiente programa en C llamado *ex_5_6*, que a partir del número del día de la semana introducido (del 1 al 7), si dicho día es laborable escribe el nombre del día correspondiente por la pantalla, y si no escribe festivo:

```
#include <stdio.h>

int main(void) {

    int dia;

    printf("Introduce un día de la semana (entre 1 y 7) : ");
    scanf("%d", &dia);

    printf("El día es ... ");

    switch (dia) {
        case 1: printf("lunes\n");    break;
        case 2: printf("martes\n");  break;
    }
```

```

    case 3: printf("miércoles\n"); break;
    case 4: printf("jueves\n");    break;
    case 5: printf("viernes\n");  break;
    case 6:
    case 7: printf("festivo\n");   break;
    default: printf("incorrecto\n");
}

return 0;
}

```

¿Cuándo se puede utilizar la alternativa múltiple en lenguaje C y cuándo no?

¿Se podría escribir este programa con estructuras alternativas *if* anidadas, en lugar de la alternativa múltiple *switch*? ¿Qué ventajas tiene entonces el *switch* sobre el *if*?

5.7 Crea un programa llamado *ex_5_7*, que pida los coeficientes *a* y *b* de una ecuación de primer grado y calcule la solución.

$$ax + b = 0$$

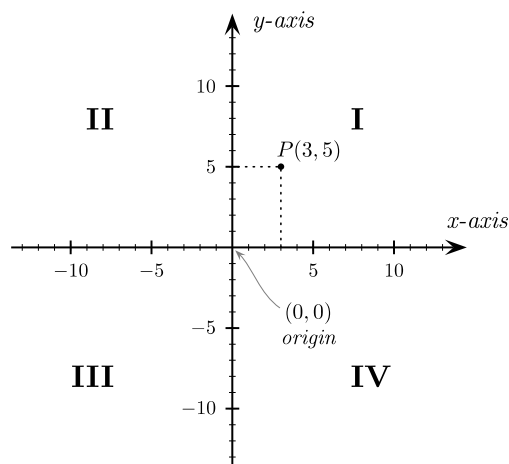
Ten en cuenta que existen tres posibles soluciones:

- Cuando $a \neq 0$ existe la solución única $x = -b/a$.
- Cuando $a = 0$ y $b \neq 0$ no existe solución.
- Cuando $a = 0$ y $b = 0$ existen infinitas soluciones.

5.8 Crea un programa llamado *ex_5_8*, que pida por teclado el tamaño de un tornillo y muestre por pantalla el texto correspondiente al tamaño, según la siguiente tabla:

de 1 cm (incluido) hasta 3 cm (no incluido)	→	pequeño
de 3 cm (incluido) hasta 5 cm (no incluido)	→	mediano
de 5 cm (incluido) hasta 6.5 cm (no incluido)	→	grande
de 6.5 cm (incluido) hasta 8.5 cm (no incluido)	→	muy grande

5.9 Amplia el ejercicio 4.3 para que además de encontrar el punto medio de dos puntos del espacio bidimensional, escriba por pantalla a qué cuadrante del plano pertenece dicho punto medio.



Ejercicios Adicionales

5.10 Observa el siguiente programa en C llamado *ex_5_10*, y di qué realiza:

```
#include <stdio.h>

int main(void) {
    int a, b, c, r;

    printf ("Dame tres valores enteros: ");
    scanf ("%d %d %d", &a, &b, &c);

    r = (a < b) ? ((a < c) ? a : c) : ((b < c) ? b : c);

    printf ("Resultado: %d\n", r);

    return 0;
}
```

5.11 Crea un programa llamado *ex_5_11*, que pida los coeficientes *a*, *b* y *c* de una ecuación de segundo grado y calcule la solución.

$$a x^2 + b x + c = 0$$

La formula matemática que resuelve esta ecuación es la siguiente:

$$x = \frac{-b \pm \sqrt{b^2 - 4ac}}{2a}, \text{ es decir, hay dos soluciones } x_1 = \frac{-b + \sqrt{b^2 - 4ac}}{2a} \text{ y } x_2 = \frac{-b - \sqrt{b^2 - 4ac}}{2a}$$

Ten en cuenta los siguientes casos especiales en la resolución:

- Si $a = 0$ la ecuación es de primer grado, pero se puede calcular el resultado utilizando el algoritmo del ejercicio 5.7.
- Si $b^2 - 4ac < 0$ las raíces son imaginarias, pero se puede mostrar el resultado separando la parte real de la imaginaria., o bien decir que no tiene resultados reales.

(En C, para calcular la raíz cuadrada podemos utilizar la función *sqrt()*, y para elevar al cuadrado basta multiplicar un número por el mismo).

5.12 Crea un programa llamado *ex_5_12*, que pida una fecha formada por tres valores numéricos (día, mes y año), y determine si la fecha corresponde a un valor válido.

Pista: se debe tener presente el valor de los días en función de los meses y de los años. Es decir:

- Los meses 1, 3, 5, 7, 8, 10 y 12 tienen 31 días.
- Los meses 4, 6, 9 y 11 tienen 30 días.
- El mes 2 tiene 28 días, excepto cuando el año es divisible por 4, que tiene 29 días.

Práctica 6: Estructuras de control iterativas

Objetivos de la práctica

- Trabajo con la estructura de control iterativa de condición inicial: *while* ...
- Trabajo con la estructura de control iterativa de condición final: *do ... while* ...
- Trabajo con la estructura de control iterativa repetitiva: *for* ...

Repaso previo

- http://es.wikibooks.org/wiki/Programación_en_C/Instrucciones_de_control

Ejercicios Obligatorios

6.1 Observa el siguiente programa en C llamado *ex_6_1*, que calcula la media entre una serie de valores que el usuario introducirá por teclado hasta que finalmente introduzca el valor 0:

```
#include <stdio.h>

int main(void) {

    float x, media, suma;
    int num;

    printf("Cálculo de la media de una serie de datos\n");
    printf("-----\n");

    /* Inicializamos las variables */
    suma = 0;
    num = 0;

    /* Leemos datos y los procesamos hasta que introducen 0 */
    do {
        printf("Introduce un valor (0 para acabar) : ");
        scanf("%f", &x);
        suma = suma + x;
        num = num + 1;
    } while (x != 0);

    /* Como hemos contado el cero como valor, restamos 1 al
     * número de términos a la hora de calcular la media. */
    media = suma / (num-1);

    /* Damos los resultados */
    printf("La media de los elementos es %8.2f\n", media);

    return 0;
}
```

¿Qué hacen las instrucciones del tipo *variable = variable + expresión aritmética*?

¿Cómo se podría haber realizado con una estructura iterativa de condición inicial (*while*)?

¿Cómo se podría haber realizado con una estructura iterativa repetitiva (*for*)?

¿En qué caso especial el algoritmo del programa no funciona? ¿Cómo lo podrías solucionar?

- 6.2 Crea un programa llamado *ex_6_2*, que muestre los elementos de la siguiente serie, así como su suma ([problema de Basilea](#)):

$$Serie = \sum_{i=1}^n \frac{1}{i^2} = 1 + \frac{1}{2^2} + \frac{1}{3^2} + \dots + \frac{1}{n^2}$$

Diseña el algoritmo en las tres estructuras iterativas (condición inicial, final y repetitiva) y decide cuál estructura es la más apropiada para este caso.

- 6.3 Crea un programa llamado *ex_6_3*, que calcule el factorial de un número entero introducido por el usuario. El número introducido por el usuario debe ser más grande que 0 y más pequeño que 20. Si no fuera así, el programa debe pedirlo de nuevo tantas veces como sea necesario.

Recuerda que el factorial de un número entero es dicho número multiplicado por todos sus antecesores:

$$n! = n \times (n-1) \times (n-2) \times \dots \times 1$$

¡Cuidado con el tipo de datos número entero en C y sus valores máximos de representación!

- 6.4 Amplia el ejercicio 4.3 para que encuentre el centro de gravedad (el punto medio) de varios puntos del espacio bidimensional que el ordenador pedirá por teclado hasta que el usuario introduzca el punto origen de coordenadas (0,0). Para ello deberás acumular el valor de las coordenadas y al mismo tiempo contar el número de valores introducidos por el usuario.

Ejercicios Adicionales

- 6.5 Crea un programa llamado *ex_6_5*, en el que el usuario introduzca números enteros hasta adivinar el número aleatorio entre 0 y 100 generado al azar por el ordenador. El programa debe avisar si el número introducido por el usuario es más grande o más pequeño que el número generado aleatoriamente.

La instrucción de C que te permite generar un número aleatorio entre 0 y $n-1$, ambos incluidos, es `rand()%n`, y la instrucción de C que nos permite que los números aleatorios sean diferentes en cada nueva ejecución del programa es `srand(time(NULL))` o bien `srand(getpid())`

- 6.6 En la tienda de los hermanos Roque es tradición presentar las latas de conserva apiladas triangularmente: en el primer piso una lata, en el segundo piso dos latas, en el tercer piso tres, y así sucesivamente. Por ejemplo, seis latas se ponen así:

```
      *
     * *
    * * *
```

Los hermanos tienen grandes problemas para realizar los pedidos de latas, ya que no todo

número de latas se puede apilar triangularmente. Por ejemplo, 8 latas no se pueden apilar. Crea un programa llamado `ex_6_6`, en el que dado un número natural introducido por el usuario, comprueba si es adecuado para apilar.

- 6.7 Crea un programa llamado `ex_6_7` que calcule y visualice los elementos de la serie de Fibonacci. Esta serie se define de la siguiente manera:

$$\text{Fibonacci}(0) = 0$$

$$\text{Fibonacci}(1) = 1$$

$$\text{Fibonacci}(n) = \text{Fibonacci}(n-1) + \text{Fibonacci}(n-2)$$

El usuario tan solo introducirá el número de elementos que quiere visualizar.

- 6.8 Crea un programa llamado `ex_6_8` que pida al usuario dos números enteros a y b por teclado y devuelva el resultado de realizar su multiplicación mediante sumas. Es decir:

$$a \times b = a + a + a + \dots + a \quad (a \text{ sumado } b \text{ veces})$$

Ten en cuenta que tanto a como b pueden ser números negativos.

- 6.9 Los microprocesadores de las calculadoras realizan el cálculo de la mayoría de funciones matemáticas (sin, cos, etc.) mediante sus desarrollos de la [serie de Taylor](#), que tan solo contienen sumas, restas, multiplicaciones y divisiones. Por ejemplo:

$$\sin(x) = x - \frac{x^3}{3!} + \frac{x^5}{5!} - \frac{x^7}{7!} + \frac{x^9}{9!} + \dots$$

Crea un programa llamado `ex_6_9` que calcule el seno mediante su aproximación en serie de Taylor, parando el cálculo cuando el término calculado, en valor absoluto, sea más pequeño o igual que un valor de error ϵ introducido por el usuario o fijado por el programa: $\epsilon \leq |x^n / n!|$

- 6.10 Crea un programa llamado `ex_6_10` que calcule la raíz cuadrada de un número real positivo a introducido por el usuario. El cálculo se realizará mediante el desarrollo en serie,

$$x_1 = a$$

$$x_{n+1} = \frac{1}{2} \left(x_n + \frac{a}{x_n} \right)$$

y debe parar cuando dos aproximaciones sucesivas difieran en menos de un valor ϵ dado por el usuario o fijado por el programa.

