Series HOMEPET

Introducción al C++

Segunda Parte: un C con esteroides

Ing. Simón Galliano Vidal, ret.

5 Temas
13 Ilustraciones
31 Tablas
5 Programas auxiliares
20 Programas resueltos
30 Ejercicios propuestos
1 Ejercicio de comparación

noviembre 2020

Publicado online por El Archivo Histórico de la Ciudad de Manzanillo

URL: http://www.enciclopedia-manzanillo.cu

Fecha de publicación: 20 de noviembre de 2020

Versión: 3.2C.1020

Autor: S. GALLIANO - 2019

Publicista: Lic. Delio Orozco.

ISBN: 978-959-7179-67-2(2)

Páginas: 160

CONTENIDOS

ILUSTRACIONES

TABLAS

Segunda Parte: un C con esteroides

La potencia de la STL

El proceso de preparar programas para una computadora digital es especialmente atractivo, no sólo porque puede ser económica y científicamente premiado, pero también porque puede ser una experiencia estética, casi como componer poesía o música.

Donald E. Knuth

Profesor Emérito de la Universidad de Stanford

Ing. Simón Galliano Vidal, ret. 2020

1 - SISTEMA DE DESARROLLO

> *Code::Blocks es un completo IDE multiplataforma para el desarrollo de programas en el lenguaje ISO C++, creado en C++ y liberado bajo la Licencia pública general de GNU. (Wikipedia en español)*

Para esta segunda parte de la serie y en un sistema operativo (SO en lo adelante) Windows 10 Home x64, con un Celeron® J1800 @ 2.41 GHz y 8 Gb de memoria, se usa el IDE Code::Blocks, versión 20.03-r11983 de dominio libre porque:

- Es gratis y está en Internet (http://www.codeblocks.org)
- Está hecho totalmente sobre C++, siendo muy liviano al SO
- Se ejecuta en cualquier versión actual de Windows™, sea XP, Vista, Seven, 8 ó 10.
- Automáticamente busca en el SO un compilador cualquiera para C++ y lo integra. Acá se usó la suite para Windows tdm-gcc-4.6.1, edición estándar de 32 bits para C++ que es bastante conforme con ISO C++11[1].
- El manejo de su entorno laboral es relativamente estable, cómodo, eficiente y flexible.
- Tiene opciones para crear programas de consola (*Console application*, en inglés.)

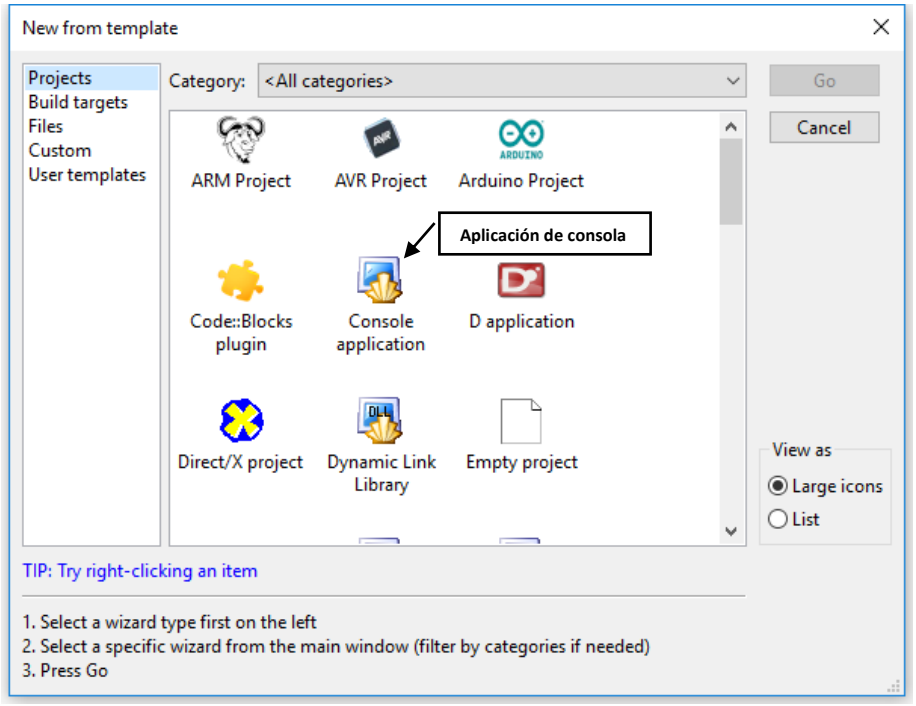


Ilustración 1. Opciones básicas de inicio

Todos los programas del texto toman como opción *Console application*

Esto no es un respaldo oficial o a ultranza de dicho IDE. Allá afuera hay un buen puñado de ellos. Siéntase libre de estudiar usando uno de su gusto, aunque tendrá que ajustar lo que aquí se muestra, siempre con la esperanza de que las adaptaciones sean verdaderamente mínimas; en la inmensa mayoría de los casos, nulas, ¡el Santo Grial de la portabilidad!

[1] ISO C++ se refiere específicamente a la versión normalizada de C++ y en el texto se referencian como la misma cosa. Algunos fabricantes de compiladores se apartan de la norma y pierden portabilidad. Eso no pasa aquí.

En Microsoft Windows™ y bajo este IDE los programas correrán en una ventana separada que el usuario cerrará al concluir éstos, pero al correr aparte el ejecutable, para mantener la salida en consola basta con incluir **`<cstdlib>`** (*C++ standard library* en inglés), la biblioteca estándar de ISO C++, y pasarle al SO la orden de pausar antes del retorno a Windows**:** **`system( "pause" );`**

Objetivos del texto

En un nivel introductorio y con la precondición de que el lector llega directamente del campo del ANSI C o de uno similar[2], los objetivos generales de este segundo tomo son dos:

1. Enseñar elementos de programación genérica de plantillas de funciones.
2. Mostrar como potenciar al C++ mediante el uso de la biblioteca estándar de plantillas o *Standard Template Library*, en inglés –STL en lo adelante.

   Esto último significa que se tratará de enseñar al programador novato a resolver un problema delegando a la STL el trabajo de manejar los datos en contenedores apropiados, mediante algoritmos adecuados, pudiendo desarrollar sus propias plantillas de funciones si así lo desea.

El objetivo particular de cada tema básico es enseñar tanto la aplicación de plantillas de funciones, como las bondades de la STL en programas o ejercicios planteados previamente en el primer tomo de la serie, cubriendo un conjunto de conceptos que le son asociados de alguna suerte, con un programa de ejemplo como patrón de comparación contra otro similar, dado en el primer tomo, demostrando la ganancia en poder de abstracción, facilidad y seguridad de codificación que un programador, ya sea novel o de experiencia estándar, puede obtener con algo de voluntad y un mínimo de esfuerzo, lo cual lo elevará sobre su actual nivel amateur o profesional.

Sepa el amable lector que la gran mayoría de los programas dados, problemas propuestos y fragmentos de código aquí proporcionados fueron tomados del tomo uno de esta serie, adaptados a los nuevos objetivos pedagógicos, todos comprobados por el autor y sincronizados con sus contrapartes del tomo anterior. Dispense pues cualquier gazapo que aparezca.

Una palabra sobre los ejercicios

Reinventar la rueda es una expresión que se utiliza para describir situaciones en las que el esfuerzo para solucionar un problema aparentemente nuevo es redundante o carente de sentido, puesto que la solución existe, pero se desconoce o se evita. (Wikipedia en español)

Partiendo de que el lector sabe programar al menos un poco en ANSI C, los ejercicios fueron cuidadosamente diseñados para que se apoyara en conocimientos previos obtenidos durante el estudio del tomo uno de esta serie, pero son autocontenidos por si no conoce ese primer tomo o no se le tiene a mano. Los programas resueltos y los ejercicios propuestos fueron tomados para ser tratados según el contenedor bajo estudio, pero identificados para una posible comparación, buscándose en lo posible que las salidas coincidieran con las de sus contrapartes.

[2] O sabe programar en C++ como Súper C, que el autor define: *cuando el programador usa un compilador C++ y sus características propias dentro del lenguaje C, pero sin aplicar a fondo los fundamentos de la programación orientada a objetos o la STL*. Ello beneficia a un programa ANSI C con la mejor seguridad que brinda la compilación de C++ y sus facilidades de uso de las E/S, manejo de archivos, dominio de complejidades, etc.

EODA[3] esta línea de trabajo es lo más deseable, ya que así se puede constatar la superioridad del empleo de la STL contra la ardua labor de concebir nuevas y nuevas soluciones frente a cada situación problémica enfrentada, reinventando una y otra vez la bendita rueda.

Este primer tema se dedica a presentar el sistema de desarrollo basado en el ya conocido IDE Code::Blocks, pero con un estilo algo más relajado, acercando al lector al tipo de programación distintiva del ANSI C, aunque sin perder nunca de vista la claridad de lectura humana del código. El segundo tema revisa brevemente la historia detrás de la STL y analiza su filosofía de aplicación. El tercer tema se consagra a los elementos de la programación genérica, y desarrolla un sistema de plantillas que luego será utilizado en el resto del libro. El cuarto tema es el más largo y presenta el uso de la STL como sustituto de código manual por uno predefinido, aplicado a la medida de un problema, utilizando varios contenedores y algoritmos en ejemplos puntuales del tomo anterior. El quinto y último tema estudia la ampliación de los conceptos básicos para poder gestionar estructuras complejas.

El texto cuenta con 13 ilustraciones, 30 tablas de consulta, 5 problemas auxiliares, 20 problemas resueltos y termina con un conjunto de 30 ejercicios propuestos.

El texto exige un esfuerzo extra del lector y el autor le pide excusas de antemano

¿Cómo escribir buen código?

La elegancia en la escritura y la claridad en la lectura de un programa no es un asunto de gustos: ¡es una necesidad del oficio. Ver más adelante el Zen de Python, que entre otras cosas dice que la legibilidad cuenta. (Introducción al C++, tomo 1, pág. 15) Recordar que un buen código es aquél que posee **(a)** las características del Zen de Python; **(b)** está correctamente sangrado con estilo consistente a lo largo del trabajo; y **(c)** además posee las cualidades de ser:

1. Íntegro, pues da respuestas correctas a los cálculos hechos sobre los datos suministrados.
2. Preciso, ya que los valores calculados responden a las ecuaciones matemáticas usadas.
3. Fiable, porque hace lo planeado durante su ciclo entero de vida.
4. Claro, pues es legible por un programador afín.
5. Modular, al estar sus partes constituyentes auto contenidas en funciones/clases C++ lo mejor formadas posible.
6. Simple, ya que es modularmente corto, sencillo y entendible.
7. General, a causa de estar hecho en base a lineamientos que responden al entorno del problema.
8. Robusto, si es resistente a todo tipo de errores.
9. Tipificado, si estandariza la forma de su uso, la toma de los datos y la presentación de los resultados obtenidos.
10. Portable, ya que es ejecutable en diferentes plataformas con un mínimo de cambios, o ninguno.
11. Eficiente, pues costó menos recursos que lo planeado.
12. Eficaz, porque está descrito por sus usuarios al menos como bueno.

Un código mientras más posea de estas condiciones, mejor cumple con ser bueno

Pero ahora que ya se conoce algo el lenguaje, se pueden poner dos o más instrucciones juntas, obviar las llaves en construcciones que caben en una o dos líneas y relajar un poco la coherencia de un módulo, siempre cuidando de mantener un código pitónico, bien legible.

[3] EODA: En Opinión Del Autor…

La Filosofía del Zen de Python

Según (Wikipedia en español), el código que sigue los lineamientos de legibilidad y transparencia del lenguaje de programación Python, se dice que es pitónico y garantiza un código razonablemente libre de errores. El programador que desconoce o ignora estos conceptos generalmente produce código ofuscado en mayor o menor medida. Seguidamente se muestran esos principios:

- Bello es mejor que feo.
- Explícito es mejor que implícito.
- Simple es mejor que complejo.
- Complejo es mejor que complicado.
- Plano es mejor que anidado.
- Disperso es mejor que denso.
- La legibilidad cuenta.
- Los casos especiales no son tan especiales como para quebrantar las reglas.
- Lo práctico gana a lo puro.
- Los errores nunca deberían dejarse pasar silenciosamente...a menos que hayan sido silenciados explícitamente.
- Frente a la ambigüedad, rechaza la tentación de adivinar.
- Debería haber una —y preferiblemente sólo una— manera obvia de hacerlo.
- Aunque esa manera puede no ser obvia al principio, a menos que usted sea holandés.
- Ahora es mejor que nunca.
- Aunque nunca es a menudo mejor que ahora mismo.
- Si la implementación es difícil de explicar, es una mala idea.
- Si la implementación es fácil de explicar, puede ser que sea una buena idea.
- Los espacios de nombres son una gran idea ¡Hagamos más de esas cosas!

Caracteres en español

En el sistema por donde se montó el curso, la salida de caracteres fuera del ámbito más estrecho del código ASCII (de los números 32 a 127) no es afortunada. Cuando un escape presenta tres números, C++ los interpreta en formato octal. Por ejemplo, la á tiene el código ASCII decimal 160, pero para sacarla a consola dentro de un literal (texto entre comillas simples o dobles), se debe poner la secuencia de escape \240 octal (160 decimal.) En el texto se emplean los siguientes códigos embebidos en las cadenas de los programas:

Tabla 1. Códigos de escape en formato octal para el idioma español

\240 es la á	**\241 es la í**	**\243 es la ú**	**\244 es la ñ**
\265 es la Á	**\326 es la Í**	**\351 es la Ú**	**\245 es la Ñ**
\202 es la é	**\242 es la ó**	**\201 es la ü**	**\250 es el ¿**
\220 es la É	**\340 es la Ó**	**\232 es la Ü**	**\255 es el ¡**

Inclusión de ficheros

La directiva **`#include`** dice al preprocesador que abra el fichero indicado e incluya su contenido en ese mismo lugar, por eso en esta parte de la serie y para subrayar la aplicación de los componentes de la STL, se tratarán de incluir los ficheros

en el lugar adecuado, al menos, siempre que se pueda, puesto que, en C++ al igual que en su ancestro, para usar algo no contenido en su núcleo básico (por ejemplo, un componente auxiliar o una facilidad), debe "conocerse" primero.

En (Introducción al C++, tomo 1, págs. 27-28) se plantea el orden de inclusión correcto: los primeros son los ficheros de cabecera hechos por el programador, llamados entre comillas; el resto va llamado entre paréntesis angulares. Entre ellos están (si los hay) los que sean incluidos directamente por el IDE en uso (aquí no se usan) y los de las bibliotecas de terceros tales como STLPort, Boost o Blitz++ (el texto no las usa.) Luego van los propios del C++ (que si se usan) y si no queda de otra, los del ANSI C en último lugar (el texto tampoco los usa), siempre recordando que estas últimas no deben ni mezclarse, ni usarse en código nuevo.

Salida formateada

En muchas ocasiones se desea poner la salida en un estado especial. Por ejemplo, salidas financieras que deben poner los números alineados, columnas nítidas de datos o de nombres, tablas visualmente placenteras, etc.

Ud. puede controlar cómo un programa formatea la salida usando métodos de la clase **`ios_base`** y manipuladores (funciones que pueden ponerse en el flujo de inserción) definidos en las cabeceras **`iostream`** e **`iomanip`**. Estos métodos y manipuladores le permitirán controlar la base numérica, la anchura del campo de salida, el número de lugares decimales exhibidos, el sistema empleado en mostrar valores en punto flotante y otros elementos. (Prata, C++ Primer Plus, p. 1078)

Tabla 2. (simplificada) Manipuladores de flujo

Manipulador	Propósito
`boolalpha`	Activa la salida booleana en inglés: **`true`** o **`false`**
`nobooalpha`	Desactiva la salida booleana; salen números: **`0`** ó **`1`**
`showpoint`	Activa la salida del punto decimal en números de punto flotante.
`noshowpoint`	Desactiva la salida del punto decimal en números de punto flotante.
`showpos`	Activa la salida del signo **`±`** antes del número.
`noshowpos`	Desactiva la salida del signo **`±`** antes del número.
`noskipws`	Tiene presente los caracteres en blanco a la entrada de datos —por omisión.
`skipws`	Ignora los caracteres en blanco a la entrada de datos.
`dec`	Activa la salida en formato decimal —por omisión.
`oct`	Activa la salida en formato octal.
`hex`	Activa la salida en formato hexadecimal.
`endl`	Da salida a una nueva línea y limpia el búfer.
`fixed`	Valores mostrados en notación "normal" —por omisión.
`scientific`	Valores mostrados en notación científica.
`right`	En la salida, números alineados a la derecha —por omisión.
`left`	En la salida, números alineados a la izquierda.
`internal`	El sistema escoge el formato de alineación.
`endl`	Saca una nueva línea y limpia el flujo.
`setiosflags(flags f)`	Activa las banderas de formateo especificadas en **`f`**
`resetiosflags(flags f)`	Desactiva las banderas de formateo especificadas en **`f`**
`nouppercase`	Saca los símbolos de los formatos octal y hexadecimal en minúsculas —por omisión.
`uppercase`	Saca los símbolos de los formatos octal y hexadecimal en mayúsculas
Para usar los manipuladores set* hay que incluir a la biblioteca <iomanip>	
`setfill(c)`	Llena el ancho del campo de salida con el carácter **`c`**; espacio en blanco, por omisión
`setprecision(d)`	Fija la cantidad **`d`** de dígitos decimales mostrados en los números de punto flotante.
`setw(e)`	Fija el ancho de salida en **`e`** espacios.

