

ANTONIO GARRIDO CARRILLO

FUNDAMENTOS DE PROGRAMACIÓN
con la STL

GRANADA
2016

© ANTONIO GARRIDO CARRILLO
© UNIVERSIDAD DE GRANADA
© Fotografías y cubierta: ANTONIO GARRIDO CARRILLO
FUNDAMENTOS DE PROGRAMACIÓN con la STL
ISBN: 978-84-338-5917-4.
Depósito legal: GR.624-2016
Edita: Editorial Universidad de Granada.
Campus Universitario de Cartuja. Granada.
Imprime: Gráficas La Madraza. Albolote. Granada.

Printed in Spain

Impreso en España.

Cualquier forma de reproducción, distribución, comunicación pública o transformación de esta obra sólo puede ser realizada con la autorización de sus titulares, salvo excepción prevista por la ley.

Índice general



1	Lenguajes de programación	1
1.1	Esquema funcional de un computador	1
1.1.1	Funcionamiento de un computador	3
1.2	Codificación de la información	5
1.2.1	Sistemas de numeración	5
1.2.2	Sistema binario en los computadores	7
1.2.3	Representación de datos simples	8
1.3	Lenguajes de programación	12
1.3.1	El lenguaje C++	14
1.3.2	Desarrollo de software	15
1.3.3	El proceso de traducción	16
1.4	Problemas	19
2	Introducción a C++	21
2.1	Un programa básico	21
2.1.1	Escritura del fichero fuente	22
2.2	Datos y expresiones	23
2.2.1	Variables	24
2.2.2	Literales	25
2.2.3	Operaciones	26
2.2.4	Operador de asignación	28

2.3	Entrada/Salida básica	30
2.3.1	Operadores de E/S estándar	31
2.3.2	Ejemplo	32
2.3.3	Entrada de datos. Algunos detalles	34
2.4	Funciones de biblioteca	35
2.5	Inicialización de variables y constantes	37
2.6	Problemas	38
3	La estructura de selección	43
3.1	Introducción	43
3.2	Operadores relacionales. Selección simple	44
3.2.1	Operadores aritméticos y relacionales	46
3.3	Selección doble <i>if/else</i>	47
3.3.1	Anidamiento	47
3.3.2	Formato de escritura	49
3.3.3	El operador ternario “?:”	49
3.4	Expresiones booleanas. Operadores lógicos	50
3.4.1	Evaluación en corto	52
3.5	Selección múltiple <i>switch</i>	53
3.6	Booleanos en C. Algunos detalles	54
3.7	Representación gráfica	56
3.8	Problemas	56
4	La estructura de repetición	59
4.1	Introducción	59
4.2	Bucles <i>while</i>	59
4.3	Bucles <i>for</i>	63
4.3.1	Nuevos operadores: incremento y decremento	65
4.3.2	Nuevos operadores de asignación	67
4.3.3	El operador coma	67
4.3.4	Ejemplos	67
4.4	Bucles <i>do/while</i>	71
4.5	Estructuras de control anidadas	72
4.6	Declaración y ámbito de una variable	75
4.6.1	Ocultación de variables	76
4.7	Programación estructurada y sentencia <i>goto</i>	76
4.7.1	Sentencias <i>continue</i> y <i>break</i>	77
4.8	Problemas	77

5	Funciones	81
5.1	Introducción	81
5.2	Funciones	82
5.2.1	Un programa con funciones. Declaración y definición	84
5.2.2	Funciones <i>void</i>	91
5.2.3	Paso por valor y por referencia	93
5.2.4	Función <i>main</i> . Finalización del programa	97
5.3	Diseño de funciones	98
5.3.1	Variables globales	100
5.3.2	Funciones y programación estructurada	101
5.3.3	Especificación de funciones	104
5.3.4	Pruebas	106
5.3.5	Errores	106
5.4	Problemas	107
6	Vectores de la STL	109
6.1	Introducción	109
6.2	Vectores	110
6.2.1	Declaración	110
6.2.2	Acceso al contenido	112
6.2.3	Funciones y tipo vector	117
6.2.4	Búsqueda con vectores de la STL	118
6.2.5	Algoritmos clásicos de ordenación	121
6.3	Vectores dinámicos	132
6.3.1	Eficiencia y vectores dinámicos	136
6.4	Matrices	137
6.4.1	Declaración	138
6.4.2	Matrices rectangulares	139
6.4.3	Matrices no rectangulares	141
6.5	Problemas	143
7	Cadenas de la STL	145
7.1	Introducción	145
7.2	Tipo string	146
7.2.1	Declaración	146
7.2.2	Literal de tipo string	146
7.2.3	El tipo <i>string</i> como contenedor de <i>char</i>	147
7.2.4	E/S de objetos string	147
7.2.5	Asignación de cadenas	150
7.2.6	Operaciones relacionales	151
7.2.7	Otras operaciones frecuentes	152
7.2.8	Funciones y tipo string	156