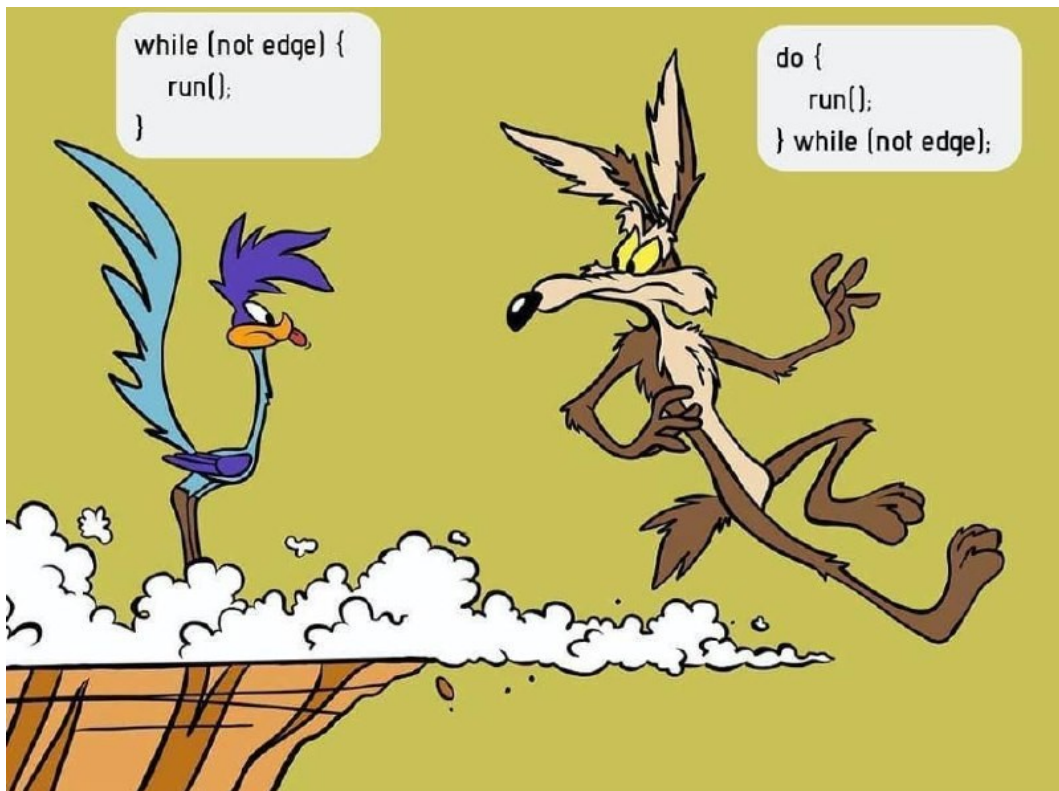
- 6.11 Crea un programa llamado `ex_6_11` que dibuje un triángulo rectángulo con asteriscos, a partir de un número entero introducido por el usuario que será el número de asteriscos de la base y el número de asteriscos de la altura. Por ejemplo, para el número 6 el programa debe imprimir:

```
*
**
***
****
*****
```

Pista: primero haz un programa que imprima una línea de asteriscos de una determinada longitud.

6.12 El teorema de Fermat expresa que no existen números naturales a , b y c tales que $a^n + b^n = c^n$ excepto para $n \leq 2$. Crea un programa llamado `ex_6_12` que lo compruebe para $n = 2$ y los primeros cien números naturales a y b .

¿Qué deberías cambiar a tu programa para que lo compruebe con cualquier n entre 3 y 10?



Práctica 7: Estructuras de almacenamiento homogéneas unidimensionales (vectores y strings)

Objetivos de la práctica

- Introducción al concepto de variable estructurada homogénea.
- Declaración de vectores y de cadenas de caracteres.
- Referencia directa e indirecta (indexada) de los elementos.
- Recorrido y operaciones sobre los elementos.

Repaso previo

- http://es.wikibooks.org/wiki/Programación_en_C/Vectores
- http://es.wikibooks.org/wiki/Programación_en_C/Cadenas_de_caracteres
- http://es.wikipedia.org/wiki/Algoritmo_de_ordenamiento

Ejercicios Obligatorios

- 7.1 Sin ayuda del ordenador, determina el valor de los vectores a y b en cada paso de la ejecución de las siguientes instrucciones:

```
int a[3], b[3], i;
a[0] = 2;
a[1] = 4;
a[2] = a[0] + a[1];
b[1 + 1] = a[0] - 1;
b[a[0]-1] = 2*a[1] - 1;
b[2.0*a[0]-a[1]] = b[0] + 1;
for (i = 0; i < 3; i++)
    b[i] = b[i] + i;
b[i] = 9;
```

Descubre el error semántico y los dos errores de concepto que esconde el programa.

- 7.2 Observa el siguiente programa en C llamado `ex_7_2`, que primero lee un vector de diez números reales y a continuación calcula su módulo, según la fórmula:

$$\text{sea } \vec{v} = (\vec{v}_1, \vec{v}_2, \dots, \vec{v}_n) \text{ entonces } |\vec{v}| = \sqrt{\vec{v}_1^2 + \vec{v}_2^2 + \dots + \vec{v}_n^2}$$

Compila el programa y realiza una depuración paso a paso con traza de variables para poder visualizar en todo momento el estado del vector.

```
#include <stdio.h>
#include <math.h>

#define MAX 10
```

```

int main(void) {

    float v[MAX];
    int i;
    float suma;

    printf("Cálculo del módulo de un vector\n");
    printf("-----\n\n");

    /* Llenamos el vector */
    for (i = 0; i < MAX; i++) {
        printf("Introduce el elemento %d : ", i);
        scanf("%f", &(v[i]));
    }

    /* Calculamos la suma de los cuadrados */
    suma = 0;
    for (i = 0; i < MAX; i++) {
        suma = suma + v[i]*v[i];
    }

    /* Imprimimos el módulo */
    printf("/nEl módulo del vector es %f\n", sqrt(suma));

    return 0;
}

```

¿Qué ventaja nos ofrece trabajar con una constante en este programa?

7.3 Crea un programa llamado *ex_7_3* para rellenar un vector de 15 número enteros:

- Con valores aleatorios entre 1 y 10, y a continuación diga cuantos pares e impares hay.
- Con valores aleatorios entre 1 y 10, y a continuación sume los que estén en posiciones que son múltiplos de 3.
- Con los primeros valores de la serie de Fibonacci.
- Con valores introducidos por el usuario, y a continuación que los imprima al revés.
- Con valores introducidos por el usuario, donde cada valor se debe pedir de nuevo hasta que esté entre 1 o 10.
- Con valores introducidos por el usuario, que deben formar una secuencia creciente.
- Con valores introducidos por el usuario, que no deben estar repetidos.

7.4 Crea un programa llamado *ex_7_4* que almacene en un vector la nota de los alumnos de un grupo de prácticas, y posteriormente calcule y visualice el número de notas que aparecen dentro de los siguientes intervalos:

[0 , 5[Insuficiente
[5 , 7[Aprobado
[7 , 9[Notable
[9 , 10]	Excelente

Tened en cuenta que, aunque los grupos de prácticas tienen un máximo de treinta alumnos,

cada grupo puede tener un número de alumnos diferente. El programa debe funcionar para cualquier grupo.

- 7.5 Crea un programa llamado *ex_7_5*, que dado un vector de 50 elementos enteros, lo descomponga en dos, uno formado por los valores pares y otro formado por los valores impares. En los dos vectores resultantes los valores se podrán consecutivamente, uno detrás del otro, sin huecos.
- 7.6 Crea un programa llamado *ex_7_6*, en el que el usuario introduzca una frase y el programa calcule y diga la longitud en caracteres ASCII de dicha frase.
- 7.7 Crea un programa llamado *ex_7_7*, en el que el usuario introduzca una frase y el programa invierta la frase y la imprima. Por ejemplo *abracadabra* invertida sería *arbadacarba*.

Ejercicios Adicionales

- 7.8 Crea un programa llamado *ex_7_8*, que dado un vector de 15 elementos con valores aleatorios, sea capaz de ordenar el vector y dar el resultado por pantalla.
- 7.9 Crea un programa llamado *ex_7_9*, que dados dos vectores ordenados realice la fusión de ambos para obtener un tercer vector también ordenado. Cada vector contiene cinco elementos.
- 7.10 Crea un programa llamado *ex_7_10*, en el que el usuario introduzca una frase y el programa calcule en número de palabras de dicha frase.
- Pista grande: para contar palabras podemos contar las veces que pasamos de un carácter que no es del alfabeto a uno que sí lo es. Para saber si un carácter es una letra, en C tenemos la función *isalpha(caracter)* de la librería *ctype.h*.
- 7.11 Crea un programa llamado *ex_7_11*, en el que el usuario introduzca una frase y una tabla de cifrado, es decir, una serie de caracteres y los correspondientes substitutos.
- El programa deberá cifrar la frase utilizando la tabla de cifrado, es decir, para cada carácter de la tabla a sustituir, buscará dicho carácter en la frase reemplazándolo por su sustituto.
- 7.12 Crea un programa llamado *ex_7_12*, que dado un número introducido en una base cualquiera *b1* sea capaz de convertirlo a otra base cualquiera *b2*. Como nos vemos limitados a la hora de trabajar con bases por la cantidad de símbolos de que disponemos, utilizaremos los símbolos 0, 1, ..., 9, A, B, ..., Z pudiendo trabajar así con bases hasta la base 36. El procedimiento puede ser el siguiente:
- (1) Pedimos las dos bases *b1* y *b2*.
 - (2) Leemos el número en base *b1* y lo convertimos a base 10 mediante el método de las potencias sucesivas.

(3) Convertimos el número de base 10 a base b_2 mediante el método de las divisiones sucesivas.

7.13 Observa el siguiente bloque de código. Debería iterar unas pocas veces, pero en lugar de ello puede entrar en un bucle infinito (depende del compilador utilizado). ¿Por qué?

```
int v[10];
int i;
for (i=0; i<12; i++) {
    printf("Inicializo el elemento %d\n", i);
    v[i] = 0;
}
```

Práctica 8: Estructuras de almacenamiento homogéneas multidimensionales (matrices)

Objetivos de la práctica

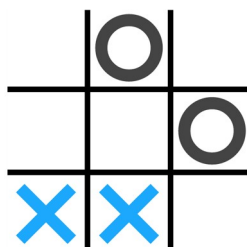
- Introducción al concepto de variable estructurada homogénea de dimensión mayor que uno.
- Declaración de matrices.
- Referencia directa e indirecta (indexada) de los elementos.
- Recorrido y operaciones sobre los elementos.

Repaso previo

- http://es.wikibooks.org/wiki/Programación_en_C/Vectores

Ejercicios Obligatorios

8.1 En el juego del tres en raya dos jugadores se turnan para colocar sus piezas, de una en una, sobre un tablero o matriz 3×3:



- a) Escribe el código que inicializa una matriz con las posiciones del tablero de la imagen.
- b) Escribe el código que dibuja dicho tablero, lo más similar posible.
- c) Escribe el código que pide dónde introducir la siguiente ficha, una X, comprobando que las coordenadas sean de una casilla vacía.
- d) Escribe el código que comprueba si dicha pieza introducida genera un tres en raya, bien sea horizontal, vertical, o diagonal.

8.2 Crea un programa llamado `ex_8_2`, que permita realizar la suma de dos matrices bidimensionales (utiliza un máximo de 10×10 elementos). La suma de matrices viene dada por la siguiente fórmula:

$$R[i][j] = M_1[i][j] + M_2[i][j]$$

Una vez escrito y compilado el programa, realiza una ejecución paso a paso con seguimiento de las variables para ver el funcionamiento de las iteraciones anidadas.

¿Qué cambiarías en el anterior algoritmo o programa para que en lugar de sumar matrices las multiplique? La multiplicación de matrices viene dada por la siguiente fórmula:

$$R[i][j] = \sum_{k=1}^n M_1[i][k] \times M_2[k][j]$$

8.3 Un/a alumno/a de informática desea realizar una estadística de las horas de estudio mensuales dedicadas a cada una de sus asignaturas. Crea un programa llamado *ex_8_3*, que dada la siguiente tabla nos permita calcular:

- El total anual de horas dedicadas a cada asignatura.
- El total mensual de horas dedicadas a estudiar.
- El nombre y el total de horas de la asignatura más estudiada.

	Enero	Febrero	...	Diciembre	TOTAL
Asignatura 1					
...					
Asignatura 5					
TOTAL					

Ejercicios Adicionales

8.4 Supón que dispones de la siguiente tabla de distancias kilométricas:

	Barcelona	Gerona	Lérida	Tarragona	Zaragoza	Teruel
Barcelona		100	156	98	296	409
Gerona			256	198	396	509
Lérida				91	140	319
Tarragona					231	311
Zaragoza						181
Teruel						

Crea un programa llamado *ex_8_4*, que permita calcular:

- La distancia entre dos poblaciones, el nombre de las cuales será introducido por el usuario.
- Las dos ciudades más alejadas entre sí y la distancia que las separa.
- La distancia total recorrida en el itinerario circular que pasa por todas las ciudades en el siguiente orden: primera, segunda, tercera, ..., última y primera de nuevo.