Algunos manipuladores una vez fijados, están en acción hasta que se les desactiva; por ejemplo, **showpoint** y **noshowpoint**. A **setw** hay que llamarlo cada vez que se hace una operación de salida. Otros accionan hasta que son sustituidos por uno similar, por ejemplo, **scientific** da la salida en formato científico hasta que es sustituido por **fixed**, que da la salida normal. También se pueden controlar con los métodos **setf** y **unsetf** de **cout**. Por ejemplo:

```
cout.setf(ios_base::scientific);    // fija el manipulador
cout.unsetf(ios_base::scientific);  // libera el manipulador
```

Un consejo

A veces se hacen salidas en distintos puntos del programa y aparece un efecto colateral traducido en que un formateo anterior está afectando adversamente a otra salida. Para evitar esto se recomienda aplicar la técnica del ladrón: *al salir, poner todo como estaba al entrar*. Suponga que desea su salida muestre el punto decimal en los valores y tenga precisión hasta las milésimas. Entonces:

```
ios::fmtflags oldFlags = cout.flags();                                  // formato original
cout << setiosflags(ios::fixed | ios::showpoint) << std::setprecision(3); // formato deseado
```

Luego, al salir del módulo, restaura los valores originales, para que no influyan en salidas subsiguientes:

```
cout.flags(oldFlags); // repone el formato original
```

La presentación de materias

Desafortunadamente y con cierta frecuencia a medida que se avanza, se presentará algún tópico requiriendo de conocimientos que no han sido explicados todavía, producto de que la enseñanza en general, y la de la computación en particular, raramente avanza en línea recta, antes bien, avanza en espiral dialéctica y ascendente. (Introducción al C++, tomo 1, pág. 15)

Y este es uno de esos casos. O se usan facilidades que aún no se han estudiado, o se estudian primero esas facilidades. Si se hiciera esto último, se tendría que desarrollar previamente las técnicas brindadas por la STL y al final, retomar las plantillas sobre ejemplos ya hechos, duplicando código, alargando innecesariamente el espacio del texto y comprometiendo la claridad de exposición.

El autor se decantó por la primera opción

---

2 - ¿QUÉ ES LA STL?

> *La STL define poderosos componentes reutilizables, basados en plantillas, que implementan muchas estructuras de datos y algoritmos comunes utilizados para procesar esas estructuras de datos. (Deitel & Deitel, pág. xx)*

Operacionalmente hablando, es el otro objetivo principal del estudio introductorio en este segundo texto de la serie. Mediante su uso el programador podrá resolver sus problemas creando con relativa facilidad y alta eficiencia un código bueno y minimalista con el conjunto de las cualidades ya mencionadas en la página 10, más la rapidez de escritura.

- ✓ STL es el acrónimo de *Standard Template Library* (Biblioteca Estándar de Plantillas en español.)

La STL está compuesta mayormente de plantillas de clases, estructuras de datos y algoritmos, y ellos pueden ser aplicados a casi cualquier tipo de dato y ahí radica su belleza, su facilidad de uso y su inmensa potencia.

Brevísima historia

Las bases del lenguaje *C Con Clases* fueron formalmente creadas por Bjarne Stroustrup como parte de la tesis para su doctorado, en un periodo de tiempo que abarcó desde mediados de los años '70 hasta principios de los '80. El lenguaje resultante —conocido en la comunidad como *La Bestia*— fue renombrado C++ por Rick Mascitti al ser utilizado en 1983 por primera vez fuera de los Laboratorios Bell de AT&T. La STL se le incorporó unos 12 años después, a finales del '94 y principios del '95, y el primer estándar salió a finales del '97 y principios del '98, hace ya casi ¡23 años a la fecha de hoy!

La STL fue inicialmente creada por los desarrolladores Alexander Stepanov y Meng Lee (empleados en ese entonces de la compañía Hewlett-Packard) y se basó en trabajos previos sobre el lenguaje ADA hechos por Stepanov y el profesor David Musser (que en aquel momento enseñaba computación en el Instituto Politécnico Rensselaer) aplicando técnicas propiedad de la compañía Silicon Graphics. Fue puesta a disposición pública por ambas compañías y usada por el comité ANSI/ISO nombrado como X3J16/WG21, a fines del '94.

Stroustup continúo trabajando en el diseño del lenguaje y al día de hoy es un miembro influyente del comité del estándar del ISO C++

Oficialmente la norma contempla la STL como parte integrante de la biblioteca estándar de C++, aunque en la práctica estén lógicamente separadas.

Objetivo

El objetivo de su creación abarcó dos ideas básicas: adicionar al lenguaje el paradigma de la programación genérica y facilitarle una gran capacidad de abstracción sin pérdida notable de eficiencia. Hoy día los compiladores C++ vigentes están perfeccionados para minimizar cualquier penalización de abstracción derivada de un uso intensivo de la STL. Este enfoque proporciona polimorfismo en tiempo de compilación que es mucho más eficiente que el tradicional en tiempo de ejecución.

- ✓ Polimorfismo, es la cualidad representada por "una interfaz, muchos métodos", significando que con una llamada a un método o una función (una interfaz) se pueden resolver muchos problemas mediante la aplicación de algoritmos diferentes (muchos métodos) que comparten esa misma llamada.

Los tres tipos de polimorfismo vigentes son:

1. Polimorfismo en tiempo de ejecución. Aparece cuando se escriben funciones sobrecargadas y el compilador escoge cuál llamar por los datos que el programa suministra a su interfaz: un método, pero muchas interfaces. Se usó en el primer tomo de esta serie.
2. Polimorfismo en tiempo de compilación. Surge cuando el compilador particulariza la plantilla al dato que el programa suministra a su interfaz y la integra allí donde aparece: un método, pero muchos tipos de datos. Se usará y estudiará en este tomo de la serie.
3. Polimorfismo de clases. Es parte definitoria de la OOP y se usa con las clases: una llamada común (método) a muchas clases, historia para el siguiente tomo…

Al perfeccionar el desempeño de C++, una de las tantas intenciones por parte de Stroustrup fue extenderle mecanismos que permitieran la manipulación de objetos. Posteriormente, durante la plasmación del primer estándar aparecieron las facilidades de la programación genérica, de ahí su desfase en el tiempo de aparición, a poco más de 10 años de la creación

de C++, que trajo como consecuencia negativa una desincronización en el estilo de nombrado de funciones de la biblioteca estándar y algoritmos de la biblioteca estándar de plantillas que son comunes[4].

Por todo lo anterior se dice que el C++ es un lenguaje de programación multiparadigma (tres paradigmas de programación: imperativa, genérica y orientada a objetos) e híbrido, porque el programador de C++ sólo está obligado a usar el primero de ellos (como se hizo antes, en el primer tomo de esta serie); el resto lo explota si lo sabe, o lo desea hacer. El autor concuerda completamente con (Prata, C++ Primer Plus, p. 9) en que…*Esta triple herencia es una bendición y un problema a la vez. Hace muy potente al lenguaje, pero eso también significa que hay mucho que aprender*.

El estándar ha intentado definir tanto el contenido, como el desempeño de C++, proporcionando facilidades para **(a)** enseñarlo a todos los que lo deseen dominar; **(b)** enseñarlo a todos los niveles, desde el principiante hasta el gurú, que aquí siempre tiene algo que aprender; **(c)** emplearlo en la creación de nuevas aplicaciones o en el mantenimiento de las ya establecidas; y **(d)** exportar eficazmente su código a diferentes plataformas; todo esto con un mínimo de esfuerzos.

Y EODA ha triunfado, puesto que al correr de los años ISO C++ se ha convertido en el lenguaje *de facto* a conocer, desplazando en mayor o menor grado a todos sus rivales.

La estabilidad y portabilidad[5] conseguidas por C++ propician la construcción de aplicaciones profesionales que son:

1. Consistentes, ya que presentan interfaces y comportamientos ampliamente tipificados.
2. Eficientes, puesto que entregan un buen manejo de la rapidez tanto en la producción de código, como en la ejecución de sus aplicaciones.
3. Eficaces, porque hacen relativamente poco consumo de todo tipo de recursos, y se caracterizan por su alto grado de fiabilidad, reusabilidad y buena mantenibilidad.

La tipicidad de C++ está dada en su biblioteca estándar (**`cstdlib`**), que provee al núcleo del lenguaje los componentes fundamentales para el manejo de las operaciones de E/S, cadenas de caracteres, estructuras básicas de datos, algoritmos de ordenamiento, búsqueda y mezcla, y soportes para la computación numérica y la internacionalización.

Pero cuando la STL se integró al desarrollo de C++, constituyó una especie de columna central de carga para aquella, a la cual brinda soluciones que, al generalizar múltiples contenedores y al aplicar algoritmos genéricos eficientes y modernos a muchas colecciones de datos, le permiten al programador usar las mejores innovaciones en esta área, sin tener la obligación de saber cómo trabajan. Y esto es muy importante, porque como ya se planteó:

- ✓ Por otro lado, la STL suministra numerosos recursos para viabilizar el desempeño del humilde codificador, cuyos conocimientos requeridos para un trabajo dado ya no les son forzados a obtenerlos en el estudio formal de los cursos de Ciencias de la Computación, y se acercan cada vez más a los de un técnico, eso sí, altamente calificado. (Introducción al C++, tomo 1, pág. 157)

El impacto de C++ en el mundo digital es inmenso. Muchos de los programas, aplicaciones, juegos y aun, sistemas operativos son codificados usando C++. Por ejemplo, todas las aplicaciones mayores de Adobe, como Photoshop, InDesign, etc. son desarrolladas en C++. Verás que el browser con que surfeas la Internet está escrito en C++, así como Windows 10, Microsoft Office y la piedra miliar del buscador de Google. (Black Dog Media: Linux Tricks and Tips, pág. 40)

Si el lenguaje no es fácil de manejar, tampoco es imposible. Habrá que esforzarse más, pero la recompensa será muy satisfactoria para el estudioso lector. ¡Adelante!

[4] Es fácil ver los árboles cuando no se ve bien al bosque, pero se le asegura al atento lector que puede vivir perfectamente con ello.

[5] La portabilidad viene en dos formas: *Compila una vez, corre donde sea*, como la da el Java y *Escribe una vez, compila donde quiera*, como la da el C++.

Componentes fundamentales

Todo el concepto [de la STL] se basa en separar datos y operaciones. Los datos son administrados por los contenedores, las operaciones son configuradas por algoritmos adaptables. Los repetidores son el elemento de unión entre ambos: permiten la interacción de cualquier algoritmo con cualquier contenedor. (Josuttis, p. Cap. 5.1)

Aun cuando la STL suministra variadas herramientas al programador, está basada fundamentalmente en tres componentes y su cooperación interrelacionada: los contenedores, que contienen todo tipo de valores; los algoritmos, que procesan esos valores; y los repetidores, que permiten hacer ciclos a los algoritmos sobre los valores contenidos en las colecciones o contenedores.

- ✓ Contenedores: plantillas de clases que implementan las estructuras de datos más comunes.
- ✓ Algoritmos: plantillas de funciones que realizan operaciones comunes sobre los valores de los contenedores.
- ✓ Repetidores: punteros especializados, usados por los algoritmos para iterar a través de los valores de los contenedores.

El contenido aquí presentado es (o al menos, pretende ser) apropiado para un texto introductorio de reducidas dimensiones, pero el autor reconoce que el manejo de la STL en verdad no es nada elemental. Aquí será usada sin ahondar demasiado y sin abarcarla en toda su amplitud, pues el objetivo central de esta parte no contempla su estudio en profundidad. Para eso se recomienda el estudio del tercer tomo de esta serie y entre la bibliografía dada y EODA, en este orden, a (Prata, C++ Primer Plus), (Halterman), (Josuttis), (Schildt) o (Lischner)

Este texto enseñará al lector a trabajar un ISO C++ potenciado: como si fuera un ANSI C con esteroides

Contenedores

Un contenedor o colección es un constructo que puede contener varios valores. Los contenedores de la STL son homogéneos, es decir, guardan valores del mismo [desconocido] tipo. (Prata, C++ Primer Plus, p. 978) Hablando en un nivel abstracto, los contenedores son y siempre han sido parte de los lenguajes de programación de Alto Nivel. Cada lenguaje principal de hoy día tiene al menos cuatro contenedores básicos: uniones, estructuras, arreglos y conjuntos o como se les llame ahí. Los modernos dedicados exclusivamente a un área o tarea específica, por lo general presentan contenedores más potentes y especializados, tales como árboles, bultos, grafos, etc.

En C++, los contenedores de la STL son implementados como plantillas de clase, así que pueden almacenar casi cualquier cosa. Sólo se requiere el cumplimiento de tres condiciones: **(1)** el objeto almacenado debe poder comportarse de a últimas como un valor nativo simple; **(2)** los valores deben poder ser creados y asignados libremente; y **(3)** un original y su copia deben ser iguales byte por byte, aunque estén en localidades de memoria diferentes. Esto se cumple casi que automáticamente.

Hay varios tipos de contenedores. Por ejemplo, la clase **`vector`** suministra un arreglo dinámico, las clases **`deque`** y **`list`** entregan una lista doblemente enlazada con implantación diferente para cada una, etc. La elección depende de las características y el comportamiento requeridos por el problema. Cada contenedor tiene sus ventajas y desventajas frente a una situación puntualmente dada, permitiéndole resolver diversos requisitos y en este sentido, las uniones, estructuras y arreglos de C++ —elementos inherentes al lenguaje, como ya se dijo— también son contenedores.

Como corolario, un problema usualmente presenta más de una solución mediante el uso de la STL. Desde luego, el autor solo ofrece una; queda de parte del curioso lector el investigar cada ejemplo por su cuenta. Una de las mejores formas de aprender alegre y eficazmente: ¡mediante la experimentación propia!

Tipos de contenedores

Amable lector, los contenedores desafortunadamente no serán analizados a cabalidad, pues por la extensión y profundidad del tema, se comprometería tanto la condición introductoria, como el tamaño limitado de este texto. Se muestran en una tabla simplificada y sencilla, compilada de los trabajos de (Josuttis) y (Lischner).

Tabla 3. Tipos de contenedores

Contenedores	Características	Clases

Secuenciales	Conjuntos en los que cada elemento posee una posición en el contenedor que depende exclusivamente del momento y lugar de su inserción.	`vector`
		`deque`
		`list`

Asociativos	Conjuntos ordenados por diseño en los que la posición de cualquier elemento sólo depende del valor real de su clave y no está influenciada por el momento y lugar de su inserción.	`set`
		`multiset`
		`map`
		`multimap`

Adaptados	Parten del contenedor **`deque`**, aunque debido a su trabajo especializado presentan una interfaz restringida y optimizaciones automáticas.	`stack` `queue`
Utilitarios	Permite tratar dos valores heterogéneos como si fueran uno. Usado en general por muchas plantillas de la STL y específicamente por mapas y mapas múltiples, también se aplica en funciones que devuelven dos valores.	`pair`
	Aunque algunos autores lo ven al revés, es una generalización del anterior, pudiendo contener hasta 10 elementos que son empacados y tratados como si fueran uno.	`tuple`

Seudocontenedores	Son similares a los otros, pero les falta al menos una característica básica que impide sean verdaderos contenedores.	`string`
		`array`
		`valarray`

Nota: las cadenas **`string`** el **`array`** y el **`valarray`** se clasifican como seudocontenedores porque, aunque se les pueden aplicar repetidores y algoritmos de la STL, todos carecen como mínimo de las funciones básicas de un contenedor para el acceso directo a la cabeza (**`push_front()`**) o a la cola (**`push_back()`**), o no poseen el concepto de **`begin`** o **`end`**.