7.3	El tipo char: codificaciones	156
7.3.1	Código ASCII: Extensiones	157
7.3.2	Unicode	160
7.3.3	Editor de texto y codificación usada	162
7.4	Problemas	162
8	Estructuras y pares	163
8.1	Introducción	163
8.1.1	Tipos de datos definidos por el usuario	164
8.2	Estructuras	164
8.2.1	Declaración de variables	165
8.2.2	El operador punto	165
8.2.3	Copia y asignación de estructuras	167
8.2.4	Ordenar estructuras	170
8.3	Pares con la STL	175
8.3.1	El tipo <i>Pair</i>	175
8.3.2	Operaciones con <i>pair</i>	177
8.3.3	Diccionarios	178
8.4	Problemas	183
9	Recursividad	185
9.1	Introducción	185
9.1.1	Recursión directa e indirecta	186
9.2	Ejemplos de funciones recursivas	186
9.2.1	Factorial de un número entero	186
9.2.2	Suma de los elementos de un vector	188
9.2.3	Listado de los elementos de un vector	189
9.3	Gestión de llamadas: la pila	189
9.3.1	La pila del sistema	190
9.3.2	Pila y recursividad	192
9.3.3	Gestión de la memoria	192
9.4	Diseño de funciones recursivas	193
9.4.1	Implementación de funciones recursivas	195
9.4.2	Descomposición en números primos	199
9.5	Recursivo vs iterativo	201
9.5.1	Recursión de cola	202
9.6	Ejemplos de funciones recursivas	203
9.6.1	Sucesión de Fibonacci	203
9.6.2	Torres de Hanoi	203
9.6.3	Ordenación por selección	205
9.6.4	Ordenación por mezcla	206
9.7	Problemas	208

10	Introducción a flujos de E/S	211
10.1	Introducción	211
10.2	Flujos de E/S	212
10.2.1	Flujos y buffers	212
10.2.2	Flujos globales predefinidos	214
10.3	Operaciones básicas con flujos	216
10.3.1	Tamaño finito de los flujos	216
10.3.2	Estado de los flujos	218
10.3.3	Otras funciones de entrada útiles	223
10.4	Flujos asociados a ficheros	225
10.4.1	Apertura y cierre de archivos	226
10.4.2	Paso de flujos a funciones	230
10.5	Flujos y C++98	232
10.5.1	Apertura con <i>cadena-C</i>	233
10.5.2	Reset del estado con <i>open</i>	233
10.6	Problemas	233
11	Compilación separada	235
11.1	Introducción	235
11.2	Compilación separada	236
11.2.1	Compilación y enlazado	237
11.2.2	El preprocesador. Ficheros <i>h</i> y <i>cpp</i>	238
11.3	El preprocesador: Múltiples inclusiones de archivos cabecera	240
11.3.1	La directiva <i>define</i>	240
11.3.2	Compilación condicional	242
11.3.3	Un ejemplo más complejo	242
11.3.4	Dependencias entre archivos	246
11.4	Bibliotecas	246
11.5	Espacios de nombres	248
11.5.1	Creación de espacios de nombres. Nombres calificados	249
11.5.2	Eliminando la calificación: <i>using</i>	251
11.5.3	Espacio de nombres y distintos archivos	251
11.5.4	Evitando <i>using</i> en archivos cabecera	252
11.6	Objetos y múltiples ficheros	252
11.6.1	Variables locales a un fichero	253
11.6.2	Constantes globales	254
11.6.3	Funciones locales a un fichero	255
11.7	Problemas	255