8.5 Una matriz “casi nula” es una matriz con un alto porcentaje de elementos nulos. Una matriz “casi nula” con k elementos no nulos se suele representar almacenando los elementos no nulos en una matriz de k filas y tres columnas, conteniendo cada columna de esta matriz la fila, la columna y el valor de los elementos no nulos, respectivamente. Por ejemplo:

$$\text{la matriz "casi nula"} \begin{pmatrix} 0 & 0 & 3 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \\ 0 & 6 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 \end{pmatrix} \text{ se puede representar por } \begin{pmatrix} 1 & 3 & 3 \\ 3 & 2 & 6 \\ 3 & 3 & 1 \\ 5 & 5 & 1 \end{pmatrix}$$

Crea un programa llamado *ex_8_5*, que convierte una matriz “casi nula” en representación

normal a la nueva representación más compacta.

- 8.6 Crea un programa llamado `ex_8_6`, que genere [cuadrados mágicos](#). El programa leerá un número natural ($2 < n < 11$), y calcule una matriz mágica de orden n . Una matriz de orden n (tamaño $n \times n$) se dice que es mágica si contiene los valores $1, 2, 3, \dots, n \times n$ y cumple la condición de que la suma de los valores almacenados en cada fila y columna coinciden. Por ejemplo, veamos la matriz mágica de orden 3 y la matriz mágica de orden 4.

8	1	6	15
3	5	7	15
4	9	2	15
15	15	15	

Para la construcción de la matriz mágica de orden impar se deben seguir la siguientes reglas:

- Colocamos el número 1 en la celda correspondiente al centro de la primera fila .
- Seguimos colocando los siguientes números avanzando una celda hacia arriba y a la derecha.
- Consideramos que la matriz cumple la propiedad de la circularidad, es decir, si salimos por la derecha se vuelve a entrar por la izquierda, y si se sale por arriba se entra por abajo.
- Si la celda donde corresponde el siguiente número de la lista está ya ocupada, entonces se coloca éste en la celda que haya debajo del último número colocado.

16	2	3	13	34
5	11	10	8	34
9	7	6	12	34
4	14	15	1	34
34	34	34	34	

Para la construcción de la matriz mágica de orden par se deben seguir la siguientes reglas:

- Colocamos los números $1, 2, 3, \dots, n \times n$ consecutivamente, de izquierda a derecha y de arriba a abajo.
- Invertimos los valores de la diagonal principal.
- Invertimos los valores de la segunda diagonal principal.

Práctica 9: Estructuras de almacenamiento heterogéneas (registros)

Objetivos de la práctica

- Introducción al concepto de variable estructurada heterogénea.
- Declaración de variables estructuradas heterogéneas.
- Declaración de nuevos tipos de datos.
- Acceso a los elementos (campos) de las variables estructuradas heterogéneas.

Repaso previo

- http://es.wikibooks.org/wiki/Programación_en_C/Estructuras_y_Uniones

Ejercicios Obligatorios

9.1 Diseña un tipo de datos para representar cada una de las siguientes entidades:

- a) Un punto del espacio tridimensional.
- b) Un píxel de una imagen en color (RGB).
- c) Una imagen en color, de dimensiones máximas 800×600, pero puede tener otras menores.
- d) Un número complejo (también llamado número imaginario).
- e) Una fecha.
- f) Una persona: nombre, fecha de nacimiento y teléfono de contacto.
- g) Una agenda con capacidad para guardar 100 personas.

Si los datos que componen un nuevo tipo estructurado son todos del mismo tipo (por ejemplo, el punto del espacio), ¿Cuándo es mejor utilizar un tipo estructurado homogéneo (vector) y cuando un tipo estructurado heterogéneo (registro)?

9.2 Observa el siguiente programa en C llamado `ex_9_2`, que permite almacenar los datos personales de los alumnos (máximo 20 alumnos). De cada uno de ellos guardaremos el nombre, el apellido y las notas obtenidas en cinco exámenes. El programa calculará y guardará la nota final como la media de las cinco anteriores.

La aplicación pide al usuario el número de alumnos con los que trabajará y los datos de cada uno de ellos. A continuación, calcula la nota final de todos los alumnos y visualiza el nombre y la nota final de todos los alumnos.

```
#include <stdio.h>

#define MAX_ALUM 20
#define MAX_NOTAS 5
```

```

struct t_alumne {
    char nombre[20];
    char apellido[30];
    float notas[MAX_NOTAS];
    float media;
};

struct t_classe {
    struct t_alumno alumnos[MAX_ALUM];
    int num_alum;
};

int main(void) {

    struct t_classe clase;
    int i, j;

    /* Pedir el número de alumnos */
    do {
        printf("¿Cuántos alumnos tenemos (máximo %d) ? ", MAX_ALUM);
        scanf("%d", &(clase.num_alum));
    } while ((clase.num_alum < 1) || (clase.num_alum > MAX_ALUM));

    /* Pedir los datos de los alumnos */
    for (i = 0; i < clase.num_alum; i++) {
        getchar();
        printf("\nIntroduce el nombre del alumno %d : ", i);
        gets(clase.alumnos[i].nombre);
        printf("Introduce el apellido del alumno %d : ", i);
        gets(clase.alumnos[i].apellido);
        for (j = 0; j < MAX_NOTAS; j++) {
            do {
                printf("Introduce la nota %d del alumno %d : ", j, i);
                scanf("%f", &(clase.alumnos[i].notas[j]));
            } while ((clase.alumnos[i].notas[j] < 0) ||
                (clase.alumnos[i].notas[j] > 10));
        }
    }

    /* Calcular la nota final de los alumnos */
    for (i = 0; i < clase.num_alum; i++) {
        clase.alumnos[i].media = 0;
        for (j = 0; j < MAX_NOTAS; j++) {
            clase.alumnos[i].media = clase.alumnos[i].media +
                clase.alumnos[i].notas[j];
        }
        clase.alumnos[i].media = clase.alumnos[i].media / MAX_NOTAS;
    }

    /* Escribir el resultado de la nota final de cada alumno */
    printf("\nInforme de notas finales\n");
    printf("-----\n");
    for (i = 0; i < clase.num_alum; i++) {
        printf("%s %s : %4.2f\n", clase.alumnos[i].nombre,
            clase.alumnos[i].apellido, clase.alumnos[i].media);
    }

    return 0;
}

```

¿Cambia alguna instrucción en un programa por el hecho de realizarlo con registros?

9.3 Crea un programa llamado *ex_9_3*, que amplíe el ejercicio 8.2 para que permita realizar multiplicaciones de matrices no cuadradas, utilizando para almacenar la información de cada matriz un registro que contendrá los siguientes campos:

- Una matriz de 10×10 elementos.
- Una variable que indique el número de filas reales de la matriz.
- Una variable que indique el número de columnas reales de la matriz.

```
struct t_matriz {  
    float valores[10][10];  
    int fil;  
    int col;  
}
```

Las matrices pueden tener un número de filas diferente al de las columnas, con una dimensión máxima de 10 filas por 10 columnas. Ten en cuenta que para poder multiplicar dos matrices, *A* y *B*, se ha de cumplir la siguiente condición:

número de columnas de *A* = número de filas de *B*

y que la matriz resultante, *R*, cumple:

número de filas de *R* = número de filas de *A*

número de columnas de *R* = número de columnas de *B*

9.4 Crea un programa llamado *ex_9_4*, que amplíe el ejercicio 6.4 que calculaba el centro de gravedad de un conjunto de puntos del espacio bidimensional. Ahora los puntos se deben almacenar en un vector, y cada punto es una estructura formada por sus coordenadas:

t_punto	x	y
---------	---	---

Ejercicios Adicionales

9.5 Crea un programa llamado *ex_9_5*, que permita sumar y multiplicar dos polinomios. Dichos polinomios los implementaremos como un registro donde guardaremos:

- El grado del polinomio. Limitaremos el grado máximo a 10.
- Los coeficientes del polinomio.

	grado	coeficientes									
t_polinomio		0	1	2	3	4	5	6	7	8	9

Te hará falta recordar las normas de suma y multiplicación de polinomios. Te recomiendo hacer un ejemplo de cada para ver la mecánica de trabajo. Por ejemplo:

Polinomio 1: $2x^3 + 3x^2 - 2$	3	-2	0	3	2						
--------------------------------	---	----	---	---	---	--	--	--	--	--	--

Polinomio 2: $-2x^2 + x - 3$	2	-3	1	-2							
------------------------------	---	----	---	----	--	--	--	--	--	--	--

Suma: $2x^3 + x^2 + x - 5$

3	-5	1	1	2					
---	----	---	---	---	--	--	--	--	--

Mult: $-4x^5 - 4x^4 - 3x^3 - 5x^2 - 2x + 6$

5	6	-2	-5	-3	-4	-4			
---	---	----	----	----	----	----	--	--	--

9.6 Nuestra lista de la compra está formada por la lista de productos a comprar (máximo 100) y el número de productos a comprar. De cada producto guardamos: nombre, precio por unidad, y unidades a comprar.

Crea un programa llamado *ex_9_6*, con un menú que permita:

1. Añadir productos (avisa si el producto existe)
 2. Quitar productos (avisa si el producto no existe)
 3. Listar productos (imprime el total a pagar)
 4. Ordenar lista por dinero gastado en producto (opcional)
 0. Finalizar.

Para resolver el programa, sigue estos pasos o consejos:

(0) Primero escribe el “esqueleto” del programa: dibujar el menú, pedir la opción, el switch-case vacío de código, y todo esto envuelto por el bucle que se repite mientras la opción sea diferente de finalizar.

(1) Escribe el apartado de añadir sin comprobar que el producto esté repetido.

(2) Escribe el apartado de listar.

(3) Comprueba que lo que llevas hecho funciona listando un producto que hayas añadido.

(4) Comprueba que funciona en el caso en que, después de haber añadido un producto, en el menú seleccionas añadir otro producto. Al listar deberían aparecer los dos.

(5) Comprueba que funciona en el caso en que quieras añadir un producto y la lista esté llena. Para facilitar la prueba pon el límite de productos de la lista de la compra a 2 ó 3.

(6) Cuando añadir ya funcione, implementa la funcionalidad de comprobar si el ítem está repetido. Decide qué quieres hacer en caso de que el ítem esté repetido. ¿Tan solo avisar al usuario? ¿O mejor pedir los nuevos datos para modificar el ítem que ya existía?

(7) Escribe el apartado de sacar, pero de momento sin buscar por el nombre. Únicamente pidiendo el número de casilla del producto.

(8) Puedes escoger dos maneras de borrar: “marcando” el ítem como borrado, o desplazar los siguientes ítem una posición hacia atrás. Si habías escogido borrar escribiendo un código especial en el ítem, recuerda que tendrás que modificar el apartado de listar para que no liste los ítem con dicho código, y el apartado de añadir para que utilice las posiciones de registros marcados como borrados para nuevos ítem.

(9) Comprueba que funciona en todos estos casos:

- Borrar el primer elemento de una lista de varios ítem.
- Borrar el último elemento de una lista de varios ítem.
- Borrar un elemento del medio de una lista de varios ítem.
- Borrar el único elemento de una lista de un ítem.

(10) Modifica el apartado de sacar para que busque por el nombre. Comprueba que funciona

sacando un ítem que existe, y también intentando sacar uno que no existe.

- 9.7 Pregunta complicada: una estructura está formada por nueve chars, que ocupa cada uno un byte, y un float, que ocupa cuatro bytes. En total la estructura debería ocupar trece bytes. Sin embargo, ocupa 16 bytes. ¿A qué es debido?

```
struct t_alumno {
    char dni[9];
    float nota;
};

struct t_alumno a;

printf("dni: %ld bytes\n", sizeof(a.dni));
printf("nota: %ld bytes\n", sizeof(a.nota));
printf("t_alumno: %ld bytes\n", sizeof(a));
```

Práctica 10: Funciones y modularidad

Objetivos de la práctica

- Definición y concepto de subprograma
- Paso de parámetros por valor y por referencia.
- Diseño descendente de una aplicación.
- Introducción al concepto de recursividad.