¿Cuándo usar los contenedores?

La tabla que sigue da las guías más generales al escoger un contenedor para resolver un problema, siempre advirtiendo que usualmente se pueden aplicar a un problema dos o más de ellos. Pero el atento lector las debería de tener en cuenta durante la producción de código nuevo porque una aplicación bien ajustada al problema puede simplificar bastante el código a producir.

Tabla 4. Guía general para escoger un tipo de contenedor

Contenedor	Problema a resolver
`vector`	Fue pensado para sustituir al arreglo estándar C++, pero trabaja con casi cualquier secuencia, pues posee la estructura interna más sencilla de todos los contenedores: un arreglo en la tienda libre. Tiene acceso aleatorio y el entrar/sacar valores por la cola es simple, flexible y muy rápido. No introduce por la cabeza, y a veces hay que reajustar su tamaño manualmente.
`deque`	Muy similar a un vector y presenta casi su misma interfaz, posee la estructura interna de un conjunto de colas doblemente enlazadas, pero tiene acceso aleatorio y el entrar/sacar valores por ambos lados, cabeza y cola, es simple, flexible y muy rápido. Y no hay que reajustar su tamaño.
`list`	Posee la estructura interna de una cola doblemente enlazada y sirve para inserciones, remociones y reordenamientos a voluntad del programador. Tiene acceso secuencial y para ver un valor hay que visitarla, pero tomar o sacar valores por cualquier lado es simple, flexible y muy rápido.
`array`	Fue pensado para otorgar a un arreglo C++ la eficiencia de uno clásico y la seguridad de la STL; es estático, de tamaño fijo y usa la pila, pero responde a los algoritmos y repetidores de la biblioteca como si fuera un vector.
`valarray`	Para trabajos matemáticos hechos sobre un vector lineal numérico.
`stack`	Especializado para trabajar estructuras de pila o FIFO: inserción y remoción por la cabeza
`queue`	Especializado para trabajar estructuras de cola o LIFO: inserción por la cola y remoción por la cabeza.

set	Usado cuando la prioridad es la búsqueda muy rápida de elementos del contenedor.
multiset	Si frente a la condición anterior, los elementos pueden estar repetidos.
map	Usado cuando se van a procesar pares de valores únicos del tipo clave/valor
multimap	Usado cuando se van a procesar pares de valores repetidos del tipo clave/valor, o cuando se va a crear un diccionario o algo parecido.
tuple **pair**	Si se van a procesar dos o más valores heterogéneos, pero relacionados entre sí como un todo y no se desea recurrir a una estructura (**struct**) o a una clase (**class**). También sirven como parámetros para pasar valores heterogéneos —relacionados o no— a métodos o funciones.

Notas:

- Acceso aleatorio se consigue con el operador de desplazamiento **[]** y un índice adecuado.
- Acceso secuencial significa que para encontrar un valor dado hay que "visitar" el contenedor.
  - ✓ En muchas ocasiones se presenta la tarea de leer desde un contenedor una cantidad de elementos no definida, o hacerle algo a cada uno de los elementos de una secuencia, o hacer algo con todos esos elementos. A estos recorridos, se les llama *visitar los elementos del contenedor*.

¿Cuál es el contenedor más apropiado?

La norma C++98, en su sección 23.1.1 aconseja sobre la preferencia de contenedores. Dice: el vector es el tipo de secuencia que debe ser empleado por omisión…la deque es la estructura de datos a elegir cuando la mayoría de las operaciones de inserción o de borrado son hechas al inicio o al final de la secuencia. (Sutter, More Exceptional C++, pág. Item 7)

Al escoger un contenedor para trabajar arreglos el autor destaca esta línea de trabajo: si es un arreglo simple preferir un **vector**, pero si el tamaño se sabe de antemano y no es excesivo, preferir un **array**; si es un trabajo matemático considerar un **valarray**; si las operaciones de E/S pueden ser hechas por ambas puntas, preferir una **deque**; para pilas usar el **stack**; para colas usar la **queue**.

Aunque la norma no lo estipula, las listas (**list**) se estructuran como doblemente enlazadas, pero se diferencian en varios puntos de los vectores y las colas dobles antes mencionados:

1. No hay acceso aleatorio: para ver un valor hay que recorrer la lista. Esta operación es relativamente lenta.
2. La inserción o remoción de valores es muy rápida en cualquier posición.
3. No necesita de operaciones para ampliar o relocalizar capacidades.
4. Brinda muchos métodos para el movimiento de valores en versiones que son más abstractas que los algoritmos con el mismo nombre a los cuales sustituyen.

Si el trabajo se enfoca en esos puntos vale la pena considerar las listas como una alternativa viable.

Repetidores

Tema de comprensión nada fácil, que EODA requiere de esfuerzo extra para ser asimilado a cabalidad, los punteros conforman una de las herramientas más potentes de C++, principalmente porque dan acceso y control sobre la tienda gratis, pero también son muy peligrosos, porque permiten realizar acciones hostiles al programa… (Introducción al C++, tomo 1, pág. 127)

Los repetidores son objetos que pueden moverse a través de los elementos de una secuencia, mediante una interfaz común que les llega a través de los punteros ordinarios. Modelados según el concepto de abstracción pura, *cualquier elemento que opere como un repetidor, incluyendo índices y punteros,* ***ES UN REPETIDOR***.

Tabla 5. Acciones de los repetidores

Operador	Acción
`*`	Accede al elemento a que apunta el repetidor. Es indirección
`++`	Avanza el repetidor al próximo elemento. Es aritmética de punteros.
`--`	Retrocede el repetidor al elemento anterior. Es aritmética de punteros.
`==`	Verdadero si dos repetidores se refieren a elementos iguales del contenedor.
`!=` `not_eq`	Verdadero si dos repetidores se refieren a elementos distintos del contenedor.

Los repetidores son realmente punteros disfrazados que están bajo el control de C++, siendo tan peligrosos como sus hermanos, pero sólo en el sentido de que, si se aplica un repetidor incorrecto a un algoritmo, o se aplica incorrectamente —sobre todo indireccionando al nodo que marca el fin de una secuencia— se obtiene un comportamiento indefinido en la aplicación, haciendo inestable a la aplicación y con un resultado casi siempre catastrófico.

Tipos de repetidores

Se usan para moverse a través de los elementos presentes en los contenedores. Su principal ventaja es la pequeña interfaz común para cualquiera de sus usuarios, independientemente de su estructura interna, porque cada tipo de contenedor posee su repetidor propio, que trabaja correctamente al ser invocado. Por ejemplo, para ordenar una secuencia se necesitan repetidores que realicen acceso aleatorio, buscando eficiencia. A continuación, se listan sus habilidades.

Tabla 6. Tipos de repetidores

Repetidor	Habilidad	
De salida	Permite escribir una secuencia completa en un pase. El repetidor puede avanzar al próximo elemento, pero no retroceder. La indirección sólo permite asignar valores. Este tipo de repetidor no admite comparaciones. Está presente en listas, sets, multisets, mapas y multimapas.	
De entrada	Permite leer una secuencia completa en un pase. El repetidor puede avanzar al próximo elemento, pero no retroceder. La indirección retorna un valor que puede ser leído, pero no alterado. Está presente en listas, sets, multisets, mapas y multimapas.	
De avance	Permite leer o escribir valores de una secuencia. El repetidor puede avanzar al próximo elemento, pero no retroceder. Puede sustituir a cualquiera de los dos anteriores. Está presente en listas, sets, multisets, mapas y multimapas.	
Bidireccional	Similar al anterior, pero se puede mover en ambas direcciones. Se encuentra en vectores, deques, cadenas C++ y arreglos STL	
De acceso aleatorio	Es bidireccional, pero además soporta el operador de desplazamiento `[]` o el método `at()` para acceder a cualquier índice en la secuencia, y permite el avance a saltos mediante la aritmética de punteros. Puede sustituir o ser sustituido por un puntero común y corriente. Si se restan dos de ellos, el valor obtenido es la distancia o zancada que les separa.	
Auxiliar	`n = distance(fin, ini)`	Devuelve la distancia o zancada entre `fin` e `ini`, en el rango `[ini:fin]`.
	`advance(i, n)`	Avanza `n` posiciones al repetidor `i`. Recordar que la primera posición corresponde al índice cero, la segunda al uno, etc. ¡Cuidado! Fácilmente puede pasarse de límites.

Notas:

- Menos el repetidor de salida, los demás pueden ser constantes y se invocan mediante el prefijo `const_` agregado a su nombre. Si se les indirecciona su valor, éste también es constante. Por lo demás, operan igual a los tabulados.
- Siempre que se trate con contenedores cuyos valores no se deben alterar, preferir un repetidor constante.
- Siempre preferir el operador de preincremento antes que el de posincremento, por ser este último marginalmente menos eficiente, ya que necesita guardar el valor anterior en una variable temporal y finalmente, destruirla.
  - ✓ Así pues, para cualquier repetidor pos (y cualquier tipo abstracto de dato) debería preferir ++pos (OK y más rápido) antes que pos++ (OK, pero no tan rápido.) (Josuttis, p. Cap. 7)

Algoritmos

En las ciencias de la computación y disciplinas relacionadas, un algoritmo es un conjunto prescrito de instrucciones bien definidas, ordenadas y finitas que permite llevar a cabo una actividad mediante pasos sucesivos que no generen dudas a quien deba hacer dicha actividad. (Wikipedia en español) Los algoritmos de la STL están dados por plantillas de funciones globales que usan repetidores para procesar los elementos de uno o más conjuntos, y pueden aplicar una buena cantidad de técnicas a cada uno de esos elementos. Al ser funciones globales se codifican de una vez, pero trabajan con contenedores arbitrarios porque como ya se dijo, la interfaz del repetidor es común para todos.

La programación genérica de los algoritmos es considerada como un paradigma genérico-funcional donde los datos y las operaciones son separados en partes que pueden interactuar a través de cierta interfaz, aunque debe saberse que:

- El uso no es intuitivo.
- Algunas combinaciones de estructuras de datos y algoritmos no pueden interactuar. Si es el caso, el compilador avisará.
- Una cierta combinación *contenedor—repetidor—algoritmo* podría llevar a un mal desempeño aun cuando funcionara, degradando la eficiencia de la acción. Controlar eso es responsabilidad del programador.

Tipos de algoritmos

Los algoritmos se agrupan en siete categorías que son:

Tabla 7. Tipos de algoritmos

Algoritmo	Aplicación
Invariantes	Permiten contar elementos, buscar valores en la secuencia, valores extremos, zancadas y comprobar igualdades y desigualdades.
Operan sobre datos numéricos	Calculan sumatorias, productos interiores y transforman valores relativos en absolutos y viceversa.
Modifican datos	Posibilitan copias, transformaciones, mezclas, asignación o generación de valores e intercambios y reemplazos.
Remueven datos	Remueven valores, sean cualesquiera y estén duplicados o no.
Cambian datos de lugar	Revierten el orden, los rotan, permutan o barajan.
Ordenan datos	Varios tipos y criterios de ordenamiento total o parcial, más la transformación de secuencias en bultos (**heap**) y su trabajo con ellos.
Ordenan datos por rangos	Búsqueda total o parcial, comprobación de la presencia de valores en la secuencia, mezcla de dos secuencias, y cuatro operaciones de álgebra de conjuntos (unión, intersección, diferencia y diferencia simétrica.)

Nota: en este contexto, **heap** no se refiere a una forma de trabajo con la memoria, sino a una estructura de datos tipo árbol débilmente ordenado. Si es un **heap** máximo cada nodo padre tiene un valor mayor que el de cualquiera de sus nodos hijos; si es mínimo, es al revés.

Es importante aprender tanto las ventajas como las desventajas del uso de los algoritmos de la STL para aprovecharse correctamente de su potencia, tópico cuyo estudio exhaustivo, por su extensión y profundidad, desafortunadamente queda fuera del alcance de un texto introductorio, como es de comprender. Entonces, ¿qué se hizo en el texto?

Cada vez que un programa resuelto emplea uno o más componentes de la STL, se explican

Además, el industrioso lector debe buscar sus propias fuentes y examinar lo más posible la literatura dada en esta parte, que puede ser conseguida —incluso mucho más actualizada— en la Internet, reservorio que almacena innumerables libros formalmente dedicados a este estudio.

3 – ELEMENTOS DE PROGRAMACIÓN GENÉRICA

> *Sólo después de que C++ logró algún éxito, fue que Stroustrup adicionó las plantillas, permitiendo la programación genérica. Y sólo después de que las características de las plantillas habían sido usadas y refinadas, se hizo aparente que quizá fueran una adición tan significativa como la OOP[6]. (Prata, C++ Primer Plus, p. 15)*

La programación genérica de funciones[7] es el primero de los dos objetivos pedagógicos principales del estudio introductorio de este segundo texto de la serie.

- ✓ La programación genérica de funciones las generaliza para que puedan usarse en más de una ocasión, parametrizando sus interfaces y retornando los datos de la forma más simple posible, evitando (en lo posible) detalles concretos.

Las plantillas de funciones y clases son el núcleo alrededor del cual se desarrolla la moderna biblioteca estándar ISO C++ (`cstdlib`): afecta las cadenas de caracteres, flujos de E/S, contenedores, iteradores, algoritmos y mucho más, deviniendo en patrones para crear, en fase de compilación, funciones o clases como instancias, similarmente a como una clase crea objetos como instancias, pero en tiempo de ejecución.

Una plantilla toma uno o más parámetros y cuando se va a instanciar, el compilador le suministra valores para sus parámetros. Teniendo implementaciones y/o comportamientos diferentes, según sus argumentos (valores pasados a los parámetros), esta forma de programar es llamada Programación Genérica.

Las plantillas de funciones nunca son más lentas que las funciones a las que sustituyen debido a que definen detalles concretos en tiempo de compilación, por tanto, aunque engrosan el código en muy poco, añaden mucha eficiencia a los programas, aportándoles rapidez de ejecución y produciendo de forma relativamente fácil un buen código con todas sus cualidades. Una vez depuradas y puestas a punto se usan muchas veces.

Programación de funciones

C++ es más que un mero lenguaje de programación orientado a objetos. El pleno poder de C++ se ve al programar con plantillas. Las plantillas son el núcleo alrededor del cual se desarrolla la biblioteca estándar: cadenas de caracteres, flujos de E/S, contenedores, iteradores, algoritmos y mucho más. (Lischner, p. Cap. 7)

Centrada en el polimorfismo paramétrico, una plantilla de función es la descripción de una función definida independientemente de sus parámetros, que pueden ser diferentes tipos de contenedores o distintos tipos de datos, pero que no se conocen al momento de la ejecución. Al pasar el tipo de contenedor y/o de dato como parámetro(s), el compilador produce un código específico y lo integra en el lugar dónde la llama el flujo del programa.

- ✓ Polimorfismo paramétrico: es cuando una función (o clase) está definida independientemente de sus parámetros, presentando una interfaz, pero muchos tipos de contenedores o datos.

Al aislar los elementos comunes de la solución de un problema, puede crearse la plantilla de una función capaz de manejar varias situaciones que enfrenta el programador…y alguna otra en que no ha pensado todavía. Esta panacea no viene sin inconvenientes. EODA los más notorios son:

---

[6] En lo adelante y como en el tomo anterior, OOP significa programación orientada a objetos.

[7] Genérico/a: que se refiere a un conjunto de elementos del mismo género. Por ejemplo, “gato” es un nombre genérico que sirve para referirse a cualquier gato. (VOX: diccionario de uso del español de América y España.)

- La plantilla no se define: se implementa. Por eso es considerada unidad atómica de código en C++. No se puede tener una parte de su implementación en un fichero y el resto en otro, por lo que todo el código se pone en un archivo de cabecera que no está pareado por la contraparte habitual, aunque se comprende que un archivo dado puede contener varias plantillas, como es el caso de la biblioteca estándar o el de este texto.
- Como en tiempo de codificación no se conoce su forma definitiva, el compilador sustituye la plantilla por una versión adaptada al entorno donde se llama y la inserta *in situ*. Si hay diez aplicaciones de una misma plantilla en un programa, la función será adaptada diez veces al código que las llama e incorporada allí donde aparece, engrosando el programa final, aunque realmente en muy poco.
- La programación genérica de clases no es nada trivial y se considera tradicionalmente difícil de escribir o interpretar. Es prácticamente un lenguaje *per se*. Esto se corrobora tratando de leer algunas plantillas de clases que aparecen en la biblioteca estándar de la instalación del curioso lector. Y aun cuando las plantillas de funciones son más simples, la dificultad inherente subsiste. Se requiere un esfuerzo extra por su parte.