12	C++11/14	257
12.1	Introducción	257
12.2	Inicialización de tipos simples	258
12.2.1	Llaves para tipos simples	259
12.2.2	Deducción del tipo	260
12.3	El tipo <i>vector</i>	261
12.3.1	Terminación con <code>>></code>	261
12.3.2	Bucle para recorrer un contenedor	261
12.3.3	Listas de inicialización	263
12.3.4	Optimización de movimiento de objetos	265
12.4	El tipo <i>string</i>	266
12.4.1	El tipo <i>string</i> como contenedor	266
12.4.2	Conversiones a/desde tipos numéricos	268
12.5	Estructuras, pares y tuplas	270
12.5.1	Inicialización con llaves	270
12.5.2	Tuplas	271
12.5.3	Pares y tuplas	273
A	Solución a los ejercicios	275
A.1	Lenguajes de Programación	275
A.2	Introducción a C++	275
A.3	La estructura de selección	278
A.4	La estructura de repetición	283
A.5	Funciones	290
A.6	Vectores de la STL	296
A.7	Cadenas de la STL	307
A.8	Estructuras y pares	311
A.9	Recursividad	316
A.10	Introducción a flujos de E/S	319
B	Generación de números aleatorios	323
B.1	Introducción	323
B.2	El problema	323
B.2.1	Números pseudoaleatorios	324
B.3	Transformación del intervalo	325
B.3.1	Operación módulo	326
B.3.2	Normalizar a $U(0,1)$	326

C	Ingeniería del software	329
C.1	Introducción	329
C.2	Actividades en el proceso del software	330
C.3	La necesidad de un enfoque de calidad	330
C.4	Paradigmas en el desarrollo del software	331
C.5	Modelo del sistema	332
C.6	Paradigmas de la programación	333
C.6.1	Programación procedimental	333
C.6.2	Programación modular	334
C.6.3	Abstracción de datos	334
C.6.4	Programación orientada a objetos	335
C.6.5	Programación genérica	336
C.6.6	Metaprogramación	336
C.6.7	Programación funcional	337
C.7	Desarrollo de programas	337
C.7.1	Análisis	338
C.7.2	Diseño	338
C.7.3	Implementación	341
C.7.4	Pruebas	341
D	Tablas	343
D.1	Tabla ASCII	343
D.2	Operadores C++	344
D.3	Palabras reservadas de C89, C99, C11, C++ y C++11	346
	Bibliografía	349
	Índice alfabético	353



El objetivo de este libro es presentar los conceptos fundamentales de programación en el contexto del lenguaje de programación C++. A diferencia de otros libros, no se pretende mostrar todos los detalles de este lenguaje, sino enseñar los conceptos más básicos de programación.

Aprender a programar

Comenzar en el mundo de la programación no es una tarea trivial. A pesar de que un programador experimentado probablemente mirará los contenidos del curso convencido de su simplicidad, un estudiante sin ningún conocimiento previo podría encontrar complejos los detalles más insignificantes. Generalmente, la experiencia oculta la dificultad de los detalles de la implementación para enfocar el interés en el diseño de la solución.

En muchas ocasiones, la tarea de plantear un curso para aprender a programar centra su interés en las metas que se quieren conseguir, antes que en los primeros pasos que se deben dar. De hecho, este planteamiento es el que lleva, en muchos casos, a crear cursos iniciales novedosos en los que se pretende que el alumno esté realizando diseños orientados a objetos al mes de comenzar su formación o que realice complejas interfaces de usuario en base a un lenguaje de muy alto nivel. Si realiza un programa en tres líneas que realiza una operación sorprendente, no significa que sepa programar.

Si desea formar a un buen programador, es absurdo intentar conseguir que éste realice un gran diseño a los pocos meses de comenzar su formación. Muchas decisiones de diseño son consecuencia de un largo período de formación y experimentación. Si revisa los programas que durante años ha ido generando un buen programador, seguramente descubrirá una diferencia notable entre los primeros y los últimos. Es posible que los últimos le parezcan obras de arte, pero no olvide que su formación comenzó a un nivel más básico.