Repaso previo

- http://es.wikibooks.org/wiki/Programación_en_C/Use_de_funciones
- <http://elvex.ugr.es/decsai/c/apuntes/recursividad.pdf>
- www.lcc.uma.es/~lopez/modular/recursion/transp_recursion.pdf

Ejercicios Obligatorios de Funciones

10.1 Observa el siguiente programa:

```
#include <stdio.h>

int main(void) {
    int i, j;

    for (i = 1; i <= 5; i++) {
        for (j = 1; j <= 5; j++) {
            printf("*");
        }
        printf("\n");
    }

    printf("Hola\n");

    for (i = 1; i <= 5; i++) {
        for (j = 1; j <= 5; j++) {
            printf("*");
        }
        printf("\n");
    }

    return 0;
}
```

Vamos a mover del programa principal el código que dibuja un recuadro a un subprograma.
¿Qué ventajas crees que tiene trabajar con subprogramas sobre el anterior programa?

```
#include <stdio.h>

void recuadro() {
    int i, j;
```

```

    for (i = 1; i <= 5; i++) {
        for (j = 1; j <= 5; j++) {
            printf("*");
        }
        printf("\n");
    }
}

int main(void) {
    recuadro();
    printf("Hola\n");
    recuadro();
    return 0;
}

```

Ahora vamos a hacer que el programa principal le pase información al subprograma a través de un par de variables globales compartidas. ¿Qué ventajas tenemos sobre el programa anterior si se pueden enviar información? ¿Se pueden compartir variables locales?

```

#include <stdio.h>

int n_fil, n_col;

void recuadro() {
    int i, j;
    for (i = 1; i <= n_fil; i++) {
        for (j = 1; j <= n_col; j++) {
            printf("*");
        }
        printf("\n");
    }
}

int main(void) {
    n_fil = 3;
    n_col = 6;
    recuadro();
    printf ("Hola\n");
    n_fil = 5;
    n_col = 2;
    recuadro();
    return 0;
}

```

Ahora vamos a hacer que el programa principal le pase información al subprograma a través de parámetros de entrada. ¿Qué ventajas tenemos sobre el programa anterior? ¿Es el parámetro una variable global o local al subprograma?

```

#include <stdio.h>

void recuadro(int n_fil, int n_col) {
    int i, j;
    for (i = 1; i <= n_fil; i++) {
        for (j = 1; j <= n_col; j++) {
            printf("*");
        }
        printf("\n");
    }
}

```

```
int main(void) {
    recuadro(3, 6);
    printf("Hola\n");
    recuadro(5, 2);
    return 0;
}
```

Por último, vamos a mover el subprograma a un nuevo fichero llamado cuad.c. ¿Qué ganamos con ello?

```
#include <stdio.h>
#include "cuad.c"

int main(void) {
    recuadro(3, 6);
    printf("Hola\n");
    recuadro(5, 2);
    return 0;
}
```

10.2 Observa el siguiente subprograma:

```
#include <stdio.h>

void de_horas_a_segundos() {
    float horas, segundos;
    printf("¿Cuántas horas? ");
    scanf("%f", &horas);
    segundos = horas*3600;
    printf("En segundos son %f\n", segundos);
}

int main(void) {
    de_horas_a_segundos();
    return 0;
}
```

Ahora vamos a quitar del subprograma la entrada por teclado y salida por consola. ¿Qué ganamos si la entrada de información al subprograma es mediante parámetros de entrada y la salida mediante un valor de retorno?

```
#include <stdio.h>

float de_horas_a_segundos(float horas) {
    float segundos;
    segundos = horas*3600;
    return segundos;
}

int main(void) {
    float dias, seg;
    printf("¿Cuántos días? ");
    scanf("%f", &dias);
    seg = de_horas_a_segundos(dias*24);
    printf("Los segundos son %f\n", seg);
    return 0;
}
```

10.3 Escribe los subprogramas o funciones correspondientes a las siguientes especificaciones:

- a) Un subprograma tal que, dados dos reales a y b , calcula el logaritmo en base b de a según la siguiente fórmula: $\log_b a = (\log a) / (\log b)$.
- b) Un subprograma tal que, dado un número real, calcula el signo del número y devuelva -1, 0 ó 1 según si el número es negativo, cero o positivo.
- c) Un subprograma tal que, dado un carácter, calcule si el carácter en cuestión es una letra o no.
- d) Un subprograma tal que, dada una fecha formada por día, mes y año, devuelva 0 ó 1 según sea correcta o no. Reutiliza el código del ejercicio 5.12.
- e) Un subprograma tal que, dados dos números reales, retorna un dato leído por teclado que se encuentra dentro de los límites marcados por dichos valores. Si el número introducido no está entre el mínimo y máximo marcado por los parámetros, debe continuar pidiéndolo.
- f) Un subprograma tal que, dados dos vectores de 10 elementos, retorna el vector resultante de sumar los dos anteriores.
- g) Un subprograma tal que, dada una matriz 3×3 , retorna el valor de su determinante según la [regla de Sarrus](#):

$$\det \begin{pmatrix} a_{11} & a_{12} & a_{13} \\ a_{21} & a_{22} & a_{23} \\ a_{31} & a_{32} & a_{33} \end{pmatrix} = (a_{11}a_{22}a_{33} + a_{12}a_{23}a_{31} + a_{13}a_{21}a_{32}) - (a_{13}a_{22}a_{31} + a_{12}a_{21}a_{33} + a_{11}a_{23}a_{32})$$

- h) Un subprograma tal que, dado un polinomio, retorna su derivada. Utiliza el tipo `t_polinomio` definido en el ejercicio 9.5.
- i) (Opcional) Un subprograma tal que, dado un número entero, calcula recursivamente su factorial (dicho número entero multiplicado por todos sus antecesores):

$$n! = n \times (n-1) \times (n-2) \times \dots \times 1 = n \times (n-1)! \text{ si } n > 1 \text{ y } 1 \text{ si } n=1 \text{ ó } n=0$$

10.4 Observa el siguiente subprograma. ¿Qué prevés que imprimirá? Haz un seguimiento paso a paso dibujando en un papel todas las variables.

```
#include <stdio.h>

void intercambia(float a, float b) {
    float aux = a;
    a = b;
    b = aux;
}

int main(void) {
    float x = 3, y = 5;
    intercambia(x, y);
    printf("%f %f\n", x, y);
    return 0;
}
```

¿Qué significan los cambios que vamos a hacer en el siguiente programa? ¿Qué prevés que imprimirá? ¿Qué es un parámetro de entrada y que es un parámetro de salida? ¿Qué relación tienen con los parámetros de salida los operadores `&` y `*` en lenguaje C?

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

```

void intercambia(float *a, float *b) {
    float aux = *a;
    *a = *b;
    *b = aux;
}

int main(void) {
    float x = 3, y = 5;
    intercambia(&x, &y);
    printf("%f %f\n", x, y);
    return 0;
}

```

10.5 Observa el siguiente subprograma. ¿Qué valor prevés para las variables *x* e *y* después de llamar al procedimiento *prueba*? Haz un seguimiento paso a paso dibujando en un papel todas las variables.

```

#include <stdio.h>

int x, y;

void prueba(int n1, int *n2) {
    int x;
    x = *n2;
    y = 5*x*x;
    n1 = y;
    *n2 = 4;
}

int main(void) {
    x = 25;
    y = 2;
    printf("Antes      x = %d    y = %d\n", x, y);
    prueba(x, &y);
    printf("Después    x = %d    y = %d\n", x, y);
    return 0;
}

```

¿Qué diferencia hay entre paso de parámetros por valor y por referencia?

¿Cuál sería el valor para las variables *x* e *y* después de llamar al procedimiento *prueba* si fuera *n1* el parámetro por referencia, en lugar de *n2*?

¿Cuál sería el valor para las variables *x* e *y* después de llamar al procedimiento *prueba* si no existiera la declaración de la variable local *x* dentro del subprograma?

10.6 Crea un programa llamado *ex_10_6*, que amplíe el ejercicio 9.3 que multiplica (o suma) matrices. La definición de las matrices será el mismo registro. Sólo hará falta definir tres subprogramas:

- Una función que lea por teclado los elementos de una estructura de tipo *t_matriz*, pasada como parámetro:

```
void leer_matriz(struct t_matriz *M);
```

- Una función que imprima por pantalla los elementos de una estructura de tipo *t_matriz*, pasada como parámetro:

```
void ver_matriz(struct t_matriz M);
```

- Una función que calcule el producto de dos matrices pasadas como parámetro, y deje el resultado en una tercera matriz también pasada como parámetro, en el caso que sea posible:

```
int mult_matriz(struct t_matriz M1, struct t_matriz M2, struct t_matriz *R);
```

La función para multiplicar matrices intentará calcular $R = M1 \times M2$ y, si el cálculo es posible, devolverá en R el valor de la matriz producto y el valor de retorno de la función será cierto (ó 1). Si el cálculo no es posible el valor de retorno de la función será falso (ó 0).

¿Cuál es la mejor manera de pasar una matriz o un registro como parámetro de un subprograma: por valor o por referencia? ¿Por qué?

10.7 Observa el siguiente subprograma. ¿Qué hace? ¿Cómo puedes probarlo?

```
#include <stdio.h>

int main(int argc, char* argv[]) {

    int i;

    printf("%d argumentos:\n", argc);

    for (i = 0; i < argc; i++) {
        printf("  %s\n", argv[i]);
    }

    return 0;
}
```

Ejercicios Obligatorios de Recursividad

“Para entender la recursividad, antes hay que entender la recursividad.” - Anónimo

Recordatorio previo a problemas de recursividad. Algunos de los problemas más habituales de los métodos recursivos son los siguientes:

- No finalización del método: puede ocurrir que siempre, o sólo en el caso de determinados parámetros de entrada, el método recursivo no alcance nunca un caso no recursivo y, por lo tanto, no finalice.
- Desbordamiento de pila: cuanto mayor sea el número de invocaciones anidadas en un momento dado, mayor es el tamaño que ocupa la pila, porque necesita almacenar más datos. Existe el riesgo de que, para determinados parámetros, se sobrepase el tamaño máximo de esta pila, lo cual desemboca en una cancelación abrupta del programa (“stack overflow”).
- Cálculo repetido de los mismos datos: un algoritmo recursivo mal programado puede dar lugar a que se realice muchas veces el mismo cálculo, provocando que el número de invocaciones recursivas realizadas crezca exponencialmente. Por ejemplo, la sucesión de Fibonacci.

10.8 Observa la siguiente función que utiliza recursividad. ¿Cuál es el resultado si llamo *metodo(6)*? Dibuja las llamadas.

```
void metodo(int n) {
    if (n<2)
```

```

    printf("X");
else {
    metodo(n-1);
    printf("0");
}
}

```

10.9 Observa la siguiente función que utiliza recursividad. ¿Cuál es el resultado si llamo *metodo1(6)*? ¿Y si llamo al procedimiento *metodo1(7)*? Dibuja las llamadas.

```

void metodo1(int n) {
    if (n == 0)
        printf("En metodo 1 con N: %d\n", n);
    else
        metodo2(n);
}

void metodo2(int n) {
    printf("En metodo 2 con N: %d\n", n);
    metodo3(n-1);
}

void metodo3(int n) {
    printf("En metodo 3 con N: %d\n", n);
    metodo1(n-1);
}

```

10.10 Determina qué calcula la siguiente función recursiva. Escribe una función iterativa que realice la misma tarea.

```

int func(int n) {
    if (n == 0)
        return(0);
    else
        return( n+func(n-1) );
}

```

10.11 Programa una función recursiva y otra iterativa para calcular el máximo común divisor de dos números enteros aplicando las siguientes propiedades recurrentes:

$$\text{mcd}(a, b) = \text{mcd}(a-b, b) \quad \text{si } a > b$$

$$\text{mcd}(a, b) = \text{mcd}(a, b-a) \quad \text{si } a < b$$

$$\text{mcd}(a, b) = a \quad \text{si } a = b$$

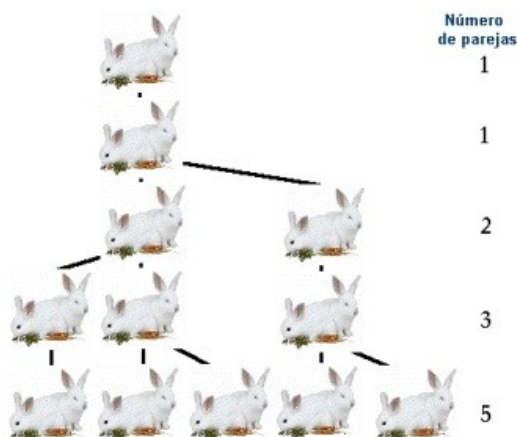
10.12 Calcula el número de llamadas que se generan para calcular el número de Fibonacci mediante el algoritmo recursivo básico, para un cierto número n , pasado por parámetro. Programa el método recursivo que, para evitar cálculos repetidos, aceptase como parámetro un vector de resultados.

$$\text{Fib}(n) = \text{Fib}(n-1) + \text{Fib}(n-2), \quad \text{para } n > 1$$

$$\text{Fib}(n) = 1, \quad \text{para } n = 1$$

$$\text{Fib}(n) = 0, \quad \text{para } n = 0$$

La famosa serie de Fibonacci responde a la siguiente pregunta: “Una pareja de conejos tarda un mes en alcanzar la edad fértil, a partir de ese momento cada vez engendra una pareja de conejos, que a su vez, tras ser fértiles engendrarán cada mes una pareja de conejos. ¿Cuántos conejos habrá al cabo de un determinado número de meses?”.