En el tercer tomo de esta serie se dedica un tema completo al estudio de la programación genérica, pero ahora se ha de saltar adelante, porque se necesitarán a lo largo de este tomo. Si no se aplicaran, esta parte sería más larga y pesada de la cuenta. Es otro ejemplo de la enseñanza no lineal en este campo.

¿Cuándo usar las plantillas de funciones?

Cuando se le aplica el mismo algoritmo a un conjunto de funciones, donde la diferencia estriba en los tipos de contenedores o datos, porque si no, usualmente se les aplica la técnica de los parámetros con valores por omisión o la de sobrecarga de funciones, tópicos estudiados en el primer tomo de esta serie. La codificación de plantillas de funciones no sólo unifica la solución, si no que contribuye de forma natural a crear un buen código, además de aportar rapidez y seguridad a la conclusión del trabajo. La sintaxis de creación es[8]:

```
template typename<lista_de_Tipos> [inline]
tipo_de_retorno NOMBRE(lista_de_parámetros) {...}
```

El modelo especifica que **`NOMBRE`** es un nuevo tipo de función que toma una lista de parámetros (hay que ponerlos uno a uno, separados por comas, con sus tipos usualmente parametrizados) y el tipo de retorno declara el tipo de dato que devuelve. Aquí se destacan tres características:

1. Es costumbre llamar al tipo parametrizado de dato en una plantilla como **`T`**, y cuando hay más de uno, **`T1`**, **`T2`**, etc. Apartarse de ella significa obscurecer el código sin razón alguna.

2. La palabra-clave **`template typename`** especifica y enfatiza que el nombre que identifica es un nuevo tipo de dato y que no es lo mismo una platilla de funciones que una de clases.

Antiguamente al comienzo del código se declaraba **`template class`** indistintamente para clases o funciones, y si se está trabajando en mantenimiento retroactivo debería considerarse esta forma, pero la actualizada y a poner siempre en código nuevo es usar **`template typename`** para las plantillas de funciones y **`template class`** para las de clases, porque el compilador las distingue, de modo que una plantilla de clases no puede ser **`inline`**.

3. Independientemente de lo postulado en (Introducción al C++, tomo 1, pág. 79) las plantillas de funciones deben hacerse **`inline`** por dos razones de peso:

- No hay que repetirlas en ningún lugar porque el compilador ya las integra directamente al código.
- Al distribuir una aplicación para su uso, se entregan los ficheros-fuente de encabezamiento en texto común y corriente, pero los ficheros de plantillas se entregan pre compilados como ficheros–objeto, para que el usuario

[8] Todo puede ir en una línea, pero la costumbre dicta usar dos para esclarecer algo la inevitable ofuscación presente.

los enlace en sus trabajos sin requerir de su compilación *in situ*, porque las plantillas son elementos que poseen un alto valor añadido.

> La verdad es que las compañías norteamericanas Hewlett-Packard y Silicon Graphics Computer Systems, Inc. fueron muy altruistas en su momento al poner sus técnicas en el dominio público.

Reutilización del código

Hay tres maneras de conseguir la reutilización de código, algo muy deseado por todos, pues una vez desarrollado y depurado, además de aportar el conjunto de las doce cualidades ya mencionadas en la página 10, suma rapidez al desarrollo de las aplicaciones.

La reutilización del código se basa en:

1. La programación modular. Las funciones obran como módulos reusables: se crean una vez y se aplican muchas veces a disímiles situaciones. Por ejemplo, si se usa una función pre compilada de la biblioteca estándar (o de bibliotecas de terceros), se está aplicando este tipo de reusabilidad. Esto se utilizó ampliamente en el tomo anterior.
2. La programación genérica. Las plantillas son módulos "en blanco" que se codifican una vez, se depuran cuidadosamente y luego se aplican en diversas circunstancias, obteniendo muy buenos resultados. Es el tema a ver ahora, más encauzado específicamente a las funciones.
3. La OOP. Aplica las plantillas a las clases, aportándoles la cualidad de reutilización, pero eso es una historia para el tomo siguiente.

Unas situaciones

Véanse las situaciones que aparecieron en el desarrollo del primer tomo de la serie:

1. En muchos de los programas del primer tomo se tuvo que imprimir a consola la secuencia de valores de un arreglo, manejados en un instante dado para cada caso, con funciones tales como:

```
             1er. Parámetro             | 2do. Prm.  | 3er. Prm.
void mostrar( const int ary[],          | int n      |          );
void mostrar( const int tabla[][COLS],  | int n      |          );
void mostrar( int *pVect                |            |          );
void mostrar( int *pTabla               |            |          );
void mostrar( int pTabla[][COLS],       | int filas  |          );
void mostrar( int *pTab,                | int f,     | int c    );
void mostrar( const int *pAry,          | int n,     | int cols );
```

   Aunque internamente sean distintas, al inspeccionarlas se nota que cada implementación tiene un primer parámetro (constante o no) que es el tipo de dato con que opera y puede o no tener uno o más parámetros, pero lo importante es que todas hacen lo mismo: ponen en consola los distintos valores de un tipo de dato en forma visualmente agradable, con o sin información adicional.

   Con este tipo de codificación dispersa no es fácil ganar en robustez, pues hay que producir código específico para cada ocasión y dicha producción siempre está sujeta a errores. Tampoco se gana en fiabilidad, integridad o precisión porque cada vez hay que probar y depurar al módulo y comprobar la justeza de la respuesta. La tipicidad se pierde entre tantas producciones como se observa en los ejemplos, y la rapidez de entrega se ve ralentizada por las veces que hay que concebirlo todo desde cero.

   Y también hay que considerar que la creación de código mediante operaciones de *corte-y-empaste* tiene buenas probabilidades de deslizar algún que otro error, posibilidad que aumenta con el tamaño y dificultad de la tarea.

Desde luego, una mejor opción fue crear unas plantillas, codificadas en un archivo que se llamó **`print.h`**, adicionándoles las inclusiones adecuadas, flexibilizando la llamada que ahora podía ser de cualquier tipo nativo y no sólo de enteros y aceptar varios contenedores y parámetros. Una vez que fueron probadas y refinadas, se pudieron usar en el texto una y otra vez sin problemas, ayudando a minimizar su tamaño y a producir un código pitónico, con sus cualidades asociadas.

2. En varios ejemplos del primer tomo se requirieron valores aleatorios, tanto de enteros como de números reales y se utilizaron un par de funciones y varias técnicas para su reproducción, todas proporcionadas por la biblioteca estándar, pero insatisfactorias para aplicaciones altamente especializadas. La norma C++11 proporciona un nuevo método de mayor calidad y más efectivo para generar esta clase de números. No es que se tenga que utilizar normalmente, pero…

   Una solución para la creación de valores azarosos de alta calidad en los programas del texto fue codificar tres plantillas que emplearon esta técnica avanzada, codificadas en un único archivo que se llamó **`rnd.h`** Se usó un motor tipo Mersenne Twister que garantiza una alta aleatoriedad y tres distribuciones matemáticas: la Distribución Lineal Discreta, la Distribución Lineal Continua y la Distribución Normalizada de Gauss. Ver sus definiciones en Wikipedia, artículo principal: Distribución (estadística)

3. El trabajo con los arreglos numéricos lineales se hace fastidioso debido a ciertas irregularidades en su integración a la biblioteca estándar, que han quedado ahí. Además, los **`valarrays`** son seudocontenedores y no aceptan los iteradores de la STL, por la que su llenado automático de valores azarosos no se hace de la misma forma que los de los contenedores secuenciales. Siguiendo la línea recomendada por (Josuttis, p. Cap.12.2), se presentan seis plantillas especializadas para facilitar el trabajo con los arreglos numéricos en un único archivo que se llamó **`varray.h`**

Seguidamente se pasa a mostrar las plantillas que remedian las situaciones antes expuestas. Aunque el perplejo lector no entienda bien el código declarado, escríbalo, haga el ejercicio, vuelva a él de vez en cuando, y durante el estudio del texto el autor le asegura en verdad que, con un poco de voluntad y algo de esfuerzo, todo se esclarecerá feliz y paulatinamente.

Además, el tercer tomo de esta serie dedica un tema el asunto

---

Las plantillas básicas

Se crearon tres archivos de plantillas: uno para mostrar los datos de distintos contenedores de una forma visualmente agradable en consola*, teniendo presente los resultados alcanzados en el primer tomo* y se le llamó **`print.h`** El segundo se llamó **`rnd.h`**, concebido para generar valores azarosos de alta calidad y el tercero se llamó **`varray.h`**, para trabajar los arreglos numéricos lineales.

```
e:\Texto\Tomo 2\Programas\deposito>dir
 El volumen de la unidad E es SIMON
 El número de serie del volumen es: 82F8-4023

 Directorio de e:\Texto\Tomo 2\Programas\deposito

12/11/2020  11.17 AM    <DIR>          .
12/11/2020  11.17 AM    <DIR>          ..
12/11/2020  10.46 AM             2,707 print.h
12/11/2020  10.44 AM             1,723 rnd.h
12/11/2020  11.02 AM             3,417 varray.h
               3 archivos          7,847 bytes
               2 dirs  503,325,118,464 bytes libres
```

Ilustración 2. El depósito y su contenido

Doxygen: utilización mínima

El desarrollador debe adjuntar a cada plantilla suficiente información para que su usuario pueda emplearla sin grandes contratiempos. Una forma de hacerlo es valerse de una aplicación como Doxygen, integrado en Code::Blocks. Si el lector no está usando este IDE, deberá consultar su manual. Si acepta Doxygen, ¡felicidades! Y si no, pues no le queda de otras: a explorar por cuenta propia, opción muy saludable para afianzar los nuevos saberes.

El lector debe estudiar los comentarios "no estandarizados" que se muestran en el código de las plantillas. Son de Doxygen, un generador de documentación para código fuente de ISO C++, mayormente escrito por Dimitri van Heesch. Para los propósitos del curso está bien, dado que es fácil de usar, está en Internet (http://www.doxygen.org/), es integrado a Code::Blocks y es gratis.

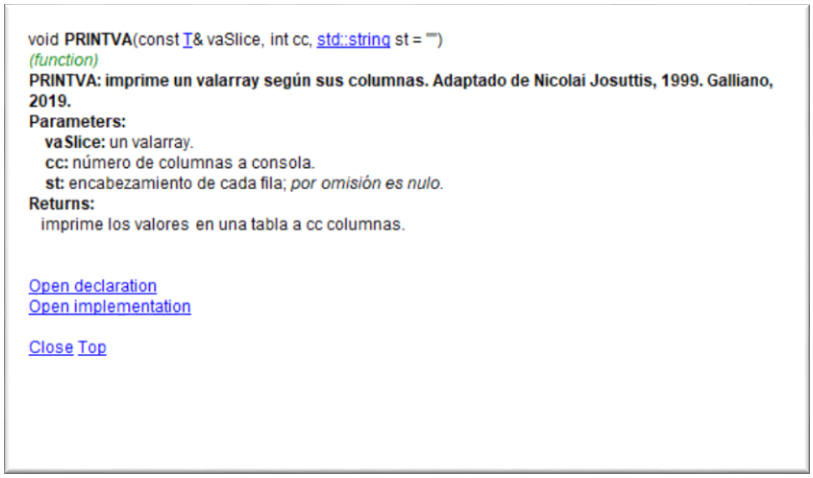


Ilustración 3. Doxygen y Code::Blocks

Un comentario de Doxygen:

- De varias líneas va encerrado entre los símbolos /** y */
- De una línea va a continuación del símbolo /// y se extiende hasta el final de la línea.
- No acepta elementos de edición en su código. Poner toda la información sin preocuparse de espacios de más, alineación de textos, cambios de línea, etc.
- Acepta letras en español. Ponerlas directamente y no como escapes.
- Acepta un conjunto de comandos de la forma *@comando*. Los tres básicos que se escriben en este orden según se necesiten son:
    1) *@brief*: el objetivo de la plantilla, que debe ser dado en detalle.
    2) *@param*: el nombre de cada parámetro y qué hace.
    3) *@return*: el tipo y valor que retorna la función, si lo hay.
- Acepta un conjunto de modificadores que alteran la salida del texto. Los dos principales son:
    a) texto *pone el texto en itálicas*.
    b) b>texto **pone el texto en negritas**.

Al comenzar a escribir el nombre de la plantilla, el auto-completamiento de Code::Blocks, gracias a Doxygen pone a disposición del programador:

- La sintaxis completa de la llamada a la función.
- La descripción de la plantilla, dada en la sección *@brief*

- La definición de cada parámetro, como se ve en la sección *@param*
- El resultado devuelto, como muestra la sección *@return*
- Opciones para abrir una de dos, su declaración o su definición.

Los códigos

Son de tres categorías: de impresión, puestas en **print.h**, para generación de números seudoaleatorios de alta calidad, puestas en **rnd.h** y para trabajar los arreglos lineales numéricos, puestas en **varray.h**

- De impresión: para mostrar valores a consola y hechas con miras a solucionar problemas selectos de (Introducción al C++, tomo 1), **LISTA** muestra los valores de cualquier tipo estándar de contenedor, separados por comas y **MUESTRA** es otra forma de verlo, pero separados por tabulaciones. **TABULA** se adaptó para sacar una tabla a consola. En todos los casos se usa el **for** basado en rangos para sus ciclos.

```
#ifndef PRINT_H_INCLUDED
#define PRINT_H_INCLUDED

#include <iostream>
#include <iomanip>

/**
@brief LISTA: imprime opcionalmente una cadena C++ seguida
por todos los elementos del contenedor K, separados por
comas, removiendo la coma final. Galliano, 2019.
@param <b>K:</b> contenedor del tipo T, que itere hacia adelante.
@param <b>st:</b> encabezamiento; <em>por omisión es nulo</em>.
@param <b>pr:</b> precisión; <em>por omisión es cero</em>.
*/
template <typename T> inline
void LISTA( const T & K, std::string st = "", short pr = 0 ) {
    std::ios::fmtflags oldFlags = std::cout.flags();
    std::cout << st << std::fixed << std::setprecision( pr );

    for ( auto x : K )
        std::cout << x << ", ";

    std::cout << "\b\b ";
    std::cout.flags( oldFlags );
}

/**
@brief MUESTRA: imprime opcionalmente una cadena C++ seguida
por todos los elementos del contenedor K, puestos en una línea,
separándolos por tabulaciones. Galliano, 2019,
@param <b>K:</b> contenedor del tipo T, que itere hacia adelante.
@param <b>st:</b> encabezamiento; <em>por omisión es nulo</em>.
*/
template <typename T> inline
void MUESTRA( const T & K, std::string st = "" ) {
    std::ios::fmtflags oldFlags = std::cout.flags();
    std::cout << st;

    for ( auto x : K )
        std::cout << '\t' << x;

    std::cout << '\t';
    std::cout.flags( oldFlags );
}
```

```
/**
@brief TABULA: imprime opcionalmente una cadena st seguida por todos
los elementos del contenedor K, en una tabla de cc columnas, con pr
decimales de precisión y mostrando o no el signo. Galliano, 2019.
@param <b>K:</b> contenedor del tipo T, que itere hacia adelante.
@param <b>st:</b> encabezamiento; <em>por omisión es nulo.</em>
@param <b>pr:</b> precisión; <em>por omisión es cero.</em>
@param <b>sg:</b> muestra el signo positivo; <em>por omisión no se ve.</em>.
@param <b>cc:</b> cantidad de columnas a consola; <em>por omisión son 10.</em>.
*/
template <typename T> inline
void TABULA( const T & K, std::string st = "", short pr = 0, bool sg = false, short cc = 10 ) {
    short col = 0;
    std::ios::fmtflags oldFlags = std::cout.flags();
    std::cout << st << std::fixed << std::setprecision( pr );
    sg ? std::cout << std::showpos : std::cout << std::noshowpos;

    for ( auto x : K ) {
        ++col;
        std::cout << std::setw( 6 ) << x;
        0 == col % cc ? std::cout << '\n' : std::cout << '\t' ;
    } // for

    std::cout << '\n';
    std::cout.flags( oldFlags );
}

#endif // PRINT_H_INCLUDED
```