Teniendo en cuenta estas consideraciones, el curso que se presenta no es más que un curso básico de programación estructurada. Sin embargo, se ha diseñado pensando en los conocimientos que el estudiante debería tener para abordar otros cursos y materias más avanzadas relacionadas con la informática y la programación.

El lenguaje de programación

El lenguaje seleccionado para desarrollar el curso es C++. La discusión sobre qué lenguaje es ideal para un primer curso ha estado presente desde el comienzo de los estudios de informática. De hecho, ya en el año 1970 se presenta el lenguaje Pascal, especialmente diseñado para aprender a programar.

Los criterios para seleccionar un lenguaje u otro pueden ser muy variados. Por ejemplo, se proponen los lenguajes más usados, los que permiten mayor productividad, o los que está usando la empresa de software que esté de moda en ese momento. Sin embargo, lo más importante debería ser tener en cuenta que el estudiante no tiene ningún conocimiento y que es un primer curso de programación, ya sea para tener algunos conocimientos o para servir como base para otros cursos. Con estas condiciones, la elección es más sencilla.

Un lenguaje bastante usado en los primeros cursos es el lenguaje C. Es un lenguaje de bajo nivel en relación a otros, basado en la programación estructurada. Su simplicidad y su uso generalizado dentro y fuera de ámbito docente lo hace un buen candidato. Sin embargo, en muchos casos se desestima porque resulta demasiado alejado de la sintaxis y los conceptos que se usan en la mayoría de lenguajes de alto nivel actuales.

Por otro lado, lenguajes como Java son una alternativa para proponer una forma docente con la que priorizar conceptos de programación dirigida a objetos. El problema es que un alumno sin ningún conocimiento está lejos de entender esos conceptos, para los que realmente necesita conocer algunas estructuras más básicas que aparecen en la programación estructurada. Estos cursos pueden resultar algo divulgativos en algunos contenidos y más eficaces cuando se dedica un tiempo a olvidar esos diseños dirigidos a objetos y a centrarse en conceptos más básicos como las estructuras de control o la abstracción funcional.

Una solución que podría interpretarse como intermedia es el uso de un lenguaje multiparadigma como C++. Éste plantea distintos paradigmas de programación, por lo que el curso podría desviarse tanto a un modelo como a otro. Por ejemplo, podría plantearse el estudio de los aspectos más cercanos a C, resultando un curso con cierto paralelismo a un curso C primero y Java después.

La versatilidad de este lenguaje es un aspecto que nos hace seleccionarlo para este primer curso, lógicamente, teniendo cuidado de diseñar un curso que seleccione los contenidos necesarios y suficientes para un curso de fundamentos de programación. De hecho, en muchos casos se rechaza como primer lenguaje dada su complejidad, justificándolo incluso diciendo que para usar este lenguaje es necesario conocer el lenguaje C. En este libro, se propone un contenido guiado por aspectos docentes, evitando profundizar en detalles técnicos del lenguaje, seleccionando los conceptos que componen un curso completo, autocontenido y que preparan al estudiante tanto para temas de más bajo nivel —con un lenguaje como C— como para temas de más alto nivel —como en encapsulamiento y la abstracción—. Precisamente, lo que debería ser un curso de fundamentos de programación.

Este enfoque permite poner este complejo lenguaje al alcance del lector, aunque no tenga conocimientos previos, y lo prepara para profundizar y asimilar fácilmente otros conceptos más complejos de éste u otro lenguaje. Por tanto, resulta una referencia especialmente útil para planes de estudios donde quieran actualizar los contenidos desde lenguajes tradicionales —como *Pascal* o C— a un lenguaje con las posibilidades de C++.

Introducción a la programación para todos

La razón principal que ha motivado la creación de este libro ha sido la necesidad de ofrecer un documento práctico que facilite tanto la docencia del profesor como el aprendizaje del alumno en las carreras de ingeniería.

La evolución de los planes de estudios en los últimos 30 años ha desembocado en un modelo de enseñanza en el que el estudiante tiene una gran responsabilidad: por un lado, se han ampliado los

contenidos de la carrera, mientras por otro se han disminuido el tiempo de enseñanza en el aula. Una parte del esfuerzo debería realizarse de forma autodidacta. Los estudiantes deben asimilar una gran cantidad de contenidos en poco tiempo, por lo que resulta muy difícil disponer de un buen material de estudio si se limita únicamente a los apuntes de clase. Por ello, es necesario disponer de algún libro de apoyo.