10.13 Programa un método recursivo que transforme un número entero positivo de base 10 a base 2.

10.14 Programa un método recursivo que transforme un número entero positivo de base 2 a base 10.

10.15 Programa un método recursivo para calcular la integral de una función (por ejemplo, el $\sin(x)$) en un intervalo dado, dividiendo este intervalo en subintervalos de longitud no menor que un determinado valor e :

$$\text{si } b-a \geq e \quad \int_a^b f(x) dx = \int_a^m f(x) dx + \int_m^b f(x) dx \quad \text{donde } m = \frac{a+b}{2}$$

$$\text{si } b-a < e \quad \int_a^b f(x) dx \simeq (b-a)f(m) \quad \text{donde } m = \frac{a+b}{2}$$

10.16 Programa un método recursivo que calcule la suma de un vector de números enteros.

10.17 Programa un método recursivo que invierta el orden de un vector de números enteros.

10.18 Programa un método que realice búsqueda binaria recursivamente en un vector de números enteros ordenado de menor a mayor.

Ejercicios Adicionales

10.19 Crea un programa llamado `ex_10_19`, que contenga el código de las siguientes funciones para trabajar con puntos del espacio bidimensional definidos por el tipo de datos:

```
struct t_punto {
    float x;
```

```
float y;
}
```

- Una función que lee un punto por teclado, y otra función que lo imprime en pantalla.
- Una función que transforma un punto (x, y) en otro $(a \cdot x, a \cdot y)$, donde a es un número real que indica el factor de escala.
- Una función que desplaza un punto (x, y) hacia $(x+a, y+b)$, donde (a, b) es otro punto que indica el desplazamiento.
- Una función que rota un punto (x, y) hasta $(x \cdot \cos \theta - y \cdot \sin \theta, x \cdot \sin \theta + y \cdot \cos \theta)$, donde θ es un número real que indica el ángulo en radianes.

Escoge como deben devolver las funciones el punto resultado, si en su valor de retorno, o en un parámetro del tipo punto pasado por referencia.

10.20 Observa el siguiente programa:

```
#include "vectores_lib.c"

int main(void) {

    float v[5], w[5];

    pide_vector(v, 5);
    imprime_vector(v, 5);

    rellena_vector(w, 5, -10, 10);
    imprime_vector(w, 5);

    suma_vectores(v, w, v, 5);
    imprime_vector(v, 5);

    ordena_vector(v, 5);
    imprime_vector(v, 5);

    return 0;
}
```

Crea la biblioteca *vectores_lib.c* que contenga el código de las siguientes funciones:

- Una función que inicializa con valores aleatorios los elementos de un vector:
`rellena_vector(int vector[], int longitud, int min, int max);`
- Una función que lee un vector, y otra que lo imprime:
`pide_vector(int vector[], int longitud);`
`imprime_vector(int vector[], int longitud);`
- Y diversas funciones que realicen todo tipo de operaciones sobre los vectores:
`suma_vectores, producto_vectores, modulo_vector, ordena_vector, media_vector, ...`

10.21 Crea un programa llamado *ex_10_21*, que permita resolver sistemas de ecuaciones lineales 3×3 mediante la [regla de Cramer](#). Dicho método utiliza el cálculo de determinantes 3×3 , para lo que puedes aprovechar el subprograma 10.3f.

10.22 Sea n un número entero de cuatro cifras diferentes. Define las funciones:

$\text{grande}(n)$ como el número más grande que se puede formar con las cifras de n

$\text{peque}(n)$ como el número más pequeño que se puede formar con las cifras de n

$\text{dif}(n) = \text{grande}(n) - \text{peque}(n)$

Por ejemplo, si tenemos $n = 1984$, entonces $\text{grande}(n) = 9841$, $\text{peque}(n) = 1489$ y $\text{dif}(n) = 8352$.

Crea un programa llamado *ex_10_22*, que dado un número entero n , compruebe que este tenga cuatro cifras diferentes y calcule y escriba los valores de la siguiente sucesión hasta que encuentre un término tal que $\text{dif}(n) = n$:

$\text{dif}(n)$

$\text{dif}(\text{dif}(n))$

$\text{dif}(\text{dif}(\text{dif}(n)))$

...

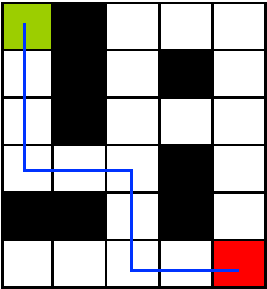
10.23 Programa el método recursivo de ordenación de vectores [quicksort](#).

10.24 Programa el método recursivo de ordenación de vectores [mergesort](#).

10.25 Imagina un laberinto formado por una matriz cuadrada de dimensiones $n \times m$, con k casillas marcadas que no se pueden atravesar. Crea un programa llamado *ex_10_25*, que permita encontrar el camino más corto para viajar de la esquina (1,1) a la esquina (n,m) sin pasar por las celdas marcadas.

El usuario introducirá los valores de n , de m y de k , así como las coordenadas de las k casillas marcadas. La salida será las coordenadas de las casillas que forman el camino. Ten en cuenta que puede no haber camino o haber más de uno.

Por ejemplo:

Laberinto	Entrada	Salida
	6 5 8	1 1 2 1 3 1 4 1 4 2 4 3 5 3 6 3 6 4 6 5

Práctica 11: Almacenamiento en ficheros

Objetivos de la práctica

- Introducción al concepto de fichero.
- Familiarizarse con el tipo de dato gestor del fichero o apuntador del fichero.
- Conocer las instrucciones básicas de acceso a un fichero.
- Aprender a trabajar con ficheros de texto y ficheros binarios.

Repaso previo

- http://es.wikibooks.org/wiki/Programación_en_C/Manejo_de_archivos

Ejercicios Obligatorios

11.1 Observa el siguiente programa que pide identificador de alumno y dos notas y escribe la media en pantalla:

```
#include <stdio.h>

int main(void) {

    char dni[10];
    float nota1, nota2;
    float media;
    int n;

    /* Pedir el número de alumnos */
    do {
        printf("¿Cuántos alumnos tenemos? ");
        scanf("%d", &n);
    } while (n < 1);

    while (n > 0) {

        /* Pedir los datos de los alumnos */
        printf("\nIntroduce el DNI (ocho números y letra) del alumno : ");
        scanf("%s", dni);
        printf("Dame sus notas : ");
        scanf("%f %f", &nota1, &nota2);

        /* Imprimir la media */
        media = (nota1 + nota2) / 2;
        printf("%s %f\n", dni, media);

        n--;
    }

    return 0;
}
```

Vamos a añadir unas pocas instrucciones para que en lugar de imprimir a pantalla, guarde la información en un fichero de texto. Puedes comprobar que funciona abriendo con un editor de textos el fichero de texto “notas.txt” que ha generado al ejecutarse.

```
#include <stdio.h>

int main(void) {

    FILE *f;

    char dni[10];
    float nota1, nota2;
    float media;
    int n;

    f = fopen("notas.txt", "wt");
    if (f == NULL) {
        printf("Error abriendo fichero\n");
        return 1;
    }

    /* Pedir el número de alumnos */
    do {
        printf("¿Cuántos alumnos tenemos? ");
        scanf("%d", &n);
    } while (n < 1);

    while (n > 0) {

        /* Pedir los datos de los alumnos */
        printf("\nIntroduce el DNI (ocho números y letra) del alumno : ");
        scanf("%s", dni);
        printf("Dame sus notas : ");
        scanf("%f %f", &nota1, &nota2);

        /* Imprimir la media */
        media = (nota1 + nota2) / 2;
        fprintf(f, "%s %f\n", dni, media);

        n--;
    }

    fclose(f);
    return 0;
}
```

Ahora observa el siguiente programa que lee la información del fichero y la imprime por pantalla. ¿Por qué crees que la última línea la imprime dos veces?

```
#include <stdio.h>

int main(void) {

    FILE *f;

    char dni[10];
    float media;

    f = fopen("notas.txt", "rt");
    if (f == NULL) {
```

```

    printf("Error abriendo fichero\n");
    return 1;
}

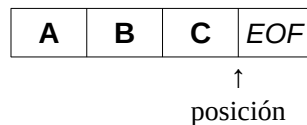
while (!feof(f)) {
    fscanf(f, "%s", dni);
    fscanf(f, "%f", &media);
    printf("%s %f\n", dni, media);
}

fclose(f);
return 0;
}

```

La explicación de porqué imprime el último dos veces tiene que ver con como se procesa la marca de final de fichero. La función *feof()* no devuelve cierto hasta que dicha marca se ha leído.

Imagina que en un fichero tienes tres registros o líneas (*A*, *B* y *C*) y a continuación la marca de final de fichero (*EOF*). Imagina además que hemos leído los tres registros y la posición actual en el fichero es justo delante de la marca de fin de fichero, tal como muestra la siguiente imagen.



Cabría pensar que hemos llegado a final de fichero y que la función *feof()* devolverá cierto. Pero, contra-intuitivamente, no es así. Vamos a ver qué está pasando y cómo solucionarlo.

El programa, que no tiene ojos como nosotros, no sabe si tiene delante la marca de final de fichero o si todavía quedan más registros. Por lo tanto la llamada a la función *feof()* devolverá falso. En el siguiente intento de operación de lectura será cuando lea dicha marca y la función *feof()* comience a devolver cierto.

Esto quiere decir que en el código anterior entra en el bucle una vez más de la esperadas, intenta leer un nuevo registro, se encuentra por fin con la marca de final de fichero y no puede leer el registro, por lo que la variable a leer continua teniendo los valores anteriores.

Podemos solucionarlo de varias maneras que ahora enumero. Escoge tu preferida. El siguiente código imprime por pantalla las líneas de un fichero de texto, pero imprime repetida la última línea.

```

while (!feof(f)) {
    fgets(linea, 49, f);
    printf("%s", linea);
}

```

La primera manera de solucionarlo es preguntar de nuevo después de la operación de lectura si hemos llegado a final de fichero. La desventaja es que en el bucle preguntamos dos veces la misma pregunta.

```

while (!feof(f)) {
    fgets(linea, 49, f);
    if (!feof(f))
        printf("%s", linea);
}

```

La segunda manera de solucionarlo es realizar un bucle infinito y romper la iteración desde dentro cuando sea final de fichero. La desventaja es que no seguimos los principios de la

programación estructurada.

```
while (1) {
    fgets(linea, 49, f);
    if (feof(f))
        break;
    printf("%s", linea);
}
```

La tercera manera consiste en aprovechar que las operaciones de lectura devuelven un valor que permite saber si dicha operación ha sido exitosa. Podemos combinar la operación de lectura con la condición del bucle. La desventaja es que el código se vuelve un poco más difícil de entender.

```
while (fgets(linea, 49, f) != NULL) {
    printf("%s", linea);
}
```

La cuarta manera consiste en leer una primera vez antes del bucle. Así el bucle comprueba si es final de fichero inmediatamente después de las lecturas.

```
fgets(linea, 49, f);
while (!feof(f)) {
    printf("%s", linea);
    fgets(linea, 49, f);
}
```

Intenta que el programa anterior deje de imprimir repetida la última línea, probando cada uno de los cuatro métodos que acabamos de ver.