- De generación: para generar números seudoaleatorios de alta calidad. **RELLENA_I** genera valores aleatorios enteros uniformes y discretos. **RELLENA_R** genera valores aleatorios reales en intervalos uniformes y continuos; ambas brindan mayor calidad que **rand()**. **GAUSS** maneja valores gaussianos normalizados, propios para trabajos estadísticos, algo que **rand()** no puede dar. Si no se entiende el código, quizás sea prudente ahora mismo ver en la página 107.

```
#ifndef RND_H_INCLUDED
#define RND_H_INCLUDED

#include <algorithm>
#include <functional>

/**
@brief RELLENA_I: llena un contenedor de tamaño dado con valores enteros
aleatorios uniformemente distribuidos en el rango [p:q]. Galliano, 2019
@param <b>C:</b> un contenedor con tamaño dado.
@param <b>p:</b> es el valor entero menor.
@param <b>q:</b> es el valor entero mayor.
*/
template <typename T> inline
void RELLENA_I( T & C, int p, int q ) {
    std::mt19937 mt( rand() );
    std::uniform_int_distribution<int>dist( p, q );
    auto g = bind( dist, mt );
    generate( C.begin(), C.end(), g );
}

/**
@brief RELLENA_R: llena un contenedor de tamaño dado con valores reales
aleatorios uniformemente distribuidos en el rango [p:q). Galliano, 2019
@param <b>C:</b> un contenedor con tamaño dado.
@param <b>p:</b> es el valor menor: float o double.
```

```
@param <b>q:</b> es el valor mayor: float o double.
*/
template <typename T> inline
void RELLENA_R(T &C, int p, int q) {
    std::mt19937 mt( rand() );
    std::uniform_real_distribution<double>dist( p, q );
    auto g = bind( dist, mt );
    generate( C.begin(), C.end(), g );
}

/**
@brief GAUSS: llena un contenedor de tamaño dado con valores
reales aleatorios gaussianos normalizados Z~(0, 1) Galliano, 2019
@param <b>C:</b> un contenedor con tamaño dado.
*/
template <typename T> inline
void GAUSS( T & C ) {
    std::mt19937 mt( rand() );
    std::normal_distribution<double>dist( 0, 1 );
    auto g = bind( dist, mt );
    generate( C.begin(), C.end(), g );
}

#endif // RND_H_INCLUDED
```

- Para valores numéricos lineales, tópico a ver en la página 57: **VA** convierte un subconjunto de un arreglo numérico de valores en un **valarray** completo; y **PRINTVA** está sobrecargada en dos versiones: en una imprime un **valarray** según los valores de su tajada y en la otra imprime un **valarray** en la cantidad de columnas estipuladas por el parámetro pasado. Se repiten las funciones de relleno de valores azarosos adaptadas al medio.

```
#ifndef VARRAY_H_INCLUDED
#define VARRAY_H_INCLUDED

#include <iostream>
#include <iomanip>
#include <valarray>

/**
@brief VA: convierte un slice (o subconjunto) en un valarray.
Adaptado de Nicolai Josuttis, 1999. Galliano, 2019.
@param <b>vaSlice:</b> un subconjunto o <em>slice</em>.
@return un valarray
*/
template <typename T> inline
std::valarray<typename T::value_type> VA(const T &vaSlice) {
    return std::valarray<typename T::value_type>(vaSlice);
}

/**
@brief PRINTVA: imprime el slice de un valarray.
Adaptado de Nicolai Josuttis, 1999. Galliano, 2019
@param <b>vaSlice:</b> un subconjunto o <em>slice</em>.
@return imprime los valores.
*/
template<typename T> inline
void PRINTVA(const T &vaSlice) {
    std::ios::fmtflags oldFlags = std::cout.flags();
    std::cout << std::fixed;

    for ( unsigned i = 0; i < vaSlice.size(); i++ )
        std::cout << std::setw(8) << vaSlice[i] << '\n';
```

```
    std::cout.flags(oldFlags);
}

/**
@brief PRINTVA: imprime un valarray según las columnas pasadas.
Adaptado de Nicolai Josuttis, 1999. Galliano, 2019.
@param <b>vaSlice:</b> un valarray.
@param <b>cc:</b> número de columnas a consola.
@param <b>st:</b> encabezamiento de cada fila; <em>por omisión es nulo</em>.
@return imprime los valores en una tabla a cc columnas.
*/
template<class T> inline
void PRINTVA(const T &vaSlice, int cc, std::string st = "") {
    std::ios::fmtflags oldFlags = std::cout.flags();
    std::cout << std::fixed;

    for ( unsigned i = 0; i < vaSlice.size() / cc; ++i ) {
        if ( st.size() > 0 )
            std::cout << st << i + 1;

        for ( int j = 0; j < cc; ++j )
            std::cout << std::setw(9) << vaSlice[i * cc + j];

        if ( vaSlice.size() / cc > 1 )
            std::cout << '\n';
    } // for unsigned i

    std::cout.flags(oldFlags);
}

/**
@brief RELLENA_VI: llena un valarray con enteros aleatorios.
Galliano, 2019.
@param <b>va:</b> un valarray.
@param <b>s:</b> tamaño
@param <b>p:</b> límite menor
@param <b>q:</b> límite superior
@return un valarray con valores al azar
*/
template<typename T> inline
void RELLENA_VI( T &va, int s, int p, int q ) {
    std::mt19937 mt( rand() );
    std::uniform_int_distribution<int>dist( p, q );

    for ( int i = 0; i < s; ++i )
        va[i] = dist( mt );
}

/**
@brief RELLENA_VD: llena un valarray con dobles aleatorios.
Galliano, 2019.
@param <b>va:</b> un valarray.
@param <b>s:</b> tamaño
@param <b>p:</b> límite menor
@param <b>q:</b> límite superior
@return un valarray con valores al azar
*/
template<typename T> inline
void RELLENA_VD( T &va, int s, int p, int q ) {
    std::mt19937 mt( rand() );
    std::uniform_real_distribution<double>dist( p, q );
```

```
        for ( int i = 0; i < s; ++i )
            va[i] = dist( mt );
    }

    /**
    @brief GAUSS_VD: llena un valarray con aleatorios normalizados.
    Galliano, 2019.
    @param <b>va:</b> un valarray.
    @param <b>s:</b> tamaño
    @return un valarray con valores gaussianos al azar
    */
    template<typename T> inline
    void GAUSS_VD( T &va, int s ) {
        std::mt19937 mt( rand() );
        std::normal_distribution<double>dist( 0, 1 );

        for ( int i = 0; i < s; ++i )
            va[i] = dist( mt );
    }

    #endif // VARRAY_H_INCLUDED
```

Programa 1: uso de plantillas

Se pide codificar un manejador que muestre la plantilla **print.h** en acción.

El programa

El manejador demuestra la versatilidad de las plantillas: trabajan con varios tipos de datos y de contendores STL. Observar la inclusión del fichero de plantillas: cada sistema reflejará dónde se puso, pero será más o menos así:

```
#include "../deposito/print.h"
#include "../deposito/rnd.h"
#include "../deposito/varray.h"
#include <vector>
#include <list>
#include <deque>

typedef std::vector<int> Ary;
using namespace std;

int main() {
    srand( time( nullptr ) );
    cout << "Programa 1: trabajando con plantillas.\n";
    vector<int> vInt( 6 );
    RELLENA_I( vInt, 10, 40 );
    LISTA( vInt, "\nVector de 6 valores entre 10 y 40\n[" );
    cout << "\b]\n";

    // listas: 6 valores enteros entre 20 y 30
    list<int>aLst( 6 );
    RELLENA_I( aLst, 10, 40 );
    LISTA( aLst, "\nLista de 6 valores enteros entre 10 y 40:\n<" );
    cout << "\b>\n";
    MUESTRA( aLst, "\nMuestra los mismos valores que el vector:\n" );
    cout << '\n';
```

```
    //  colas: 6 valores reales entre 10 y 40
    deque<double>aDek( 6 );
    RELLENA_R( aDek, 10, 40 );
    LISTA( aDek, "\nCola de 6 valores reales entre 10 y 40, con 3 dec.:\n<", 3 );
    cout << "\b>\n";

    // vectores: 30 valores gaussianos
    vector<double>vct( 36 );
    GAUSS( vct );
    TABULA( vct, "\nTabla a 6 cols de 36 valores normalizados, con 2 dec.:\n", 2, true, 6 );

    //  valarrays: 6 valores reales entre 20 y 30
    valarray<double>va( 6 );
    RELLENA_VD( va, 6, 20, 30 );
    cout << "Un valarray en memoria, de 6 valores reales.\n";
    cout << "Impreso como un vector-fila, 5 dec.:\n";
    cout << std::setprecision( 5 );
    PRINTVA( va, 6 );
    cout << "\n\nAhora impreso como un vector-columna, 4 dec.:\n";
    cout << std::setprecision( 4 );
    PRINTVA( va, 1 );
    cout << "\nAhora impreso como tabla de 3x2, 3 dec.:\n";
    cout << std::setprecision( 3 );
    PRINTVA( va, 2 );
    cout << "\nAhora impreso como tabla de 2x3, 2 dec.:\n";
    cout << std::setprecision( 2 );
    PRINTVA( va, 3 );
    cout << "\nAhora mostrando el slice(1,2,3), 1 dec.:\n";
    cout << std::setprecision( 1 );
    PRINTVA( VA( va[slice( 1, 2, 3 )] ) );
    cout << '\n';

    system( "pause" );
    return EXIT_SUCCESS;
}
```

La salida

```
Programa 1: trabajando con plantillas.

Vector de 6 valores entre 10 y 40
[39, 29, 25, 27, 20, 12]

Lista de 6 valores enteros entre 10 y 40:
<34, 19, 34, 21, 38, 24>

Muestra los mismos valores que el vector:
        34      19      34      21      38      24

Cola de 6 valores reales entre 10 y 40, con 3 dec.:
<34.833, 32.970, 21.831, 19.131, 14.230, 14.664>
```

```
Tabla a 6 cols de 36 valores normalizados, con 2 dec.:
 -1.85   -0.21   -1.33   +0.89   +0.58   +0.01
 +1.34   +1.18   +2.03   -0.60   -0.32   -0.14
 +0.26   -1.03   +0.26   +0.15   +1.53   -0.96
 -0.97   -1.27   +0.42   -0.19   -1.96   -1.37
 +1.16   +0.96   +0.25   -1.74   +0.60   -0.35
 -0.79   +0.72   -0.97   -0.52   -1.06   -1.60

Un valarray en memoria, de 6 valores reales.
Impreso como un vector-fila, 5 dec.:
 23.96184 29.26811 29.88321 22.06512 22.61867 21.37030

Ahora impreso como un vector-columna, 4 dec.:
  23.9618
  29.2681
  29.8832
  22.0651
  22.6187
  21.3703

Ahora impreso como tabla de 3x2, 3 dec.:
   23.962   29.268
   29.883   22.065
   22.619   21.370

Ahora impreso como tabla de 2x3, 2 dec.:
    23.96    29.27    29.88
    22.07    22.62    21.37

Ahora mostrando el slice(1,2,3), 1 dec.:
    29.3
    22.6
```

Notas:

- Las plantillas pueden sobrecargarse y se les aplican las mismas reglas que a las funciones.
- Los cálculos con valores de punto flotante se hacen con todos sus decimales, aunque se muestren en consola según la precisión pedida.

4 - APLICANDO LA STL

> *La biblioteca estándar de C++ utiliza plantillas para proveer una colección de contenedores y algoritmos (sobre 70) para procesar sus elementos. Esta parte es comúnmente conocida como la biblioteca estándar de plantillas o por sus siglas en inglés, STL. Como su nombre entraña, la STL contiene sus funciones genéricas y clases construidas con plantillas. (Halterman, pág. 633)*

¡Por fin! Ha llegado el momento de olvidar la programación manual de arreglos dinámicos, estructuras complicadas, listas encadenadas y ciclos dificultosos; olvidar la programación para cada caso que precise de diferentes algoritmos de búsqueda o de ordenamiento; cosas todas que el programador al menos debe conocer, y que fueron estudiadas en el primer tomo de esta serie. Para usar la STL simplemente se define el contenedor correcto y se llaman los métodos y

algoritmos adecuados para procesar los datos, lo cual le otorga al ISO C++ un nivel de abstracción muchísimo más alto que el típico de ANSI C o de la programación en súper C.

> A más abstracción, mayor facilidad de uso y mejor administración de las dificultades inherentes al problema

A continuación, serán aplicados y explicados a medida que se usen, algunos de los elementos de la STL a la solución de problemas propuestos y ejercicios presentados, mayoritariamente escogidos del primer tomo de esta serie, como mecanismo comparativo entre el desarrollo totalmente "manual" por decirlo así, y el uso de elementos modulares, que hace tan interesante al C++.

¿Cómo se hizo? Simplemente, al rehacer un programa o ejercicio, se analizó cuál era el contenedor más apropiado y se examinó en la bibliografía destacada la forma de aplicarlo, o sea, el algoritmo y el repetidor apropiados al tema. Esto se hizo buscando mostrar por cada ejemplo los repetidores más usados. Si el lector no concuerda con la solución dada o duda de su eficiencia, por favor trate de encontrar la suya propia.

Límites de su sistema

En (Introducción al C++, tomo 1, pág. 47) se codificaron los elementos técnicos de los valores que cada sistema maneja en particular, teniendo que incluir bibliotecas especializadas (**`climits`** y **`cfloat`**) y buscar en **`<limits.h>`** las definiciones de constantes del sistema tales como **`CHAR_BIT`**, **`LONG_MIN`** o **`FLT_RADIX`**. Usando la biblioteca **`<limits>`** y sus métodos, se ahorran tiempo y esfuerzos. Sus principales miembros están dados en la tabla que sigue:

Tabla 8. (simplificada) Miembros de la clase limits

Miembro	Significado
`is_specialized`	El tipo está especializado para límites numéricos
`is_signed`	El tipo tiene signo
`is_integer`	El tipo es un número entero
`is_exact`	Los cálculos no producen errores de redondeo (**`true`** para los enteros)
`is_modulo`	Si se exceden los límites, se puede obtener un resultado falso al sumarlos
`is_iec559`	Es conforme con las normas IEC 559 y IEEE 754[9]
`min()`	Valor finito mínimo (no tiene significado si **`is_signed`** es **`true`**)
`max()`	Valor finito máximo
`digits`	Cantidad de bits sin signo para caracteres y enteros
	Cantidad de dígitos en la mantisa de un número en punto flotante
`digits10`	Cantidad de dígitos decimales
`radix`	Enteros: base de su representación (casi siempre es 2) De punto flotante: base de la representación del exponente
`min_exponent`	Exponente mínimo, entero, negativo en base radix
`max_exponent`	Exponente máximo, entero, positivo en base radix
`min_exponent10`	Exponente mínimo, entero, negativo en base 10
`max_exponent10`	Exponente máximo, entero, positivo en base 10
`epsilon()`	Diferencia entre 1 y el menor valor que sea mayor que 1
`round_style`	Estilo de redondeo de los números en punto flotante
`round_error()`	Máximo error de redondeo según la norma ISO/IEC 10967-1
`has_infinity`	El tipo posee representación para la infinitud positiva
`infinity()`	La representación de la infinitud positiva, si es que existe

[9] La Comisión Electrotécnica Internacional (IEC), es una organización de estandarización en los campos de lo eléctrico, lo electrónico y las tecnologías relacionadas. El Instituto de Ingeniería Eléctrica y Electrónica (IEEE), es una asociación mundial de ingenieros dedicada a la estandarización y el desarrollo en áreas técnicas. (Wikipedia en español)

Nota: el redondeo es: indeterminado (**-1**); hacia cero (**0**); al decimal más próximo (**1**); hacia infinito (**2**); y hacia infinito negativo (**3**)

Programa 2: límites con enteros y caracteres

```
// Programa 3-Límites enteros
#include <iostream>
#include <cstdlib>
#include <limits>

typedef std::numeric_limits<char>              lChar;
typedef std::numeric_limits<short>             lShort;
typedef std::numeric_limits<unsigned short>    lUshort;
typedef std::numeric_limits<int>               lInt;
typedef std::numeric_limits<unsigned int>      lUint;
typedef std::numeric_limits<long int>          lLintl;
typedef std::numeric_limits<unsigned long int> lUlint;

using std::numeric_limits;
using std::cout;

int main() {
    cout <<
    "Programa 2: caracter\241sticas de este sistema (1)\n\n"
    "Caracteres:\n"
    " bits por byte = " << lChar::digits << '\n';
    lChar::is_signed
        ?cout << " El char presenta signo.\n"
        :cout << " El char no presenta signo.\n";
    lChar::is_specialized
        ?cout << " La cadena C++ no tiene l\241mites.\n"
        :cout << " La cadena C++ tiene l\241mites.\n";
    lChar::is_iec559
        ?cout << " Cumple con las normas IEC 559\n\n"
        :cout << " No cumple con las normas IEC 559\n\n";
    cout <<
    "N\243meros:\n short:"
    "\n  valor m\241nimo = " << lShort::min() <<
    "\n  valor m\240ximo = " << lShort::max() << '\n' <<
    "\n unsigned short:"
    "\n  valor m\241nimo = " << lUshort::min() <<
    "\n  valor m\240ximo = " << lUshort::max() << '\n' <<
    "\n int:"
    "\n  valor m\241nimo = " << lInt::min() <<
    "\n  valor m\240ximo = " << lInt::max() << '\n' <<
    "\n unsigned int:"
    "\n  valor m\241nimo = " << lUint::min() <<
    "\n  valor m\240ximo = " << lUint::max() << '\n' <<
    "\n long int:"
    "\n  valor m\241nimo = " << lLintl::min() <<
    "\n  valor m\240ximo = " << lLintl::max() << '\n' <<
    "\n unsigned long int:"
    "\n  valor m\241nimo = " << lUlint::min() <<
    "\n  valor m\240ximo = " << lUlint::max() << "\n\n";
    system("pause");
    return EXIT_SUCCESS;
}
```