Este libro intenta compensar esta dificultad para un estudiante de primer curso. Teniendo en cuenta los planes de estudios en el contexto europeo, así como el deseo de muchos lectores para el autoaprendizaje, este libro se convierte en una referencia útil para aprender a programar. No sólo puede usarse en las titulaciones de ingeniería informática, sino en otras ingenierías, en titulaciones de ciencias, en estudios medios, e incluso de forma autodidacta.

Organización del libro

La exposición de los temas se ha realizado pensando en el objetivo docente del libro. Para ello, se desarrollan los contenidos guiados por los conceptos siguiendo una dificultad incremental, sin suponer ningún conocimiento previo del lector. Se evita la exposición de contenidos guiados por el lenguaje, especialmente evitando detalles que no son importantes para la formación general de un programador.

Los temas se abordan con una discusión razonada de los contenidos, presentando los detalles que el estudiante debería conocer, pero también justificando por qué y cómo se utilizan en la práctica. Se intenta evitar la exposición “por recetas”, es decir, que el estudiante memorice soluciones en lugar de entender cómo crearlas. El estudiante debería poder crear buenas soluciones, en lugar de conformarse con “programas que funcionan”. Las soluciones deberían reflejar una capacidad de diseño que garantice los mejores resultados en cursos avanzados, donde un proyecto de cierto tamaño sólo puede resolverse con los mejores fundamentos de programación.

En este desarrollo, se exponen los temas incluyendo ejemplos ilustrativos. Además, se proponen ejercicios a lo largo del tema, independientes de los problemas finales del tema. El lector debería asimilar los contenidos y ser capaz, en gran medida, de resolver estos ejercicios propuestos como una forma de confirmar que se ha entendido el tema. Además, el libro incluye las soluciones al final para aquellos estudiantes que tengan más dificultades o quieran confirmar o comparar sus soluciones.

Contenidos

Los capítulos se han organizado para que el lector vaya asimilando el lenguaje poco a poco. En este sentido, los temas no pretenden presentar un aspecto del lenguaje para que se conozca completamente antes de pasar al siguiente. Cada tema introduce algunos aspectos adicionales para que el lector empiece a utilizarlos, pero pueden ser completados y asimilados completamente a lo largo de ese y otros capítulos posteriores.

Un repaso rápido por el índice le puede hacer entrever que es un libro organizado de una forma muy diferente a otras referencias. Aunque se trabaja con C++, no aparecen tipos tan básicos como los *vectores-C* o los punteros. En su lugar, se incluyen tipos de la STL que resuelven los mismos problemas eliminando detalles de muy bajo nivel que no son necesarios en un curso de fundamentos, y ofreciendo una interfaz bien desarrollada que facilita al alumno avanzar más fácilmente en cursos posteriores.

En el **capítulo 1** se introduce al lector en el mundo de la programación. Para ello, suponemos que no tiene experiencia anterior, y por tanto incluimos los conceptos más básicos para que entienda lo que es un ordenador, un lenguaje de programación o un programa.

En el **capítulo 2** se abordan las ideas más simples para introducir al lector en el lenguaje C++. El objetivo es que pueda realizar pequeños programas cuanto antes, para conocer de forma práctica

qué quieren decir esos conceptos expuestos en la lección. Se presentan conceptos muy sencillos —como lo que es una variable o un tipo de dato— pero que pueden resultar complicados al ser tan novedosos. Se muestran ejemplos con los primeros algoritmos en este lenguaje, programas muy simples que incluyen sentencias de declaración de datos, expresiones y E/S.

En el **capítulo 3** se introduce la idea de flujo de control y bifurcación en un programa. Ese control de flujo motiva la presentación del tipo booleano junto con los operadores adicionales, así como las sentencias **if-else** y **switch**.

En el **capítulo 4** completamos las estructuras básicas de control con los bucles: **for**, **while** y **do-while**. Los programas ya tienen bastante complejidad como para añadir algunas ideas sobre declaración y ámbito de las variables.