- 11.2 Observa el siguiente programa que pide identificador de alumno y dos notas y escribe la media en un fichero binario “notas.bin”. Una vez se han acabado de introducir los datos abre el fichero en modo lectura e imprime por pantalla toda la información que contiene:

```
#include <stdio.h>

struct t_alumno {
    char dni[10];
    float media;
};

int main(void) {

    FILE *f;
    struct t_alumno alumno;
    float nota1, nota2;
    int n;

    /* Pedir el número de alumnos */
    do {
        printf("¿Cuántos alumnos tenemos? ");
        scanf("%d", &n);
    } while (n < 1);

    /* Abrir el fichero para escribir datos de alumnos */
    f = fopen("notas.bin", "wb");
    if (f == NULL) {
        printf("Error abriendo fichero\n");
        return 1;
    }
}
```

```

/* Pedir los datos de los alumnos y escribir dni y media */
while (n > 0) {
    printf("\nIntroduce el DNI (ocho números y letra) del alumno : ");
    scanf("%s", alumno.dni);
    printf("Dame sus notas : ");
    scanf("%f %f", &nota1, &nota2);
    alumno.media = (nota1 + nota2) / 2;
    fwrite(&alumno, sizeof(alumno), 1, f);
    n--;
}

/* Cerrar el fichero */
fclose(f);

/* Abrir el fichero para leer datos de alumnos */
f = fopen("notas.bin", "rb");
if (f == NULL) {
    printf("Error abriendo fichero\n");
    return 1;
}

/* Leer los datos de los alumnos y escribirlos en pantalla */
n = 0;
while( fread(&alumno, sizeof(alumno), 1, f) > 0 ) {
    printf("Alumno %d: %s %f\n", ++n, alumno.dni, alumno.media);
}

/* Cerrar el fichero */
fclose(f);

return 0;
}

```

11.3 Observa el siguiente programa que en el fichero binario “notas.bin” del ejercicio anterior busca un alumno por su campo nombre y visualiza por pantalla su nota:

```

#include <stdio.h>
#include <string.h>

struct t_alumno {
    char dni[10];
    float media;
};

int main(void) {

    FILE *f;
    struct t_alumno alumno;
    char dni[10];
    int encontrado;

    /* Pedir el dni del alumno a buscar */
    printf("¿Qué alumno buscas? ");
    scanf("%s", dni);

    /* Abrir el fichero para leer datos de alumnos */
    f = fopen("notas.bin", "rb");
    if (f == NULL) {

```



```

    printf("Error abriendo fichero\n");
    return 1;
}

/* Leer los datos de los alumnos hasta encontrar el buscado */
encontrado = 0;
while ( !feof(f) && (encontrado == 0) ) {
    fread(&alumno, sizeof(alumno), 1, f);
    if (strcmp (dni, alumno.dni) == 0) {
        encontrado = 1;
    }
}

if (encontrado == 0)
    printf("Alumno no encontrado\n");
else
    printf("%s : %f\n", alumno.dni, alumno.media);

/* Cerrar el fichero */
fclose(f);

return 0;
}

```

- 11.4 Observa el siguiente programa que en el fichero binario “notas.bin” del ejercicio anterior busca un alumno por su posición dentro del fichero (supondremos el primer alumno en la posición 1):

```

#include <stdio.h>
#include <string.h>

struct t_alumno {
    char dni[10];
    float media;
};

int main(void) {

    FILE *f;
    struct t_alumno alumno;
    int total, n;

    /* Abrir el fichero para leer datos de alumnos */
    f = fopen("notas.bin", "rb");
    if (f == NULL) {
        printf("Error abriendo fichero\n");
        return 1;
    }

    /* Calcula cuántos registros hay en el fichero */
    fseek(f, 0, SEEK_END);
    total = ftell(f) / sizeof(alumno);
    rewind(f);

    /* Pedir el alumno a buscar */
    do {
        printf("El fichero contiene %d alumnos ¿Qué alumno buscas? ", total);
        scanf("%d", &n);
    } while ((n < 1) || (n > total));
}

```

```

/* Leer los datos de los alumnos hasta encontrar el buscado */
fseek(f, (n-1)*sizeof(alumno), SEEK_SET);
fread(&alumno, sizeof(alumno), 1, f);
printf("%s : %f\n", alumno.dni, alumno.media);

/* Cerrar el fichero */
fclose(f);

return 0;
}

```

11.5 Crea un programa llamado *ex_11_5*, que pida el nombre de un fichero de texto, y reescriba el contenido de dicho fichero en otro fichero, convirtiendo todo el texto leído a mayúsculas.

11.6 Crea los siguientes programas:

- Un programa llamado *ex_11_6_a* que genere un fichero binario llamado “multiplos.dat” con los diez primeros múltiplos de 5.
- Un programa llamado *ex_11_6_b* que lee el contenido del fichero “multiplos.dat” y lo imprime por pantalla.
- Un programa llamado *ex_11_6_c* que añade al fichero “multiplos.dat” los diez siguientes múltiplos de 5. Para ello el programa deberá averiguar cuál fue el último número escrito.

11.7 Crea un programa llamado *ex_11_7*, que permita guardar en un fichero binario una determinada colección de información que tu escogerás: lista de la compra, agenda de personas, partidos de la liga, libros de tu biblioteca, etc.

Escoge y crea el tipo de datos registro que contendrá la información de los elementos de la colección. Recuerda que en el ejercicio 9.6 ya trabajábamos con una colección de elementos que era una lista de productos a comprar, por si quieres aprovechar dicho programa.

Habrà un menú para escoger qué hacer. Las opciones mínimas serán:

1. Añadir un registro.
2. Listar los registros.
3. Modificar un registro.
4. Borrar un registro.
5. Finalizar.

La primera opción permitirá abrir un fichero (o crear un fichero nuevo, en caso de que el fichero que indicado por el usuario no exista) y añadir nuevos registros.

La segunda opción permitirá abrir un fichero existente y listar todos los registros que contiene.

La tercera opción permitirá abrir un fichero existente y realizar una búsqueda de un determinado registro por el campo que lo identifica. Si lo encuentra, permitirá modificar en resto de campos del registro, retroceder en el fichero y sobrescribir con la nueva información.

La cuarta opción permitirá abrir un fichero existente y realizar una búsqueda de un determinado registro por el campo que lo identifique. Si lo encuentra, lo borrará.

Una manera de borrar, sencilla pero ineficiente, sería crear un segundo fichero donde iríamos copiando los registros del primer fichero excepto el que queremos borrar. Después borraríamos el primer fichero y cambiaremos el nombre del segundo fichero.

Una opción más eficiente sería marcar el registro que se ha borrado, bien porque hemos añadido un campo al registro para indicar si se ha borrado o no, o bien porque guardamos en algún lugar cuáles son las posiciones de los registros borrados. Pero si escogemos esta opción deberemos modificar las otras acciones de listar, añadir y modificar para que se adapten al hecho de que hayan registros no válidos (borrados) en el fichero.

Pero puedes añadir al menú nuevas opciones que creas convenientes. Utiliza subprogramas para implementar las diferentes opciones del menú.

Ejercicios Adicionales

11.8 Nos vamos a dedicar cantar hip-hop o a componer poemas a nuestra amada pareja. Para ello, debemos encontrar rimas. Dispondremos de un fichero de texto con todas las palabras del castellano, una por línea de fichero. Crea un programa llamado *ex_11_8* que busque en dicho fichero todas las palabras que acaban igual que un determinado sufijo introducido por el usuario. Por ejemplo, si el usuario introduce “ago”, posibles palabras serían: lago, murciélago, aciago, etc.

Se puede obtener listas de palabras del castellano (“lemario”), listas de nombres y apellidos, listas de verbos conjugados, aquí: <https://github.com/olea/lemarios/archive/master.zip>

También puedes obtener una lista de 56000 palabras del castellano escribiendo en el terminal:

```
aspell -l es dump master | cut -f1 -d/ | sort > castellano.txt
```

Si prefieres una lista más grande (12 millones de palabras) puedes escribir:

```
aspell -l es dump master | aspell -l es expand > extendido.txt
```

Una curiosidad: mucha gente cree que la única palabra del castellano que contiene las cinco vocales es murciélago. Sin embargo, hay docenas y docenas de palabras que contienen las cinco vocales. ¿Te atreves a escribir un programa que las encuentre en tu fichero? Ten en cuenta las vocales con acentos.

Apéndice: acceso a servidores de bases de datos

Nuestros programas también pueden guardar información en bases de datos. Para ello deben instalar las librerías del servidor escogido. Como ejemplo, vamos a estudiar un programa que se conecta a un servidor de bases de datos relacionales MySQL, lanza una consulta SQL y procesa la tabla resultado.

Para poder ejecutarlo hace falta tener instalado el conector de C a MySQL, llamado *libmysqlclient*, que se puede descargar en la siguiente página:

<http://dev.mysql.com/downloads/connector/c/>

Pero si estamos trabajando en Linux, basta con instalar el paquete *libmysqlclient-dev* des de los repositorios de nuestra distribución Linux.

Para compilar, debemos incluir en la línea de argumentos: `$(mysql_config --cflags)`

Para enlazar, debemos incluir en la línea de argumentos: `$(mysql_config --libs)`

Por ejemplo:

```
gcc -o fichero_salida $(mysql_config --cflags) fichero_entrada.c $(mysql_config --libs)
```

La lista de funciones que podemos utilizar se encuentra explicada en la siguiente página web:

<https://dev.mysql.com/doc/c-api/8.0/en/c-api-function-reference.html>

El siguiente programa establece una conexión con el servidor SQL, lista los nombres de las tablas de una base de datos, permite escoger una tabla al usuario, y lista el contenido de dicha tabla:

```
#include <mysql.h>
#include <stdio.h>
#include <stdlib.h>
#include <string.h>

int main() {

    MYSQL *conn;
    MYSQL_RES *res;
    MYSQL_ROW fila;

    conn = mysql_init(NULL);

    /* Conectar con el servidor de base de datos */

    if (!mysql_real_connect(conn, "URLservidor", "usuario", "contraseña",
"base_datos", 0, NULL, 0)) {
        printf("%s\n", mysql_error(conn));
        exit(1);
    }

    /* enviar una consulta: lista las tablas de la base de datos */

    if (mysql_query(conn, "show tables")) {
        printf("%s\n", mysql_error(conn));
        exit(1);
    }
    res = mysql_use_result(conn);

    /* procesar los resultados, línea a línea, escogiendo qué columna queremos
*/

    printf("Tablas en la base de datos:\n");
    while ((fila = mysql_fetch_row(res)) != NULL)
        printf("%s\n", fila[0]);

    /* Ahora vamos a generar una consulta a partir de datos que introduce el
usuario
    * En un string concatenaremos sentencia SQL y datos leídos del teclado
    * hasta obtener la sentencia final */

    char consulta[100] = "";
    char tabla[20];
    printf("¿Qué tabla quieres? ");
    scanf("%s", tabla);

    strcat(consulta, "SELECT * FROM ");
    strcat(consulta, tabla);

    printf("La consulta será: %s\n", consulta);

    /* enviar una consulta: lista el contenido de la tabla escogida por el
```

```

usuario */

if (mysql_query(conn, consulta)) {
    printf("%s\n", mysql_error(conn));
    exit(1);
}
res = mysql_use_result(conn);

/* procesar los resultados, línea a línea. Para cada línea, recorreremos
todas las celdas */

int i;
while ((fila = mysql_fetch_row(res)) != NULL) {
    for (i = 0; i < mysql_num_fields(res); i++) {
        printf("%s\t", fila[i]);
    }
    printf("\n");
}

/* cerrar la conexión */

mysql_free_result(res);
mysql_close(conn);

return 0;
}

```

Apéndice: acceso a servidores de directorio LDAP

Por último, nuestros programas también pueden recuperar o guardar información en servidores de directorio. Aquí tienes el enlace a dos páginas con varios ejemplos de programas que se conectan a un servidor OpenLDAP, lanzan consultas LDAP y procesan los resultados:

<http://www.linuxdevcenter.com/pub/a/linux/2003/08/14/libldap.html>

http://fm4dd.com/programming/c-code/ad_ldaptest_c.htm

Para compilarlos y ejecutarlos debes instalar los paquetes *libldap2-dev* y *libldap-2.4-2*. Y para enlazar, debes incluir en la línea de argumentos del compilador *-lldap -llber*. Por ejemplo:

```
gcc fichero_entrada.c -o fichero_salida -lldap -llber
```

La lista de funciones que podemos utilizar se encuentra explicada en la siguiente página web:

<http://www.openldap.org/software/man.cgi?query=ldap>

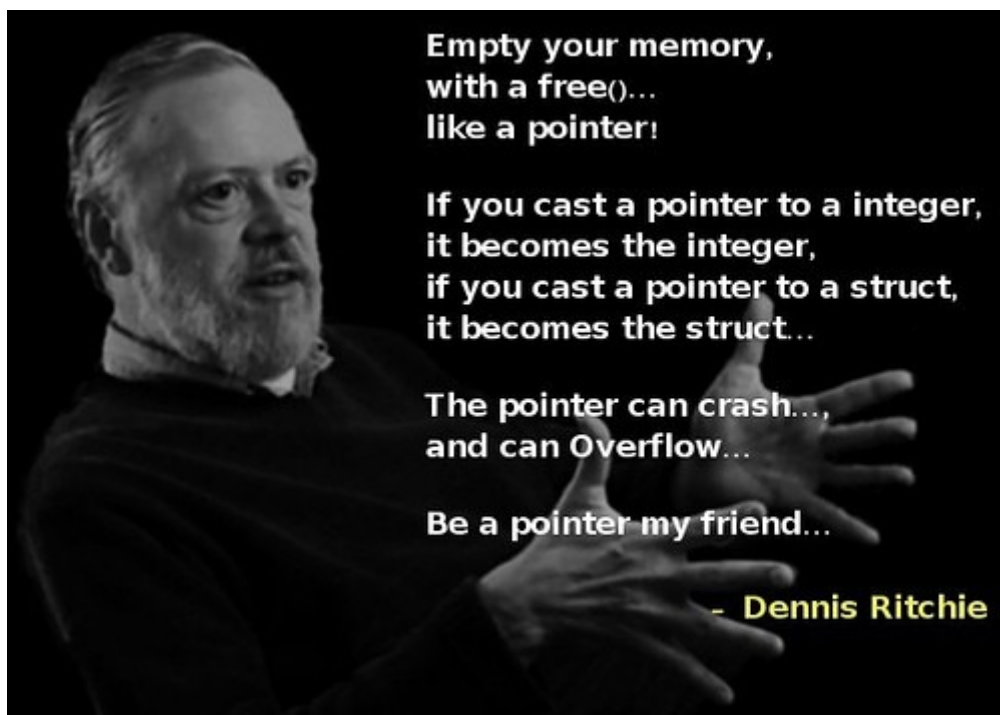
Práctica 12: Apuntadores y estructuras dinámicas de datos

Objetivos de la práctica

- Introducción al concepto de apuntador.
- Familiarizarse con los operadores de apuntadores: `&`, `*` y `->`.
- Conocer las instrucciones para reservar y liberar memoria dinámica: `malloc()` y `free()`.
- Familiarizarse con las estructuras dinámicas de datos más básicas: listas, pilas, colas y árboles.

Repaso previo

- http://es.wikibooks.org/wiki/Programación_en_C/Punteros
- http://es.wikibooks.org/wiki/Programación_en_C/Manejo_dinámico_de_memoria
- http://es.wikibooks.org/wiki/Programación_en_C/Matrices_Dinamicas
- http://www.inf-cr.uclm.es/www/cglez/downloads/docencia/punteros_c.pdf
- <http://www.ibm.com/developerworks/aix/library/au-toughgame/>
- http://www.tecnun.es/asignaturas/Informat1/AyudaInf/aprendainf/ansic/leng_c.pdf (tema 6)
- <http://c.conclase.net/edd/>
- <http://decsai.ugr.es/~jfv/ed1/tedi/cdrom/>



Ejercicios Obligatorios de Apuntadores

Recuerda que:

- El operador & delante del nombre de una variable significa “la dirección de memoria de ...”.
- El operador * delante de una variable o una expresión significa “el contenido de la dirección de memoria apuntada por ...”.

Por ejemplo, vamos a crear una variable y asignarle un valor de manera indirecta mediante un apuntador:

```
int a;      /* 'a' guardará un entero */
int *b;     /* 'b' guardará una dirección de memoria donde hay un entero */
...
a = 1;      /* asignación directa */
...
b = &a;     /* asignación indirecta */
*b = 2;     /* ahora 'a' vale 2 */
```

- La función *malloc* reserva posiciones de memoria y devuelve un apuntador a la memoria reservada, o *null* si no ha podido reservar dicha cantidad de memoria. Como el apuntador devuelto por *malloc* es a un tipo cualquiera (*void**), se debe hacer una pequeña transformación de tipo.
- La función *free* libera la memoria reservada para un apuntador mediante *malloc*. Toda memoria reservada dinámicamente se debe liberar antes de que acabe el programa. Una vez liberada dicha memoria, ya no se debe trabajar con ella.

Por ejemplo, vamos a crear dinámicamente un vector con capacidad para cinco reales, y cuando acabemos de trabajar con él liberaremos la memoria de dicho vector:

```
float *v;
...
v = (float *) malloc( 5*sizeof(float) );
...
free(v);
v[3] = 2;    /* ERROR memoria de 'v' liberada */
```

12.1 Dada la declaración de variables:

```
char *a, *b;
```

¿Qué hace la siguiente instrucción?

```
while (*a++ = *b++);
```

12.2 Dadas las siguientes declaraciones:

```
int x, m[5][5];
```

explica qué traduce el compilador para acceder al elemento de la fila tercera y columna cuarta de la matriz y asignárselo a x.

```
x = m[3][4];
```

12.3 ¿Qué diferencia hay entre las siguientes cuatro declaraciones de variables? (dibuja la memoria en los cuatro casos)

```
float v1[5] = {1.2, 3, -4, 0, 5.1};
float v2[5];
```

```
float v3[];  
float *v4;
```

12.4 Dadas las siguientes declaraciones:

```
int *ip, **ipp, (*ip4)[4], i, j;  
int ventas[3][4];
```

explica las siguientes cinco expresiones:

- a) `ip4 = ventas;`
- b) `ip = (int *) *ventas;`
- c) `ipp = (int **) ventas;`
- d) `*(*(ip4 + i) + j)`
- e) `*(*(ventas + i) + j)`

12.5 Dada la declaración de una radiografía (imagen de nivel de grises) que queremos procesar:

```
char radiografia[4000][4000];
```

¿Qué harían las siguientes instrucciones?

```
char *p = &(radiografia[0][0]);  
long i;  
for (i = 0; i < 16000000; i++) {  
    *p = (*p < 128 ? 0 : 255);  
    p++;  
}
```

¿Qué diferencia encuentras con recorrer la imagen de esta otra manera?

```
int i, j;  
for (i = 0; i < 4000; i++) {  
    for (j = 0; j < 4000; j++) {  
        radiografia[i][j] = (radiografia[i][j] < 128 ? 0 : 255);  
    }  
}
```

12.6 Explica el significado de las siguientes cuatro declaraciones:

- a) `int (*uno)[12];`
- b) `int *dos[12];`
- c) `int *fu();`
- d) `int (*fa)();`

12.7 ¿Qué diferencia hay entre paso de parámetros a una función por valor y por referencia? ¿Por qué la instrucción de lenguaje C *printf* recibe variables por valor y la instrucción *scanf* recibe variables por referencia?

```
int i;  
scanf("%d", &i);  
printf("Has introducido %d\n", i);
```


- 12.8 En el siguiente programa, ¿Cuál es el valor de las variables x e y después de llamar al procedimiento *prueba*? ¿Cuál sería el valor de las variables x e y después de llamar al procedimiento *prueba* si dicho procedimiento se modificara para que $n1$ se pasara por referencia y $n2$ por valor?

```
void prueba(int n1, int *n2) {
    int x, y;
    x = *n2;
    y = 5*x*x;
    n1 = y;
    *n2 = 4;
}

void main(void) {
    int x=25, y=2;
    printf("Antes      x = %d    y = %d\n", x, y);
    prueba(x, &y);
    printf("Después    x = %d    y = %d\n", x, y);
}
```

- 12.9 Escribe una función que recibidos dos números reales como argumento, al acabar la función haya intercambiado su valor. Ejemplo:

```
float a=2.5, b=3.1;
/* ahora 'a' vale 2.5 y 'b' vale 3.1 */
intercambia(&a, &b);
/* ahora 'a' vale 3.1 y 'b' vale 2.5 */
```

- 12.10 Escribir una función que determine si una cadena de caracteres recibida como parámetro está vacía o no.
- 12.11 Escribir una función que reciba como parámetro una cadena de caracteres y retorne su longitud
- 12.12 Escribe una función que reciba como parámetro una cadena de caracteres, devuelva cuantas palabras contiene.
- 12.13 Escribe una función que reciba como parámetro una cadena de caracteres y la devuelva invertida.
- 12.14 Escribe una función que reciba como parámetro dos cadenas de caracteres, y retorne la concatenación de una sobre la otra, terminando la cadena con el carácter '\0' (función *strcat()* de la biblioteca *<string.h>*).
- 12.15 ¿En qué ocasiones aparece el mensaje de error “null pointer assignment”? Da un ejemplo donde podría aparecer dicho error.
- 12.16 Escribe un programa que lea un número indeterminado de frases, de longitud máxima de 50

caracteres, y las guarde en un vector.

12.17 Amplia el programa anterior para que las frases puedan ser tan largas como se quiera, y que una vez leídas las frases, las ordene alfabéticamente.

Ejercicios Obligatorios de Estructuras Dinámicas de Datos

12.18 Observad el siguiente ejemplo. En él se define una estructura nodo (nodo = campos de información + apuntadores a otros nodos), compuesta por el nombre, la edad y la nota de un alumno, y un apuntador al siguiente nodo, si existe.

Al principio no existe ningún nodo, pero crearemos un par de nodos enlazados, accesibles mediante el apuntador al primer nodo:

Cuando insertamos un nuevo nodo siempre realizamos los siguientes pasos:

- 1) Reservamos espacio en memoria para el nodo, mediante la instrucción “malloc”.
- 2) Copiamos la información dentro del nodo.
- 3) Hacemos que los apuntadores dentro del nuevo nodo apunten al lugar correcto de nuestra estructura de datos.
- 4) Hacemos que los correspondientes apuntadores de nuestra estructura de datos apunten al nuevo nodo.

```
#include <stdlib.h>

typedef struct Alumno {
    char nombre[30];
    int edad;
    float nota;
    struct Alumno *siguiente;
} Alumno;

int main(void) {

    struct Alumno *primero = NULL;
    struct Alumno *aux;

    aux = (Alumno *) malloc( sizeof(Alumno) );
    strcpy(aux->nombre, "Alex");
    aux->edad = 30;
    aux->nota = 8.9;
    aux->siguiente = primero;
    primero = aux;

    aux = (Alumno *) malloc( sizeof(Alumno) );
    strcpy(aux->nombre, "Pepe");
    aux->edad = 25;
    aux->nota = 3.7;
    aux->siguiente = primero;
    primero = aux;

    exit(0);
}
```

12.19 Crea una estructura de pila donde se introducirán tan sólo nombres de alumnos (max. 30 caracteres).

- Implementa la estructura del nodo.
- Crea una función mete(nombre) que introduce un nuevo nombre en la pila.
- Crea una función saca() que elimina el último nombre introducido en la pila, devolviendo su valor.
- Crea una función lista() que imprime por pantalla los nombres introducidos en la pila.

Probad dichas funciones desde un pequeño programa principal. Vigilad que vuestro programa no se cuelgue en las situaciones especiales: sacar un nodo en la pila vacía, listar los nodos en la pila vacía, etc.

12.20 Modificad vuestra estructura de pila anterior para obtener una cola.

- Implementa la estructura del nodo.
- Crea una función mete(nombre) que introduce un nuevo nombre en la cola.
- Crea una función saca() que elimina el primer nombre introducido en la cola, devolviendo su valor.
- Crea una función lista() que imprime por pantalla los nombres introducidos en la cola.

Probad dichas funciones desde un pequeño programa principal. Vigilad que vuestro programa no se cuelgue en las situaciones especiales: sacar un nodo en la cola vacía, listar los nodos en la cola vacía, introducir el primer nodo, sacar el último nodo, etc.

12.21 Modificad vuestra estructura de pila anterior para obtener una lista ordenada.

- Implementa la estructura del nodo.
- Crea una función mete(nombre) que introduce un nuevo nombre en la lista, de manera ordenada.
- Crea una función saca(nombre) que elimina nombre escogido de la lista.
- Crea una función lista() que imprime por pantalla los nombres introducidos en la lista.

Probad dichas funciones desde un pequeño programa principal. Vigilad que vuestro programa no se cuelgue en las situaciones especiales: sacar un nodo de la lista vacía, listar los nodos en la lista vacía, introducir el primer o último nodo, sacar el primer o último nodo, etc.

12.22 Repite el ejercicio anterior, pero para una lista ordenada, doblemente enlazada y cíclica.

- Implementa la estructura del nodo.
- Crea una función mete(nombre) que introduce un nuevo nombre en la lista, de manera ordenada.
- Crea una función saca(nombre) que elimina nombre escogido de la lista.
- Crea una función lista() que imprime por pantalla los nombres introducidos en la lista.