La salida

```
Programa 2: características de este sistema (1)

Caracteres:
 bits por byte = 7
 El char presenta signo.
 La cadena C++ no tiene límites.
 No cumple con las normas IEC 559

Números:
 short:
  valor mínimo = -32768
  valor máximo = 32767

 unsigned short:
  valor mínimo = 0
  valor máximo = 65535

 int:
  valor mínimo = -2147483648
  valor máximo = 2147483647

 unsigned int:
  valor mínimo = 0
  valor máximo = 4294967295

 long int:
  valor mínimo = -2147483648
  valor máximo = 2147483647

 unsigned long int:
  valor mínimo = 0
  valor máximo = 4294967295
```

Programa 3: límites con punto flotante

```
// Programa 3: límites de los números reales.
// Fecha: 21/8/2019; 8:22 am
// S. Galliano.

#include <iostream>
#include <cstdlib>
#include <limits>

typedef std::numeric_limits<float>       lFlt;
typedef std::numeric_limits<double>      lDbl;
typedef std::numeric_limits<long double> lLdbl;

using std::numeric_limits;

int main() {
    std::cout
```

```
    << "Programa 3: caracter\241sticas de este sistema (2)\n\n"
    << "Base radix = " << lFlt::radix
    << "\nEstilo de redondeo: "; // el redondeo es único

    switch (lLdbl::round_style) {
    case -1:
        std::cout << "indeterminado.";
        break;
    case 0:
        std::cout << "hacia cero.";
        break;
    case 1:
        std::cout << "al decimal m\240s pr\242ximo.";
        break;
    case 2:
        std::cout << "hacia infinito.";
        break;
    case 3:
        std::cout << "hacia infinito negativo.";
        break;
    } // switch

    std::cout
    << "\nError m\240ximo de redondeo: " << lFlt::round_error() << "\n"
    << "\nL\241mites de los valores en punto flotante:\n"
    << "\n float:"
    << "\n  precisi\242n decimal = " << lFlt::digits10
    << "\n  l\241mite de certeza = " << lFlt::epsilon()
    << "\n  valor m\240ximo = "      << lFlt::max()
    << "\n  valor m\241nimo = "      << lFlt::min()
    << "\n  exponente decimal m\240ximo = " << lFlt::max_exponent10
    << "\n  exponente decimal m\241nimo = " << lFlt::min_exponent10
    << "\n  exponente radix m\240ximo = "   << lFlt::max_exponent
    << "\n  exponente radix m\241nimo = "   << lFlt::min_exponent
    << "\n\n double:"
    << "\n  precisi\242n decimal = " << lDbl::digits10
    << "\n  l\241mite de certeza = " << lDbl::epsilon()
    << "\n  valor m\240ximo = "      << lDbl::max()
    << "\n  valor m\241nimo = "      << lDbl::min()
    << "\n  exponente decimal m\240ximo = " << lDbl::max_exponent10
    << "\n  exponente decimal m\241nimo = " << lDbl::min_exponent10
    << "\n  exponente radix m\240ximo = "   << lDbl::max_exponent
    << "\n  exponente radix m\241nimo = "   << lDbl::min_exponent
    << "\n\n long double:"
    << "\n  precisi\242n decimal = " << lLdbl::digits10
    << "\n  l\241mite de certeza = " << lLdbl::epsilon()
    << "\n  valor m\240ximo = "      << lLdbl::max()
    << "\n  valor m\241nimo = "      << lLdbl::min()
    << "\n  exponente decimal m\240ximo = " << lLdbl::max_exponent10
    << "\n  exponente decimal m\241nimo = " << lLdbl::min_exponent10
    << "\n  exponente radix m\240ximo = "   << lLdbl::max_exponent
    << "\n  exponente radix m\241nimo = "   << lLdbl::min_exponent
    << "\n\n";

    //  fin
    system("pause");
    return EXIT_SUCCESS;
}
```

La salida

```
Programa 3: características de este sistema (2)

Base radix = 2
Estilo de redondeo: al decimal más próximo.
Error máximo de redondeo: 0.5

Límites de los valores en punto flotante:

 float:
  precisión decimal = 6
  límite de certeza = 1.19209e-007
  valor máximo = 3.40282e+038
  valor mínimo = 1.17549e-038
  exponente decimal máximo = 38
  exponente decimal mínimo = -37
  exponente radix máximo = 128
  exponente radix mínimo = -125

 double:
  precisión decimal = 15
  límite de certeza = 2.22045e-016
  valor máximo = 1.79769e+308
  valor mínimo = 2.22507e-308
  exponente decimal máximo = 308
  exponente decimal mínimo = -307
  exponente radix máximo = 1024
  exponente radix mínimo = -1021

 long double:
  precisión decimal = 18
  límite de certeza = 1.0842e-019
  valor máximo = 1.18973e+4932
  valor mínimo = 3.3621e-4932
  exponente decimal máximo = 4932
  exponente decimal mínimo = -4931
  exponente radix máximo = 16384
  exponente radix mínimo = -16381
```

Si se puede, comparar respectivamente la codificación y las salidas con las de los programas 5 y 6 del primer tomo.

Cadenas de C++

C++ ofrece una forma más segura de usar y más simple de manejar las cadenas. En el fichero de cabecera **`string`** hay una clase dedicada a tratar cadenas, que oculta la responsabilidad de su administración, oferta muchos métodos que hacen el trabajo más intuitivo, y permite su tratamiento casi como si fueran variables ordinarias, cubriendo un amplio espectro de

posibilidades en sus aplicaciones, aunque desafortunadamente, no todas. (Introducción al C++, tomo 1, pág. 41) Para completitud se reproduce la tabla simplificada de la interfaz de las cadenas C++, dada en el tomo1 de la serie[10].

Tabla 9. (simplificada) Interfaz de la clase string

Constructores	Resultados
`string c`	Crea una cadena **c** vacía
`[const] string c(texto)` `[const] string c = texto`	Crea una cadena **c** conteniendo un texto que puede ser o no constante.
Métodos	**Resultados**
`c.assign(texto)`	**c** fue creada y ahora se la asigna un texto
`c.at(i)` `c[i]`	Accede al carácter iésimo de la cadena con y sin comprobación de límites, respectivamente. La primera forma es menos eficiente en tiempo.
`c += texto` `c.append(texto)`	Añade **texto** a **c**
`c = c1 + c2`	Concatena (une) en **c** a **c1** con **c2**
`c.clear()`	Vacía la cadena **c**
`==, !=, not_eq, <, <=, >, >=`	Compara lexicográficamente dos cadenas
`n = c.size()`	Devuelve en **n** el tamaño de la cadena **c**
`b = c.empty()`	Devuelve **true** si **c** está vacía
`getline(fl, cd[, d])`	Lee la cadena **cd** al flujo de entrada **fl**; si está presente sólo lee hasta el delimitador **d**
`f_in >> v`	El flujo de entrada **f_in** lee un valor a la variable **v**
`f_out << cd`	Escribe la cadena **cd** al flujo de salida **f_out**
`c = c1.c_str()`	Devuelve el valor de **c1** como cadena constante ANSI C
`c.substr(c1)`	asigna a **c** la subcadena **c1** desde su inicio
`i = c.find(c1);`	Devuelve en **i** la posición de **c1** si está en **c**. Si no está devuelve el valor **str::npos**.

Notas:

- texto significa cualquier expresión textual legal y si es un literal, va entre comillas.
- Siempre usar **`string::size_type`** para el valor de retorno; no usar otro tipo puesto que quizás la comparación contra **`str::npos`** no funcione.
- El uso completo de **`getline()`** da la oportunidad de tratar ficheros textuales delimitados, en especial los CSV, eliminando la necesidad de usar la función **`strok()`**, propia de C.
- Si la posición de inicio está fuera de límites, al usar **`c.at(i)`** se lanza una excepción:
  `terminate called after throwing an instance of 'std::out_of_range' what(): basic_string::substr`
- La comparación lexicográfica o de diccionario entre dos textos es como sigue:
  - Se hace carácter contra carácter
  - Los números vienen primero y entran en su orden natural.
  - A viene antes que B, AA antes que AB, etc.
  - A viene antes que a, etc.

Hay un algoritmo que hace esa función, pero se usa cuando no es una cadena C++:

```
lexicographical_compare( K1.begin, K1.end(), K2.begin(), K2.end() )
```

donde K es un contenedor apropiado de la STL. La cadena C++ puede ser considerada como un contenedor de caracteres y por lo tanto, provee una interfaz para actuar con la STL mediante los métodos **`begin()`** y **`end()`** que devuelven repetidores para iterar sobre sus componentes y se le puede aplicar el algoritmo, pero si ella ya lo hace normalmente, ¿a qué complicar

[10] Las cadenas ANSI C se tratan en este texto cuando es absolutamente necesario.

más las cosas? También permite el método **`push_back()`** para aplicar insertores por la cola, aunque usualmente las cadenas se trabajan como un solo objeto al pasarlas, copiarlas o asignarlas a otras cadenas.

- ✓ Un insertor es un repetidor adaptado para permitir que un algoritmo trabaje en modo de inserción en vez de sobrescritura, que es como usualmente actúan.

Normalmente las cadenas C++ se manejan por sus métodos, pero cuando es de interés el procesamiento de sus componentes uno a uno, la aplicación de algoritmos de la STL es de mucha ayuda, tratando la cadena como un seudocontenedor.

Programa 4: cambio de tamaño

El problema 21 del cuarto bloque de ejercicios de (Introducción al C++, tomo 1, pág. 124) decía: el programa 17 pedía darle al usuario la opción de escoger entre poner su cadena en mayúsculas, minúsculas o tipo oración, pero no trabajaba bien con el español. Rehacerlo para que acepte frases en español y realice los cambios correctamente. Se estipula que la oración debe ser en minúsculas con la primera letra en mayúsculas, letra que además puede ser ASCII o española, y puede estar precedida de una apertura de interrogación o exclamación.

La solución hallada aquí involucra una cadena C++, pero las operaciones de procesamiento de palabras (*text processing* en inglés) no son cubiertas por la clase, así que no hay verdadera ganancia de sintetizar el código al cambiar las operaciones sobre cadenas ANSI C, sólo hay más seguridad. Para una explicación del algoritmo **`transform()`**, ver en la página 46.

Declaraciones

- Menú:

```
#ifndef MENU_H_INCLUDED
#define MENU_H_INCLUDED

#include <string>
//  enumera lo que pide el menú
enum { Terminar, Mayusculas, Minusculas, Oracion };

//  pone un menú en consola
//  input: nada
//  output: menú en consola
short menu();

#endif // MENU_H_INCLUDED
```

- Conversor:

```
#ifndef CONVERSOR_H_INCLUDED
#define CONVERSOR_H_INCLUDED

//  conversor.h: definiciones para el programa
//  cambia el tamaño de una letra
//  input: un caracter
//  output: el tamaño cambiado

char aMayusculas( char ch );
char aMinusculas( char ch );

#endif // CONVERSOR_H_INCLUDED
```

Definiciones

- Menú:

```
#include "menu.h"
#include <iostream>
#include <cstdlib>
#include <cctype>
using std::string;

//  un menú en consola
short menu () {
    string conjunto = "TMNO"; // opciones legales
    string opcion;
    string::size_type idx;

    while ( true ) {
        system( "cls" );
        std::cout <<
        "Problema 4: cambio de tama\244o\n\n" <<
        "(M)ay\243sculas, mi(N)\243sculas, (O)raci\242n, (T)erminar.\n"
        "Entre una opci\242n: ";
        std::getline( std::cin, opcion );
        idx = conjunto.find( toupper( opcion[0] ) );

        if ( string::npos == idx ) {
            std::cerr << "\nError: opci\242n ilegal.\n\n";
            system( "pause" );
        } else {
            return idx;
        } // if-else
    } // while
}
```

- Conversor: Las funciones manejadoras de caracteres **isalnum**, **isupper** e **islower** administran la conversión de un carácter, estrechando el universo de respuestas a las que se desean, que son los códigos de las letras propias del español. **toupper** y **tolower** hacen la conversión del carácter que cae fuera.

```
#include "conversor.h"
#include <cctype>
#include <iostream>

//  convierte una letra del español a mayúsculas
char aMayusculas( char ch ) {
    if ( isalnum(ch) and islower(ch) ) { // sólo minúsculas ASCII
        ch = toupper( ch );
    }  else {
        if ( ch == '\201' ) ch = '\232'; else //  ü -> Ü
        if ( ch == '\202' ) ch = '\220'; else //  é -> É
        if ( ch == '\240' ) ch = '\265'; else //  á -> Á
        if ( ch == '\241' ) ch = '\326'; else //  í -> Í
        if ( ch == '\242' ) ch = '\340'; else //  ó -> Ó
        if ( ch == '\243' ) ch = '\351'; else //  ú -> Ú
        if ( ch == '\244' ) ch = '\245';      //  ñ -> Ñ
    } // if
    return ch;
}

//  convierte una letra del español a minúsculas
char aMinusculas ( char ch ) { // sólo mayúsculas ASCII
    if ( isalnum(ch) and isupper(ch) ) {
```

```
            ch = tolower( ch );
        } else {
            if ( ch == '\220' ) ch = '\202'; else //  É -> é
            if ( ch == '\232' ) ch = '\201'; else //  Ü -> ü
            if ( ch == '\245' ) ch = '\244'; else //  Ñ -> ñ
            if ( ch == '\265' ) ch = '\240'; else //  Á -> á
            if ( ch == '\326' ) ch = '\241'; else //  Í -> í
            if ( ch == '\340' ) ch = '\242'; else //  Ó -> ó
            if ( ch == '\351' ) ch = '\243';      //  Ú -> ú
        } // if
        return ch;
    }
```

Programa

La operación de convertir el tamaño de un carácter se realiza tratando la cadena C++ como un contenedor de caracteres y transformando cada uno de ellos según convenga. Usa el algoritmo **transform** que será estudiado en la en la página 48.

```
#include "menu.h"
#include "conversor.h"
#include <iostream>
#include <cstdlib>
#include <algorithm>

typedef std::string Str;

int main() {
    Str frase; // la frase a convertir
    short idx; // índice de la opción

    while( true ) {
        idx = menu();
        if( Terminar == idx ) return EXIT_SUCCESS;

        //  conversión
        std::cout << "\n   Entre una frase: ";
        std::getline( std::cin, frase );

        switch( idx ) {
        case Mayusculas:
            //          desde,          hasta,        destino,        operación
            transform( frase.begin(), frase.end(), frase.begin(), aMayusculas );
            std::cout << "Frase a May\243sculas: " << frase << "\n\n";
            break;
        case Minusculas:
            transform( frase.begin(), frase.end(), frase.begin(), aMinusculas );
            std::cout << "Frase a Min\243sculas: " << frase << "\n\n";
            break;
        case Oracion:
            //  primero todo a minúsculas
            transform( frase.begin(), frase.end(), frase.begin(), aMinusculas );
            //  luego a a oración
            '\250' == frase[0] or '\255' == frase[0]  // si son los símbolos ¿ o ¡
                ? frase[1] = aMayusculas( frase[1] )  // cambiar segunda letra
                : frase[0] = aMayusculas( frase[0] ); // otrosí, cambiar primera
            std::cout << "   Frase a Oraci\242n: " << frase << "\n\n";
            break;
        } // switch
        system( "pause" );
    } // while
}
```

Salida

```
Problema 4: cambio de tamaño

(M)ayúsculas, mi(N)úsculas, (O)ración, (T)erminar.
Entre una opción: m

   Entre una frase: !una puñetera tarde de mayo!
Frase a Mayúsculas: !UNA PUÑETERA TARDE DE MAYO!
```

Funciones matemáticas

Aunque originalmente C++ usaba el mismo conjunto de funciones matemáticas que C, en la medida en que se refinaba, añadió a su biblioteca cmath versiones sobrecargadas y como resultado…ya no son iguales, aunque soportan el conjunto original. (Schildt, p. Caps. 8 y 9). Al usar una función matemática el programador siempre debería indagar si ya existe dentro de C++, para no tener que reinventar la rueda.