En el **capítulo 5** se introducen las funciones como la herramienta más básica de modularización. En este punto el alumno debe realizar un esfuerzo por empezar a cultivar su habilidad de abstracción, aunque a este nivel se limite a crear abstracciones funcionales. En este tema sólo se presentan los detalles más básicos sobre funciones, de forma que el lector sea capaz de manejar problemas más complejos y de crear soluciones estructuradas. El aumento en la complejidad de los programas permite empezar a conocer el problema de la modularización y la importancia de obtener un buen diseño.

En el **capítulo 6** se presentan los tipos de datos compuestos, concretamente los vectores y matrices. En los temas anteriores el lector debe haber asimilado los conocimientos básicos para desarrollar algoritmos de forma estructurada. En este tema, se avanza en la complejidad de los datos introduciendo nuevos tipos y relacionándolos con los conocimientos adquiridos. El contenido se basa en el tipo **vector** de la STL, por lo que será necesario que el alumno empiece a adaptarse a una nueva interfaz y comience a trabajar con objetos y sus operaciones. Estas nuevas estructuras de datos permiten empezar a introducir algunos algoritmos clásicos, como los de búsqueda y ordenación, que son una base que cualquier programador debería conocer pero que a su vez constituyen ejemplos ideales para empezar a practicar.

En el **capítulo 7** se presentan las cadenas, el tipo **string** de la STL. En gran medida, es una nueva revisión de ideas que se han expuesto en el tema anterior, aunque en este caso se especializa para una secuencia de caracteres en lugar de un vector de un tipo general. Se aprovecha el tema para hablar en mayor profundidad sobre el tipo **char** y la codificación.

En el **capítulo 8** se presentan las estructuras, como una nueva forma de definir tipos de datos compuestos, en este caso heterogéneos. Se muestran los aspectos más básicos y se relacionan con los temas anteriores. Además, se introducen los pares —tipo **pair**— de la STL, un tipo especialmente importante por su uso en otras estructuras de datos, no sólo en este lenguaje.

En el **capítulo 9** se presenta la recursividad, con lo que se completan los conocimientos relacionados con funciones en este curso básico. Aunque la recursividad no parezca más que una simple posibilidad más a la hora de llamar a una función, muchos programadores novatos encuentran dificultades para crear funciones recursivas. Ese nivel extra de abstracción sobre lo que es una función parece un problema para estos programadores. Por eso, el tema se ha retrasado para que el alumno pudiera practicar con funciones antes de plantear esta posibilidad. En este tema, se estudia en detalle el problema de su diseño, acompañándolo con múltiples ejemplos y consejos. Además, se presentan los detalles de la pila y cómo el compilador resuelve las llamadas a función. Esta visión permite entender mejor cómo funciona un programa, facilitando tareas tan importantes como la traza de un programa.

En el **capítulo 10** se estudian los flujos de E/S. Este tema se limita a estudiar los conceptos fundamentales, explicando mejor cómo funcionan los objetos **cin** y **cout**, intensivamente usados durante el curso. En este punto, se revelan algunos aspectos que habían sido esquivados en los temas anteriores, haciendo que el estudiante pueda controlar mejor estos objetos y ampliando sus posibilidades mediante la introducción de flujos asociados a ficheros.

En el **capítulo 11** se estudia la compilación separada. Aunque se han presentado sólo fundamentos del lenguaje, evitando muchos tipos y detalles, el contenido de los temas anteriores ya permite al estudiante abordar problemas de cierta complejidad. Tal vez la solución no sea la más óptima, puesto que se conoce sólo una parte del lenguaje; a pesar de eso, a este nivel, el estudiante está preparado para crear verdaderos proyectos que basados en programación estructurada configuren una buena solución a un problema. La compilación separada permite que puedan crearse estas soluciones con un buen diseño y una forma eficaz de distribuir el código.

Finalmente, en el **capítulo 12** se completa el curso con contenidos referidos a C++11/14. Desde el punto de vista de los conceptos no aporta mucho más a los temas anteriores. Pero desde el punto de vista práctico no podemos olvidar que los futuros programadores de C++ trabajarán en el estándar actual.

