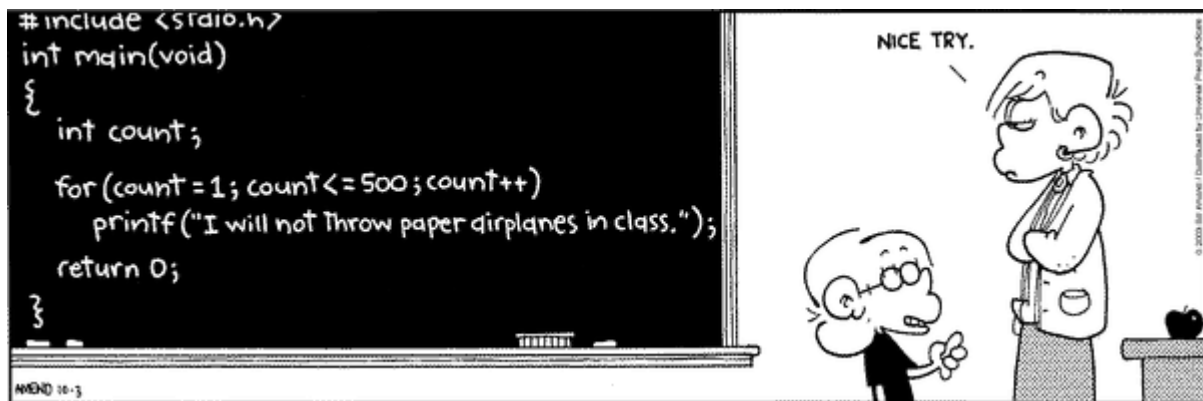


Fundamentos de programación

Ejercicios



Curso 2007/08

Ejercicios de Fundamentos de Programación - revisión 2007/08 v1.0

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 1.2 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
Anexo 1: Algorítmica (diagramas de flujo, pseudocódigo y lenguaje de programación C).....	15

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).

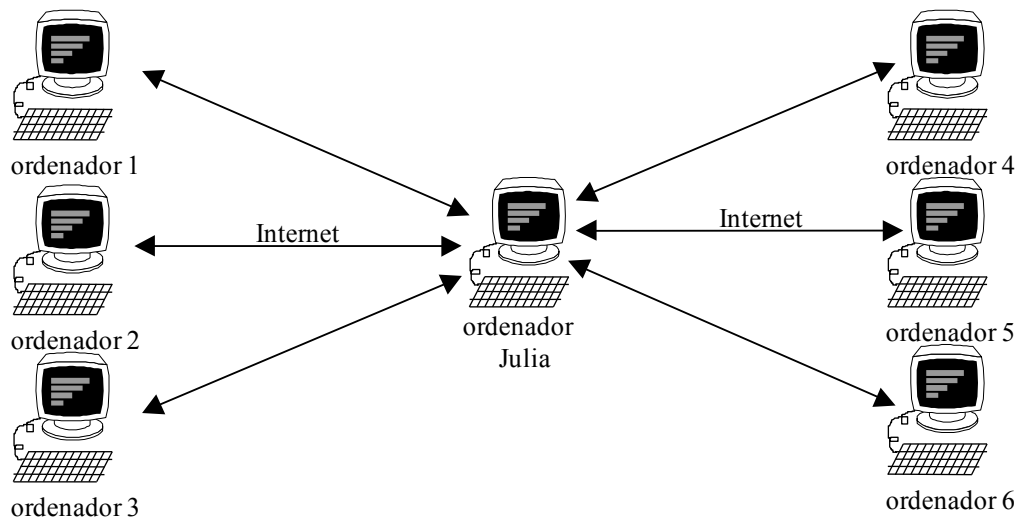
1.2 Haz un dibujo o esquema de la arquitectura de un ordenador. A continuación, en grupos de tres personas poned en común vuestros esbozos y realizad un esquema final conjunto, donde debe constar:

- 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 Core Duo.
- 1.5 ¿Qué características diferencian los sistemas operativos MS-DOS, 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 cdrom, 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 está mucho mejor):

- 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

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.

Introducción

Al final de este documento, en el Anexo 1, encontrareis una explicación de qué son los algoritmos y las estructuras de datos, así como una equivalencia entre pseudocódigo, diagramas de flujo y lenguajes C y Java.

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:

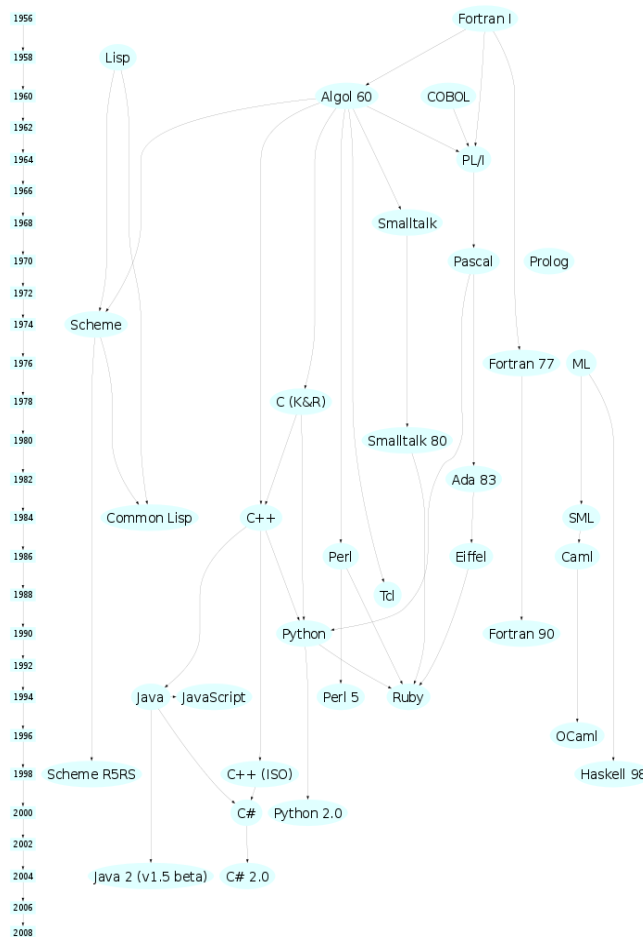
- http://es.wikipedia.org/wiki/Lenguaje_de_programación
- <http://www.digibarn.com/collections/posters/tongues/>
- http://www.oreilly.com/news/graphics/prog_lang_poster.pdf
- <http://www.levenez.com/lang/history.html>

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

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
- Curso de C: http://es.wikibooks.org/wiki/Programación_en_C
- Normas de estilo de programación en C: 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, ejecutar instrucción a instrucción visualizando el valor que toman las variables.

Existen IDEs ligeros, libres y gratuitos tanto para Linux (por ejemplo, Anjuta) como para Windows (por ejemplo, DevCPP). Existen muchos otros IDEs más pesados y completos, también libres y gratuitos: Eclipse, Kdevelop, MonoDevelop, Netbeans, ... Y otros comerciales: Borland C++, Visual Studio, ...

Sin embargo sólo he encontrado un entorno ligero que permitiera fácilmente la depuración ejecutando instrucción a instrucción visualizando el valor que toman las variables: Turbo C (para DOS), que se puede descargar actualmente en <http://dn.codegear.com/museum/>. Su versión más avanzada fue TurboC++ 3.0. (Una posible manera de ejecutarlo sería crear una máquina virtual, instalar el sistema operativo FreeDos, y posteriormente instalar TurboC).

Aviso de última hora: las últimas versiones del entorno Anjuta para Linux han mejorado bastante la depuración, pero no me da tiempo de incorporarlo en sustitución de TurboC. Lo haré en la siguiente versión de este documento. Para poder depurar sin problemas en Anjuta de momento he probado:

Archivo → Nuevo → Proyecto → C Makefile project y Opciones → Complementos → Debugger y Symbol browser . (

Otro entorno para Linux y Windows, todavía más sencillo y ligero pero sin depuración, es Geany.

Funciones del entorno TurboC

Tecla	Menú	Función
F10		Permite acceder al menú principal
F3	File → Load	Abre un archivo existente
F10,F,N	File → New	Abre un archivo nuevo
F2	File → Save	Guarda un archivo con el nombre que tiene
F10,F,a	File → Save as	Guarda un archivo con un nombre diferente
Alt + F9	Compile → Compile	Compila el programa
Ctrl + F9	Run → Run	Ejecuta el programa
F7	Run → Trace into	Ejecuta el programa paso a paso
F4	Run → Go to cursor	Ejecuta el programa hasta la posición del cursor
Ctrl + F7	Debug → Add watch	Abre una ventana para visualizar el valor de una o más variables durante una ejecución paso a paso
Ctrl + F4	Debug → Evaluate	Abre momentáneamente una ventana para visualizar y/o cambiar el valor de una variable durante una ejecución paso a paso
Alt + F5	Debug → User screen	Cambia de la pantalla de edición a la pantalla de resultados
F6	Window → Next	Cambia de ventana cuando hay varias abiertas
F1		Ayuda
May + F1	Help → Index	Muestra ayuda sobre la palabra o tema que buscamos
Alt + X	File → Exit	Sale del entorno Turbo C

El editor de Turbo C tiene opciones de trabajo con bloques: permite definir una parte de texto con el que trabajaremos, y sobre el que podemos realizar un conjunto de acciones como mover, copiar, borrar, guardar, recuperar. En la ayuda (F1) tenemos la referencia de las opciones de trabajo con el editor.

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
40000 +	-10

2000000000 + 2000000000

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:

250.30	+	300.50
1E12	+	1.0
3333333333333333.333	+	6.667
987654321004	+	-987654321002
123456789876543	+	0

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("Literal 'mesa' y 'silla' sin formato : %s %s\n" , "mesa" , "silla");
printf("Literal 'mesa' y 'silla' con formato (6) : %6s %6s\n\n" , "mesa" , "silla");

printf("Línea 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.

Anexo 1: Algorítmica (diagramas de flujo, pseudocódigo y lenguaje de programación C)

Objetivos de la práctica

- <http://es.wikipedia.org/wiki/Algoritmo>
- http://es.wikipedia.org/wiki/Estructura_de_datos
- http://es.wikipedia.org/wiki/Lenguaje_de_programaci%C3%B3n