Seguidamente se reproduce la tabla simplificada de las funciones matemáticas de ISO C++.

Tabla 10. (simplificada) Funciones de la clase cmath

Función	Resultado
`n = abs(v)`	Devuelve el valor absoluto de **v**
`n = exp(x)`	Devuelve e^x **(e ≈ 2.7183)**
`n = sqrt(x)`	Raíz cuadrada de **x**. Si (**x < 0**) hay error de dominio
`n = pow(x, y)`	Devuelve x^y. Si **(x = 0)** & **(y ≤ 0)** o si **(x < 0)** & (**y** no es entero) hay error de dominio Si el resultado es un desbordamiento (overflow) hay error de rango
`n = log(x)`	Logaritmo natural (base **e**) de **x**. Si **x** es negativo hay error de dominio; si **x** es cero hay error de rango
`n = log10(x)`	Logaritmo (base **10**) de **x**. Si **x** es negativo hay error de dominio; si **x** es cero hay error de rango
`n = ceil(x)`	El menor entero no menor que **x**
`n = floor(x)`	El mayor entero no mayor que **x**
`n = fmod(x, y)`	Devuelve el resto de **x/y**
`n = modf(x, y)`	Divide **x** en dos partes; la parte fraccional va a **n**, y carga en **y** la parte entera
`n = sin(x)`	Seno de **x** (**x** en radianes)
`n = cos(x)`	Coseno de **x** (**x** en radianes)
`n = tan(x)`	Tangente de **x** (**x** en radianes)
`n = asin(x)`	Arco seno de **x** (-1 ≤ x ≤ 1)
`n = acos(x)`	Arco coseno de **x** (-1 ≤ x ≤ 1)
`n = atan(x)`	Arco tangente de **x**
`n = atan2(x, y)`	Arco tangente de **x/y**
`n = sinh(x)`	Seno hiperbólico de **x**
`n = cosh(x)`	Coseno hiperbólico de **x**
`n = tanh(x)`	Tangente hiperbólica de **x**
Estas funciones numéricas están contenidas en la biblioteca <cstdlib>	
`tmp = div(x, y)`	**tmp.quot()** es el cociente; **tmp.rem()** es el resto de la división entera **x/y**
`srand(time(nullptr))`	Nueva secuencia para la generación de números aleatorios
`n = rand()`	Próximo número de la secuencia

Notas:

- Todos los ángulos son dados u obtenidos en radianes.
- La sobrecarga para los valores enteros y de punto flotante unificó varias funciones. Por ejemplo, **`abs(), fabs()`** y **`labs()`** ahora ISO C++ las maneja como plantilla **`abs()`**.
- Si ocurre un sumidero (**`underflow`**), se lanza la macro **`-HUGE_VAL`**. Si ocurre un desbordamiento (**`overflow`**), se lanza la macro **`HUGE_VAL`**. En ambos casos la variable global de sistema **`errno`** toma el valor de **`ERANGE`**, indicando un error de alcance.
- La estructura **`type div_t`** tiene dos campos:

```
1.  int quot; // guarda el cociente
2.  int rem;  // guarda el resto de la división
```

La biblioteca matemática **`cmath`** presenta una función **`pow(n,b)`** que eleva cualquier número real n a cualquier base b, entera o no, siempre que el valor obtenido no sobrepase el límite máximo del tipo de dato. Como en toda solución general, hay restricciones que el programador debe conocer al emplear funciones "enlatadas". En este caso en particular se tiene que:

- Si a es negativa y n no es entero, ocurrirá un error de dominio (**`domain error`**)
- Si a es cero, n es menor o igual que cero, y el resultado no puede ser representado como un número real, también ocurrirá un error de dominio (**`domain error`**)
- Si el resultado cae fuera de límites del tipo manejado, ocurrirá un error de límites (**`range error`**)
- Con algunos compiladores (que este no es el caso), si a es entero, puede haber ambigüedad, por lo que si se está escribiendo para código lo más portable posible, la función deberá invocarse como:

```
pow(5.0, 2);                    // usando un punto flotante       - preferido
pow(static_cast<double>(5), 2); // usando un encasillamiento nuevo - muy factible
pow(double(5), 2);              // usando un encasillamiento C++   - desaconsejado
```

El sistema también brinda unas constantes que ahorran tiempo y esfuerzo. Ellas son:

Tabla 11. Algunas constantes predefinidas

Constante	Valor
`M_E`	2.7182818284590452354
`M_LOG2E`	1.4426950408889634074
`M_LOG10E`	0.43429448190325182765
`M_LN2`	0.69314718055994530942
`M_LN10`	2.30258509299404568402
`M_PI`	3.14159265358979323846
`M_PI_2`	1.57079632679489661923
`M_PI_4`	0.78539816339744830962
`M_1_PI`	0.31830988618379067154
`M_2_PI`	0.63661977236758134308
`M_2_SQRTPI`	1.12837916709551257390
`M_SQRT2`	1.41421356237309504880
`M_SQRT1_2`	0.70710678118654752440

Si se va a emplear la biblioteca matemática en código nuevo, siempre invocar a **`<cmath>`**

Ciclos y eficiencia

El problema 7 de (Introducción al C++, tomo 1, pág. 66) decía: codificar una tabla de potencias del uno al diez para el cuadrado, cubo e inverso. Ahora se añade la raíz cuadrada y se presenta la tabla nítidamente con tres algoritmos diferentes, demostrando la versatilidad y la facilidad de ampliación de la STL.

Eficiencia

Siguiendo la línea de razonamiento de (Stroustrup), la eficiencia no importa mucho en programas "de juguete" como los que se tratan en un texto de aprendizaje, pues en esos casos, la sencillez y pequeñez priman. No obstante, las aplicaciones comerciales muchas veces se diseñan como sistemas, con partes que requieren cada una de una eficiencia al menos, adecuada.

Code::Blocks suministra un plugin[11] en Plugins->Code statistics que aplicado a un proyecto da el resultado que muestra la figura, con el que se puede tener una idea de la eficiencia tenida al escribir código, que en este caso sería:

$$\eta \approx (1 - \frac{\text{Code only} + \text{Code and comments})}{\text{Total}} = 1 - \frac{32 + 1}{44} \approx 0.25 \text{ ó un } 25\%$$

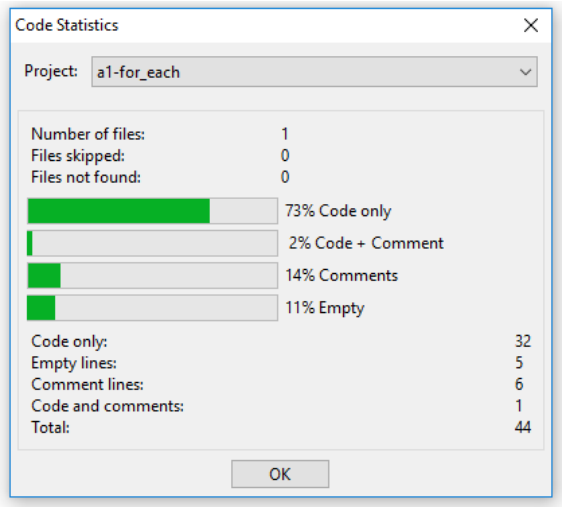


Ilustración 4. Midiendo la eficiencia

Los desarrolladores se cuestionan si pueden usar un tipo de abstracción superior al visitar una colección, pero sin afectar la eficiencia, e incluso, incrementándola. En el texto se presentan tres maneras de hacerlo: dos con sendos algoritmos y una con un **`for`** optimizado, cada uno de ellos aplicado a un tipo diferente de contenedor.

No tema el lector experimentar los ciclos con otros contenedores

Llenado eficiente de números consecutivos

El operador incremental ++ puede aplicarse a cualquier variable numérica (aunque no a una expresión), pero si se aplica a una variable de punto flotante, sólo se incrementa la parte entera. (Introducción al C++, tomo 1, pág. 38). El algoritmo **`iota()`** es un componente numérico extra de la biblioteca estándar que permite sustituir por una sola instrucción el ciclo necesario para introducir una secuencia de números en un contenedor, usando el operador incremental sobre el valor inicial. La secuencia es desde donde se marca el comienzo, hasta donde se marca el final, de uno en uno. Para usarlo, incluir la biblioteca **`<numeric>`**

Su sintaxis es: **`iota(K.begin, K.end, valor_inicial);`**

[11] En las ciencias de la computación se llama plugin a una pieza de código que se incluye en el entorno de trabajo, de modo similar a como se enchufa un efecto electrodoméstico a la corriente.

Arreglos estáticos

En la pila se guardan las variables automáticas. Variables de todo tipo son construidas en su punto de definición y destruidas inmediatamente que salen de ámbito. Su trabajo es mucho más rápido que el almacenamiento dinámico. (Introducción al C++, tomo 1, pág. 147). Un arreglo o **`array`** de la STL emula a un arreglo estático. Montado en la pila, da al programador la seguridad y comodidad que brinda la STL y la eficiencia de la pila.

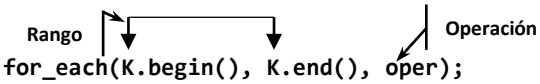
Para usarlo se debe incluir la clase **`<array>`**, sustituta más que adecuada del arreglo nativo de C++, que provee operaciones de E/S de acceso aleatorio con el operador de desplazamiento **`[]`**, usado a riesgo del programador, o con el método **`at()`**, que resguarda del error por pasarse de límites, aunque degrada en algo la eficiencia del proceso. Se le puede hacer aritmética de punteros o aplicar muchos de los algoritmos de la STL.

Tabla 12. (simplificada) Interfaz de la clase array

Constructores	Resultados
`array<T, t>a` `array<T, t>a {}`	Crea un arreglo **a**, del tipo **T**, de tamaño máximo **t,** con sus elementos puestos a cero.
`array<T, t>a {lista_de_valores}`	Crea un arreglo **a**, del tipo **T**, de tamaño máximo **t** con sus elementos puestos según la lista de valores. Si son menos, el resto se pone a cero; si son más hay un error de compilación: **`error: too many initializers for std::array<T, t>`**
`array<t> a(a1)`	Construye **a** como una copia del arreglo **a1**
Métodos	**Resultados**
`v = a.at(p)`	Devuelve el elemento en la posición **p**, y comprueba sus límites. Si son violados lanza una excepción: **`terminate called after throwing an instance of std::out_of_range'; what(): array::at`**
`v = a[p]`	Igual que el anterior, pero no comprueba los límites, por lo que es más eficiente...y más inseguro. Si aquellos son violados sigue tratando de leer y según lo que esté allí posiblemente abortará el programa.
`a = a1`	Copia los valores de **a1** para **a** en una sola instrucción.
`v = a.front()`	Toma el valor del primer elemento. No lo elimina.
`v = a.back()`	Toma el valor del último elemento. No lo elimina.
`n = a.size()`	Retorna el tamaño fijo del arreglo.
`a.swap(a1)`	Intercambia el arreglo **a** con el arreglo **a1.** Ambos son similares en tipo y tamaño.
`b = a1 == a2`	**true** si **a1** y **a2** son iguales.
`b = a1 ¡= a2` `b = a1 not_eq a2`	**true** si **a1** y **a2** son diferentes.
Repetidores	**Resultados**
`array<T, t>::iterator i` `array<T, t>::const_iterator i`	Un repetidor constante o no, de cabeza a cola.
`a.begin()`	La cabeza del arreglo
`a.end()`	El final del arreglo

Uso del algoritmo for_each()

Este algoritmo es muy flexible, pues permite el acceso, proceso y/o modificación de cada elemento de un contenedor. Llama a cada elemento en el rango dado, simplificando la tarea de visitar todos y cada uno de los elementos de un contenedor y hacerles "algo", y le aplica un proceso que desafortunadamente deberá ser codificado aparte. **`for_each`**() devuelve su operación, por lo que se puede tramitar y devolver un resultado dentro de dicha operación. Su sintaxis es:



La eficiencia de este programa es del 27%. Se usó con un arreglo STL. Observar los llamados de la STL y compararlos con el ejemplo que sigue.

Programa auxiliar a1: for_each()

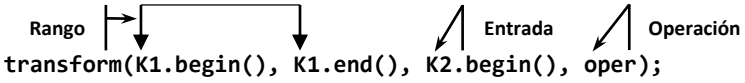
```
#include <array>
#include <iostream>
#include <iomanip>
#include <algorithm>
```

```
#include <numeric>
#include <cmath>

const int N = 10;
using namespace std;

// función para imprimir una línea de valores
void mostrar(int x) {
    cout << setprecision(0) <<
    setw(3) << x            <<
    setw(8) << pow(x, 2)    <<
    setw(8) << pow(x, 3)    <<
    setprecision(4)         <<
    setw(10) << 1.0 / x     <<
    setw(10) << sqrt(x)     << '\n';
}

int main() {
    cout << "Programa auxiliar 1: uso del for_each()\n\nTabla de n\243meros.\n" <<
    //  encabezado
    setw(40) << setfill('-') << '\n' << setfill(' ') << fixed <<
    setw(3)  << "No"         <<
    setw(8)  << "Cuad"       <<
    setw(8)  << "Cubo"       <<
    setw(10) << "Inverso"    <<
    setw(10) << "Ra\241z"    << left <<
    setw(40) << setfill('-') << '\n' << setfill(' ') << right << '\n';

    array<int, N>ary;
    iota(ary.begin(), ary.end(), 1); // llenado
    for_each(ary.begin(), ary.end(), mostrar);
    cout << setw(40) << setfill('-') << '\n';
    system("pause");
    return EXIT_SUCCESS;
}
```

Uso del algoritmo transform()

Este algoritmo optimiza la tarea de hacer una operación con todos y cada uno de los elementos de uno o dos contenedores, operación que desafortunadamente y como en el caso anterior, deberá ser codificada aparte, pero que ahora retorna el valor ya procesado. Viene en dos formas:

1. En la más complicada toma cinco argumentos para manejar dos fuentes y aplicar la operación al rango del destino. Se verá su aplicación un poco más adelante.
2. En la más simple —la que se muestra ahora— toma cuatro argumentos: los dos primeros marcan un rango de aplicación, el tercero es la entrada al destino (que puede ser el mismo contenedor del rango) y el cuarto y último es la operación a ser aplicada, por lo tanto, **`transform()`** copia y modifica los elementos de una sola pasada:



La eficiencia de este programa es del 24%. Observar los llamados de la STL sobre un arreglo normal y como visitarlo, ya que no posee directamente los iteradores **`begin()`** y **`end()`**. Se llama a las funciones y trabaja sin problemas.

Programa auxiliar a2: transform()

```
#include <iostream>
#include <iomanip>
#include <algorithm>
#include <cmath>
#include <numeric>

const int N = 10;
using namespace std;

// función para imprimir una línea de valores
int mostrar(int x) {
    cout << setprecision(0) <<
    setw(3) << x            <<
    setw(8) << pow(x, 2)    <<
    setw(8) << pow(x, 3)    <<
    setprecision(4)         <<
    setw(10) << 1.0 / x     <<
    setw(10) << sqrt(x)     << '\n';
    return x;
}

int main() {
    cout << "Programa auxiliar 2: uso del transform()\n\nTabla de n\243meros.\n" <<
    //  encabezado
    setw(40) << setfill('-') << '\n' << setfill(' ') << fixed <<
    setw(3)  << "No"         <<
    setw(8)  << "Cuad"       <<
    setw(8)  << "Cubo"       <<
    setw(10) << "Inverso"    <<
    setw(10) << "Ra\241z"    << left <<
    setw(40) << setfill('-') << '\n' << setfill(' ') << right << '\n';

    int ary[N];                    // un arreglo normal
    iota(begin(ary), end(ary), 1); // llenado
    transform(begin(ary), end(ary), begin(ary), mostrar);
    cout << setw(40) << setfill('-') << '\n';
    system("pause");
    return EXIT_SUCCESS;
}
```

El ciclo for basado en rangos

Antes de C++11, se tenía que usar **copy** para transferir los resultados en un iterador de salida **ostream_iterator <string>(cout, "\n")**, pero esto es poco elegante y más bien inflexible comparado con usar un ciclo **for**, porque escribir iteradores de salida para cada propósito es bastante involucrado y verboso. (Kalb & Ažman, pág. 47)

La norma C++11 añade una nueva y optimizada forma de hacer ciclos **for**, basada en el rango total del contenedor, que también puede ser sin problemas un arreglo estático[12], y que añade un nuevo nivel de abstracción al lenguaje, simplificando la tarea común de visitarle, como se muestra en el ejemplo y hacerle "algo" que puede ser una o varias acciones. Entre paréntesis van el tipo de dato **T** o la palabra-clave **auto** y la variable del recorrido, separados por dos puntos del objeto a recorrer:

for ([<T>|auto] x: <objeto>) { ... } ***// for***

[12] Los arreglos C++ dinámicos no parecen interactuar fácilmente con este tipo de **for**. Si es el caso usar la instrucción clásica.