Se ha dejado en un tema extra porque en un curso de fundamentos el lenguaje C++98 es suficiente, lo que hace innecesario complicarlo más. De hecho, mi recomendación es comenzar de esta forma. Una vez que están asentadas las bases de la programación, resulta mucho más sencillo añadir las novedades de C++11. Las secciones del tema están ordenadas conforme al curso, por lo que un profesor interesado en el nuevo estándar podría añadirlas de forma relativamente sencilla a las distintas partes del curso.

Además, se incluyen varios apéndices que completan los temas anteriores:

- En el **apéndice A** aparecen las soluciones a los ejercicios propuestos a lo largo del libro. Para que el lector asimile más fácilmente los contenidos, se proponen ejercicios a la largo de los temas, de forma que el estudio de éstos se realice de una forma más interactiva. Cuando se propone un ejercicio, el lector debería intentar resolverlo proponiendo su solución, tras lo que debería consultar la solución propuesta en este apéndice.
- En el **apéndice B** se muestran las herramientas más básicas que podemos usar para la generación de números aleatorios. Son las mismas que puede encontrar en C. En un curso de introducción a la programación pueden resultar muy útiles, ya sea para crear algunos programas que lo necesitan —por ejemplo, pequeños juegos— como para generar secuencias de datos para pruebas.
- En el **apéndice C** se realiza una discusión sobre la ingeniería del software. Con los conocimientos adquiridos, el lector está preparado para resolver problemas con cierta envergadura. En este sentido, resulta conveniente empezar a introducir algunos conocimientos sobre ingeniería del software, así como insistir sobre algunos aspectos relativos a la calidad del software desarrollado. La ingeniería del software es un tema que daría para escribir no uno, sino varios libros. En este apéndice, sólo se pretenden exponer algunos conceptos básicos sobre lo que un programador, aunque sea novato, debería conocer.
- En el **apéndice D** se presentan algunas tablas de interés para un programador de C++.

Finalmente, se presenta la bibliografía, donde aparece la lista de referencias que se han usado para la creación de este libro o que resultan de especial interés para que el lector amplíe conocimientos.

Sobre la ortografía

El libro se ha escrito cuidando la redacción y la ortografía; no podría ser de otra forma. Se ha intentado cuidar especialmente la expresión y la claridad en las explicaciones. Como parte de este cuidado, es necesario incluir en este prólogo una disculpa por no respetar todas y cada una de las normas que actualmente establece la Real Academia Española (RAE) sobre el español.

En primer lugar, es difícil concretar un lenguaje claro y duradero en el contexto de la informática, especialmente porque cambia e incorpora nuevos conceptos continuamente. Unido al retraso que requiere el análisis y la decisión por parte de la RAE para incluirlos en el diccionario de la lengua española, siempre podemos encontrar alguna palabra que se adapta desde el inglés y no está aún

aceptada. Por ejemplo, usaremos la palabra *unario* como traducción de *unary*. Otras palabras que usamos de forma natural cuando comentamos código en C++ pueden ser *booleano*, *token*, *buffer*, *precondición*, *endianness*, *log*, *argumento*, *preincremento*, *prompt*, etc.

Por otro lado, es habitual que el programador trabaje con documentación técnica que está escrita en inglés. Esto hace que un uso de palabras similares en español sea una forma de facilitar la lectura rápida y cómoda. Por ejemplo, la palabra *indentación* en lugar de *sangrado*, *casting* en lugar de *moldeado* o *array* en lugar de *arreglo*.

Además, es probable que haya alguna otra palabra. Por ejemplo, la palabra *sumatoria* que encuentro en la RAE como *sumatorio*. ¿Por qué la hacen masculina si *suma* es femenina? Que yo recuerde, todos mis profesores y compañeros la han llamado y la llaman *sumatoria*.

Sin duda, mis cambios son un error, pero desde un punto de vista técnico prefiero no darle excesiva importancia. Es más importante superar la dificultad de los contenidos técnicos, dejando cuestiones de la lengua en un segundo plano.

A pesar de todo, he querido ajustarme en lo posible a lo que nos dice la RAE; por respeto al estupendo y complejo trabajo que realizan, pero sobre todo por respecto a nuestra lengua materna, el principal legado común que nuestros ancestros nos han dejado.