Probad dichas funciones desde un pequeño programa principal. Vigilad que vuestro programa no se cuelgue en las situaciones especiales: sacar un nodo de la lista vacía, listar los nodos en la

lista vacía, introducir el primer o último nodo, sacar el primer o último nodo, etc.

12.23 ¿Es posible tener ordenada una lista por dos campos a la vez, por ejemplo ordenada por Nombre y también ordenada por Nota? ¿De qué manera se podría hacer?

12.24 Insertar en un árbol binario 15 valores aleatorios. Ejecutar los tres tipos de recorrido (preorden, inorden y postorden) y dibujar el árbol para cada uno de ellos.

12.25 Escribe un programa que cuente el número de nodos de un árbol binario.

12.26 Escribe un programa que calcule la suma de los nodos de un árbol binario.

12.27 Escribe un programa que calcule la altura máxima de un árbol binario. Si está vacío, la altura se considera 0, y si sólo hay la raíz se considera 1.

Trabajos finales

Objetivos de la práctica

- Diseño de una aplicación.
- Integración en una práctica de todos los conceptos vistos hasta ahora.

13.1 Base de dades d'alumnes

Volem tenir una base de dades amb tots els alumnes de l'assignatura Fonaments d'Informàtica d'un curs, on constaran els noms, grup, notes, etc. Aquesta informació es guardarà a un fitxer i en tot moment la podrem consultar o modificar. Es tracta de realitzar un programa en C que ens permeti gestionar aquestes dades.

Les fitxes dels alumnes contindran la següent informació:

```
struct t_alumne {  
    char grup[6];  
    char nom[30];  
    float notes[3];  
};
```

El programa o algorisme mostrarà un menú amb les següents opcions:

1. Obrir un fitxer d'alumnes.
2. Altes de nous alumnes.
3. Baixes d'alumnes.
4. Modificacions de les dades d'un alumne.
5. Llistat dels alumnes del fitxer.
6. Ordenació dels alumnes pel camp nom.
0. Finalitzar.

i el programa no acabarà fins que l'opció escollida per l'usuari sigui finalitzar.

Podrem tenir diferents fitxers d'alumnes. Això ens permetrà guardar en fitxers diferents les dades dels alumnes de cada curs. L'usuari escollirà amb la primera opció del menú el nom del fitxer amb el que vol treballar. Si ja s'estava treballant amb un fitxer, tancarem aquest i obrirem el nou fitxer escollit.

El disseny del programa i algorisme haurà de ser modular. Serà necessari definir els subprogrames que realitzin les següents accions:

- Obrir un fitxer, donat el seu nom passat com a paràmetre. Si no existeix el fitxer, crear-lo.
- Visualitzar per pantalla les dades d'un alumne passat com a paràmetre, incloent la seva mitjana de notes, que no hi és dins el fitxer (s'haurà de calcular).
- Cercar un alumne dins el fitxer, donat el seu nom passat com a paràmetre, i retornar la seva posició. Si no ha estat trobat, retornar -1.
- Donar d'alta nous alumnes, escrivint-los dins el fitxer. Si ha estat possible, retornar el valor booleà CERT, si no ha estat possible (l'alumne ja existia o error d'entrada/sortida al fitxer) retornar el valor booleà FALS.

- Donar de baixa alumnes existents, esborrant-los del fitxer i reorganitzant el fitxer de nou. Si ha estat possible, retornar el valor booleà CERT, si no ha estat possible (l'alumne no existia o error d'entrada/sortida al fitxer) retornar el valor booleà FALS.
- Modificar les dades dels alumnes existents, escrivint les noves dades al fitxer. Si ha estat possible, retornar el valor booleà CERT, si no ha estat possible (l'alumne no existia o error d'entrada/sortida al fitxer) retornar el valor booleà FALS.
- Llistat dels alumnes del fitxer, utilitzant el subprograma anterior per visualitzar un alumne per pantalla.
- Ordenació dels alumnes del fitxer pel camp nom.

13.2 Simulacions mitjançant un autòmat cel·lular: “el joc de la vida”

Un laboratori d'investigació distribueix una colònia de bacteris en un cultiu, que es pot considerar una superfície quadriculada de N files i M columnes. Cada casella d'aquesta superfície pot estar buida o contenir un bacteri. A partir d'una configuració inicial de bacteris en caselles, la colònia evoluciona generació a generació segons unes lleis genètiques determinades que depenen del nombre de veïns que tingui cada casella:

- Naixement: a cada casella buida que tingui exactament tres caselles veïnes ocupades per bacteris hi naixerà un nou bacteri.
- Mort per aïllament: tot bacteri que ocupi una casella amb cap o un veí mor per aïllament.
- Mort per asfíxia: tot bacteri que ocupi una casella amb més de tres veïns mor per asfíxia.
- Només sobreviuen els bacteris que tinguin dos o tres veïns.

La casella que ocupa la fila i i la columna j s'identifica mitjançant (i, j) , on $1 \leq i \leq N$, $1 \leq j \leq M$. Els veïns d'una casella (i, j) són les posicions adjacents $(i-1, j-1)$, $(i-1, j)$, $(i-1, j+1)$, $(i, j-1)$, $(i, j+1)$, $(i+1, j-1)$, $(i+1, j)$ i $(i+1, j+1)$ que estan compreses dins la superfície i que estan ocupades per un bacteri.

Els bacteris que neixen o moren no afecten fins que s'ha completat un cicle evolutiu, entenent per aquest un cicle en el que s'ha decidit la supervivència o mort dels insectes (vius en començar el cicle) d'acord a les lleis genètiques mencionades.

Així, a una superfície 4×4 , la colònia de l'esquerra de la següent figura evoluciona a les dues pròximes generacions tal i com es mostra:

.	.	*	.	.	*	*	.	.	*	.	*
.	*	*	.	.	*	*	*
.	.	*	.	.	.	*	*	.	*	.	*
.	.	*

Es demana simular l'evolució d'una colònia inicial de bacteris durant un cert nombre de transicions. Els passos a seguir seran els següents:

- Demanar la configuració del tauler. Aquesta es pot introduir per teclat o residir en un fitxer. La configuració té el següent format:
 - Els dos primers elements són dos nombres enters que indiquen el nombre de files i columnes del tauler. Cal comprovar que aquest dos valors no excedeixin les dimensions màximes del tauler.
 - Els següents elements seran parelles de nombres que indicaran on hi ha una posició del tauler ocupada. Cal comprovar que els seus valors no surtin fora del tauler definit pels dos

valors anteriors.

- Simular generacions i visualitzar-les per pantalla fins que l'usuari premi una tecla especial, per exemple ESC, per finalitzar. Per exemple

EL JOC DE LA VIDA

<p>Premi F per llegir una configuració d'un fitxer.</p> <p>Premi T per introduir una configuració per teclat.</p> <p>T</p> <p>5 5 2 2 3 2 3 3 3 4</p>	<p>Generació 0</p> <pre>. * * * *</pre> <p>Premi: F per guardar en fitxer C per la següent generació ESC per finalitzar</p>	<p>Generació 1</p> <pre>. * * * * . . . *</pre> <p>Premi: F per guardar en fitxer C per la següent generació ESC per finalitzar</p>
---	---	---

La colònia de bacteris serà una matriu d'elements del tipus cel·la. Les seves dimensions màximes venen definides per les constants MAXFIL i MAXCOL que definim a l'inici de l'algorisme.

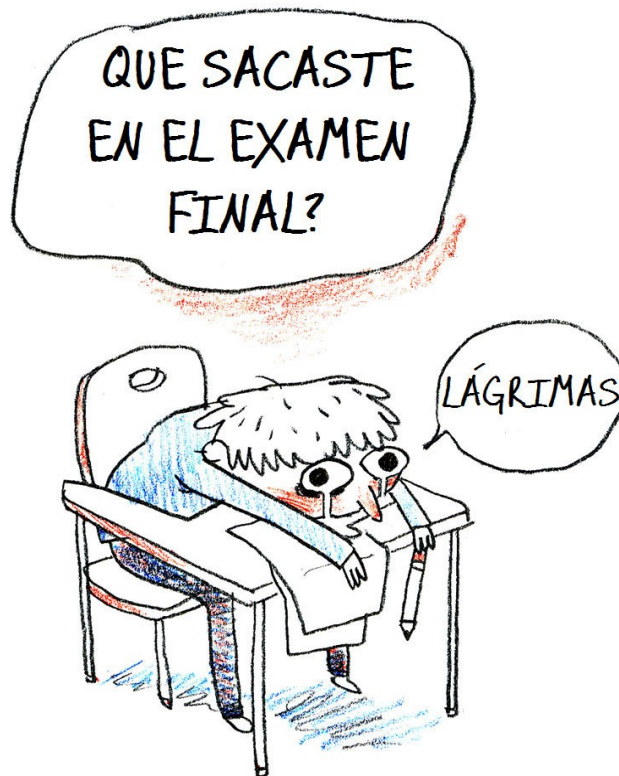
El tipus cel·la serà un registre format per una variable que indicarà si està ocupada o buida i un altre variable que guardarà el nombre de veïns que té. D'aquesta manera, per calcular la següent generació podrem recórrer la matriu cel·la a cel·la i comptar el nombre de veïns que té. A continuació tornem a recórrer la matriu marcant com a buides les cel·les dels elements que moren i marcant com a ocupades les cel·les dels nous elements que neixen.

És aconsellable (i obligatori) la utilització de subrutines en la realització de la pràctica. Per exemple:

- Una funció que retorni un nombre enter llegit pel teclat comprovant que es troba entre dos valors passats com a paràmetres de la funció. Es farà servir per llegir les dimensions de la colònia i les caselles ocupades.
- Un procediment que llegeixi la colònia d'un fitxer i un altre que la guardi.
- Un procediment que visualitzi la colònia per pantalla.
- Una funció que retorni el nombre de veïns d'una cel·la.
- Un procediment que calculi la següent generació.
- I totes les funcions i procediments que es creguin convenient per afavorir la simplicitat i llegibilitat del programa i la reutilització del codi.



¡Fin!



Bromas a parte, si te gustó aprender los conceptos básicos de programación, te recomiendo continuar aprendiendo conceptos muy interesantes que no se han practicado en este cuaderno de ejercicios: programación orientada a objetos, programación funcional, librerías de sistemas (procesos, hilos, sockets), acceso a bases de datos.

Si buscas nuevos problemas que supongan un reto para ti, te recomiendo los siguientes enlaces de competencias de programación, que vienen con ejercicios de prueba:

nivel FP	https://olimpiada-informatica.org/problem/ http://www.spoj.com/problems/classical/ https://www.jutge.org/ http://practice.geeksforgeeks.org/
nivel universitario	http://icpc.baylor.edu/worldfinals/problems https://icpcarchive.ecs.baylor.edu/index.php?option=com_onlinejudge&Itemid=8 http://uva.onlinejudge.org/index.php?option=com_onlinejudge&Itemid=8 http://code.google.com/codejam/contests.html http://www.geeksforgeeks.org/top-10-algorithms-in-interview-questions/

Unas [citas](#) para reflexionar:

“First, solve the problem. Then, write the code.” - John Johnson

“Always code as if the guy who ends up maintaining your code will be a violent psychopath who knows where you live.” - Martin Golding

“Any fool can write code that a computer can understand. Good programmers write code that humans can understand.” - Martin Fowler

“Any code of your own that you haven't looked at for six or more months might as well have been written by someone else.” - Eagleson's law

“Sometimes it pays to stay in bed on Monday, rather than spending the rest of the week debugging Monday's code.” - Christopher Thompson

“What one programmer can do in one month, two programmers can do in two months.” - Fred Brooks

“I don't care if it works on your machine! We are not shipping your machine!” - Vidiu Platon.

“Without requirements or design, programming is the art of adding bugs to an empty text file.” - Louis Srygley

“There's nothing more permanent than a temporary hack.” - Kyle Simpson

“Make it correct, make it clear, make it concise, make it fast. In that order.” - Wes Dyer

“Theory is when you know something, but it doesn't work. Practice is when something works, but you don't know why. Programmers combine theory and practice: Nothing works and they don't know why.”

“Before software can be reusable it first has to be usable.”

“I will, in fact, claim that the difference between a bad programmer and a good one is whether he considers his code or his data structures more important. Bad programmers worry about the code. Good programmers worry about data structures and their relationships.” - Linus Torvalds