Una vez certificado mi respeto a la lengua y a la RAE como institución, tengo que disculparme por introducir alguna palabra u ortografía que no es actual. Esta disculpa se refiere, especialmente, al uso de la tilde diacrítica en el adverbio *sólo* —note la tilde— y los *pronombres demostrativos*. En mi humilde opinión, la distinción gráfica con respecto al adjetivo *solo* y los *determinantes demostrativos*, respectivamente, justifican claramente su uso. Si quiero priorizar la legibilidad, la claridad, el énfasis de lo que comunico, creo que su uso está justificado.

Por supuesto, seguramente habrá otros errores ortográficos o gramaticales; éstos son fruto de mi torpeza e ignorancia, por lo que pido humildemente disculpas.

El entorno de desarrollo

El entorno de desarrollo es otra de las discusiones que se plantean repetidamente en los primeros cursos de programación. La decisión incluye no sólo el lenguaje, sino el sistema operativo, los entornos de desarrollo —*IDE*— e incluso otras herramientas.

En nuestro caso, la elección del lenguaje C++ nos abre un amplio abanico de posibilidades. Lo más sencillo es aconsejar software libre que permite a cualquier usuario comenzar sin ningún coste. Una opción ideal, por tanto, es el compilador de la *GNU* que está disponible para múltiples plataformas.

Por otro lado, nuestro interés es centrar el esfuerzo en los conceptos de programación. No es conveniente distraer al estudiante con detalles técnicos sobre gestión de proyectos antes de que sea necesario. Por eso, lo más recomendado es usar algún tipo de entorno de desarrollo que simplifique todas estas tareas.

En nuestro caso, hemos usado con éxito el entorno *codeblocks*, gratuito y disponible para distintas plataformas, incluyendo las habituales *Windows* y *GNU/Linux* tan habituales entre los estudiantes. Por supuesto, cualquier otro entorno similar podría ser válido, aunque esta decisión se sale de los objetivos de este libro. Cada profesor deberá escoger el entorno ideal basado en un compilador que siga el estándar C++, por lo que no tendrá ningún problema para llevar a la práctica las lecciones de este libro.

Agradecimientos

En primer lugar, quiero agradecer el trabajo desinteresado de miles de personas que han contribuido al desarrollo del software libre, sobre el que se ha desarrollado este material, y que

constituye una solución ideal para que el lector pueda disponer del software necesario para llevar a cabo este curso.

Por otro lado, no puedo olvidar a las personas que me han animado a seguir trabajando para crear este documento. El trabajo y tiempo que implica escribir un libro no viene compensado fácilmente si no es por las personas que justifican ese esfuerzo. En este sentido, quiero agradecer a mis alumnos su paciencia, y recordar especialmente a aquellos, que con su esfuerzo e interés por aprender, han sabido entender el trabajo realizado para facilitar su aprendizaje.

Finalmente, quiero agradecer el apoyo de los compañeros que han usado este material como guía en sus propias clases, ya que con ello me han animado a completarlo y a terminar este libro.

A. Garrido.
Mayo de 2016.

Lenguajes de programación

Esquema funcional de un computador	1
Codificación de la información	5
Lenguajes de programación	12
Problemas	19

1.1 Esquema funcional de un computador

En este apartado vamos a conocer, de forma muy simplificada, qué es y cómo funciona una computadora. Una definición simple es la siguiente (véase la figura 1.1):

Definición 1.1 — Computadora. Un *computador* es una máquina diseñada para aceptar un conjunto de datos de entrada, procesarlos^a, y obtener como resultado un conjunto de datos de salida.

^aPor procesar datos entendemos la realización de operaciones aritméticas y lógicas con ellos.



Figura 1.1
E/S de datos de una computadora.

Por otro lado, debemos tener en cuenta que con una computadora podemos realizar distintas tareas. Para ello, no sólo podemos introducir datos para procesar, sino también las instrucciones que indican cómo se procesan. Normalmente estas instrucciones están previamente almacenadas en la computadora, por lo que el usuario sólo necesita interactuar con ella por medio de la entrada y salida (E/S) de datos.

Definición 1.2 — Programa. Denominamos *Programa* al conjunto ordenado de instrucciones que indican a la computadora las operaciones que se deben llevar a cabo para realizar una determinada tarea.