La palabra-clave auto

Escribir auto quiere decir que el tipo no tiene importancia; sólo su comportamiento, definido por su uso, es importante. Escribiendo el tipo explícitamente significa que el tipo tiene que ser exactamente así. Puesto de otro modo: los tipos son intencionales todo el tiempo, nunca circunstanciales. (Kalb & Ažman)

La palabra-clave **auto** deja la deducción del tipo básico de dato al compilador porque sólo importa el comportamiento, pero para que sea correcta, la variable debe ser declarada e inicializada al mismo tiempo, como se muestra:

```
auto cantidad = 0;    // un entero
auto caracter = 'A';  // un carácter
auto limite = 101.0;  // un número en punto flotante
```

Por ejemplo, es ilegal una declaración tal como **auto x** y se emite un error de compilación **...declaration of 'auto x' has no initializer**, lo cual significa en español: la declaración '**auto x**' no tiene inicializador.

La palabra-clave **auto** es verdaderamente útil cuando se usa en conjunto con la OOP, pero también para dejar que este ciclo defina por sí mismo el tipo de dato que maneja el objeto al cual se aplica y alguna que otra tarea más, que sin su uso sería bastante dificultosa de realizar. Por ejemplo, la instrucción muestra como imprimir a consola todos los valores de un objeto dado, línea a línea:

```
for ( auto x: <objeto> )
    cout << x << '\n';
```

que se lee: *para cada elemento **x** del objeto **<objeto>***…En este caso la visita es imprimir en consola el valor. Inicialmente la etiqueta **x** representa el primer nodo de la secuencia. Luego de mostrar el primer valor en consola, el ciclo sigue automáticamente, hasta llegar al último. Si se desea modificar el valor de cada nodo, simplemente el elemento se pasa por referencia:

```
for ( auto &x: <objeto> )
    x *= 0.5; // valores a la mitad
```

En el ejemplo que sigue se usa una lista explicada más adelante, en la página 67.

Programa auxiliar a3: uso del for basado en rangos

```
#include <list>
#include <iostream>
#include <iomanip>
#include <algorithm>
#include <numeric>
#include <cmath>

const int N = 10;
using namespace std;

int main() {
    cout << "Programa auxiliar 3: el for basado en rangos.\n\nTabla de n\243meros.\n" <<

    //  encabezado
    setw(40) << setfill('-') << '\n' << setfill(' ') << fixed <<
    setw(3)  << "No"          <<
    setw(8)  << "Cuad"        <<
    setw(8)  << "Cubo"        <<
    setw(10) << "Inverso"     <<
    setw(10) << "Ra\241z"     << left <<
    setw(40) << setfill('-') << '\n' << setfill(' ') << right << '\n';
```

```
    // una lista STL
    std::list<int> ary(N);          // 10 elementos iniciales
    iota(begin(ary), end(ary), 1); // llenado completo de la lista

    // ciclo para imprimir los valores línea a línea
    for ( auto x: ary ) {
        cout << setprecision(0) <<
        setw(3) << x            <<
        setw(8) << pow(x, 2)    <<
        setw(8) << pow(x, 3)    <<
        setprecision(4)         <<
        setw(10) << 1.0 / x     <<
        setw(10) << sqrt(x)     << '\n';
    } // for

    cout << setw(40) << setfill('-') << '\n';
    system("pause");
    return EXIT_SUCCESS;
}
```

La salida

```
Programa auxiliar 3: el for basado en rangos.

Tabla de números.
----------------------------------------
 No    Cuad    Cubo   Inverso      Raíz
----------------------------------------
  1       1       1    1.0000    1.0000
  2       4       8    0.5000    1.4142
  3       9      27    0.3333    1.7321
  4      16      64    0.2500    2.0000
  5      25     125    0.2000    2.2361
  6      36     216    0.1667    2.4495
  7      49     343    0.1429    2.6458
  8      64     512    0.1250    2.8284
  9      81     729    0.1111    3.0000
 10     100    1000    0.1000    3.1623
----------------------------------------
```

Excepto por el encabezamiento particularizado, las tres salidas son iguales y sus eficiencias similares. La de esta en particular es del 30%

Notas:

- Las tres facilidades actúan sobre contenedores previamente llenos, porque el mismo acto de visitar implica la precondición de recorrer una secuencia no-vacía; si esto no se cumple, el resultado está indeterminado y es casi seguro que el programa fallará. Es responsabilidad del programador verificar este hecho.
- No siempre es posible usarlas, porque sus repetidores visitan la lista, lo que por definición consiste en moverse de la cabeza hacia la cola (o viceversa), nodo por nodo. Por lo tanto, no pueden discriminar entre nodos si se desea visitar porciones del contenedor aisladas entre sí. Desde luego, siempre se puede interrumpir la visita basándose en una condición; lo que no se puede hacer (al menos, no fácilmente) es saltarse una parte de la secuencia para continuar en otra. Cuando es el caso, simplemente usar un ciclo **`for`** estándar.

Functores estándar

Se define al functor como una clase especializada que define al **`operador()`**. Muchos algoritmos trabajan con dos parámetros porque el tercero se hizo con valores por omisión. Para los de ordenación, el tipo por omisión es de menor a mayor, o **`a->z`** con el functor **`less<T>`**. Se puede invertir el orden de un contenedor simple usando el functor de ordenamiento de mayor a menor **`greater<T>`**:

```
// ordenamiento z->a de un contenedor
sort( K.begin(), K.end(), greater<T>() );
```

donde **`K`** es el tipo de contenedor y **`T`** es el tipo de dato contenido. La STL brinda varios functores, declarados en la biblioteca **`<functional>`**, que son:

Tabla 13. Functores estándar

Functores unarios	
`negate<T>()`	Cambia el signo.
`logical_not<T>()`	Niega la expresión.
Functores binarios	
`logical_and<T>()`	Y lógico.
`logical_or<T>()`	O lógico.
`plus<T>()`	Suma.
`minus<T>()`	Resta.
`divides<T>()`	División.
`multiplies<T>()`	Producto.
`modulus<T>()`	División modulo.
`equal_to<T>()`	Igualdad.
`not_equal_to<T>()`	Desigualdad.
`less<T>()`	Menor que.
`less_equal<T>()`	Menor o igual a.
`greater<T>()`	Mayor que.
`greater_equal<T>()`	Mayor o igual a.

La importancia que esto tiene es que se pueden usar los adaptadores de funciones, functores que combinan a otros functores consigo mismos, con ciertos valores o con funciones especiales. Estos functores son:

Tabla 14. Adaptadores de funciones

Expresión	Efecto
`bind1st(op, valor)`	operación(valor, parámetro)
`bind2nd(op, valor)`	operación(parámetro, valor)
`not1(op)`	not operación(parámetro)
`not2(op)`	not operación(parámetro_1 , parámetro_2)

Usando estos adaptadores se pueden combinar distintos functores para formar expresiones muy poderosas…y obscuras. Es la llamada programación por composición funcional. Sigue un ejemplo:

Programa auxiliar 4

Se usarán los adaptadores de funciones **`bind1st`** y **`bind2nd`** con el algoritmo **`find_if`** en un sencillo ejemplo de programación por composición funcional.

Uso

```
#include <iostream>
#include <cstdlib>
#include <vector>
#include <numeric>
#include <algorithm>
#include <iterator>

int main() {
    using namespace std;
    cout << "Ejercicio auxiliar 4: ejemplos con binder1st & binder2nd.";
    typedef vector<int>::iterator iter;
    cout << "\n\nCrea un vector:\n  ";
    vector<int> vct( 4 );
    std::iota( vct.begin(), vct.end(), 1);
    copy( vct.begin(), vct.end(), ostream_iterator<int>( cout, " " ));

    int tres = 3;
    cout << "\n\nAhora crea un predicado unario 'igual_a_tres'\n"
            "uniendo el valor tres al functor binario equal_to\n"
            "y lo usa llamando a find_if.";
    binder1st<equal_to<int>> igual_a_tres = bind1st( equal_to<int>(), tres );
    iter it1 = find_if( vct.begin(), vct.end(), igual_a_tres );
    cout << "\n  " << *it1;

    cout << "\n\nY ahora hace lo mismo, pero usando bind2nd,\n"
            "al vuelo. El resultado es el mismo porque\n"
            "el operador == es conmutativo.\n  ";
    iter it2 = find_if( vct.begin(), vct.end(), bind2nd( equal_to<int>(), tres ) );
    cout << *it2 << "\n\n";
    system( "pause" );
    return EXIT_SUCCESS;
}
```

Salida

```
Ejercicio auxiliar 4: ejemplos con binder1st & binder2nd.

Crea un vector :
  1 2 3 4

Ahora crea un predicado unario 'igual_a_tres'
uniendo el valor tres al functor binario equal_to
y lo usa llamando a find_if.
  3

Y ahora hace lo mismo, pero usando bind2nd,
al vuelo. El resultado es el mismo porque
el operador == es conmutativo.
  3
```

La eficiencia es del 19%

Vectores

Si se piensa por un momento en cómo entrar 100 números manualmente por consola, cómo sacarlos manualmente a consola, o como visitarlos uno por uno, inmediatamente se comprende que serían tareas sumamente engorrosas, grandes consumidoras de tiempo y espacio, y con altas probabilidades de salir erróneas en los primeros intentos. (Introducción al C++, tomo 1, pág. 103)

Un vector de la STL o **vector** (en inglés se pronuncia *véctor*) emula a un arreglo dinámico en la tienda gratis, de modo que el programador tiene a su disposición su gran cantidad de memoria, pero sin la carga de gestionarla manualmente. Para usarlo se debe incluir la clase **<vector>**. Sustituto por excelencia del arreglo dinámico, el vector provee operaciones de E/S de acceso aleatorio, esto es, con el operador de solapamiento **[]**, usado a riesgo del programador, o con el método **at()**, que resguarda del error por pasarse de los límites del arreglo.

Emulado como una secuencia que trabaja por la cola, como sus repetidores también son de acceso aleatorio, se le puede hacer aritmética de punteros o aplicar cualquier algoritmo de la STL con muy buena eficiencia.

Tabla 15. (simplificada) Interfaz de la clase vector

Constructores	**Resultados**
vector<T>v	Por omisión: crea un vector vacío **a**, del tipo **T**
vector<T>v(n)	Crea un vector **a**, del tipo **T**, con **n** elementos puestos a cero.
vector<T>v(n, v)	Construye un vector tipo **T**, lleno con **n** copias del valor **v**, también del tipo **T**
vector<T>v(v1)	Construye **v** como una copia del vector **v1**
vector<T>v(v1.ini, v1.fin)	Construye un vector **v** con los valores de **v1**, en el rango **[ini:fin]**
Métodos	**Resultados**
v.reserve(n)	Reserva **n** nodos en el vector
v.assign(n, k)	Sobrescribe **v** con **n** copias del valor **k**
v.assign(v1.ini, v1.fin)	Sobrescribe **v** con los valores de **v1** en el rango **[ini:fin]**
val = v.at(p)	Devuelve el elemento **val** en la posición **p**, y comprueba sus límites. Si son violados lanza una excepción.
val = v[p]	Igual que el anterior, pero no comprueba los límites
i = v.insert(p, val)	Inserta **val** en la posición **p**. Devuelve un repetidor **i** al valor insertado.
v.insert(p, n, val)	Inserta **n** copias del valor **val** en la posición **p**
v.insert(p, v1.ini, v1.fin)	Inserta en **v**, en la posición **p**, los elementos de **v1** en el rango **[ini:fin]**
val = v.front()	El valor del primer elemento (no desenlista.)
val = v.back()	El valor del último elemento (no desencola.)
n = v.size()	Retorna el tamaño real del vector.
v.clear()	Vacía el vector.
b = v.empty()	**true** si el vector está vacío.
v.erase(p)	Borra el elemento en la posición **p**
v.erase(v.ini, v.fin)	Borra los elementos en el rango **[ini:fin]**
v.push_back(k)	Añade el elemento **k** a la cola.
v.pop_back()	Desencola el último elemento.
v.swap(v1)	Intercambia el vector **v1** con el vector **v**
b = v1 == v2	**true** si **v1** y **v2** son iguales.
b = v1 ¡= v2 **b = v1 not_eq v2**	**true** si **v1** y **v2** son diferentes.
Repetidores	**Resultados**
vector<T>::iterator i **vector<T>::const_iterator i**	Un repetidor constante o no, de cabeza a cola.
v.begin()	La cabeza del vector.
v.end()	El final del vector.

Notas:

- En el texto a un vector se le llama simple cuando sus elementos son tipos nativos simples, excluyendo las uniones y estructuras. Cuando sus elementos son otros contenedores, uniones o estructuras se le llama vector estructurado.
- Si el valor numérico entrado no es del mismo tipo que el del vector simple, el sistema hace una conversión implícita que puede perder precisión si va de un punto flotante de doble precisión a uno de precisión simple, o a un entero.
- Las comparaciones entre vectores del mismo tamaño son estrictas, o sea, dos vectores son iguales si sus valores homólogos son iguales —incluyendo arreglos y estructuras; otrosí son diferentes.
- Comparar vectores de tamaños diferentes da respuestas carentes de significación. Aplicar otro tipo de comparación no tiene sentido.
- La acción rápida de inserción o borrado de un elemento sólo es posible por su final, con **`push_back/pop_back`**. En cualquier otra posición no es tan rápida, porque hay que reordenar los nodos intermedios para hacer espacio al entrante; insertar un valor en un vector realmente grande pudiera llegar a notarse en el desempeño del programa. Si ese es el caso, una técnica consiste en encolar los valores y ordenar el vector en el momento de escribirlo a un periférico, lo cual es un muy buen paliativo, casi que un remedio, por la rapidez de la ordenación:

```
sort(v.begin(), v.end());                // de menor a mayor
sort(v.begin(), v.end(), greater<T>());  // de mayor a menor
```

- Como todo contenedor que admita al operador de desplazamiento `[]`, el vector permite la aritmética de punteros. Por ejemplo, **`2+v.begin()`** apunta al tercer elemento, y **`--v.end()`** al penúltimo. Un código para abarcar todo el vector, menos sus puntas, sería:

```
vector<int>::iterator i = v.begin()+1, j = v.end()-1;
```

- Es responsabilidad del codificador que el rango quede entre límites. Si al devolver el valor de un elemento, su posición cae fuera de límites, con el operador de desplazamiento `[]` se obtiene un valor espurio y es muy posible que el programa colapse. Con el operador **`at()`**, se degrada la eficiencia (si esto es o no perjudicial al programa, es decisión del codificador) y frente a un error de límites se lanza una excepción que puede ser manejada por el programa:

```
terminate called after throwing an instance of 'std::out_of_range' what(): vector::_M_range_check
```

Programa 5: producto escalar

El problema 23 del cuarto bloque de ejercicios de (Introducción al C++, tomo 1, pág. 125) decía: el producto escalar de vectores planos (producto-punto) permite determinar ángulos y distancias en el plano real ($\mathbb{R}^2$) de una forma fácil y directa. Sin embargo, añadiendo el contenedor vector simplifica la solución hallada y mantiene la complejidad bajo control. Más rapidez de escritura y de limpieza en el código del desarrollo, sin alguna pérdida de abstracción o eficiencia.

Declaración

Al declarar el vector STL como un miembro de la estructura se está creando una clase por composición y esta forma es la que se usará en el texto. Ahora la estructura es más simple y además puede manejar los métodos del vector y gestionarlo con algoritmos.

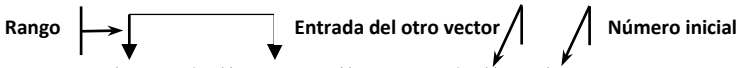
```
#ifndef VECTORES_H_INCLUDED
#define VECTORES_H_INCLUDED

#include <string>
typedef std::string Str;
#include <vector>
typedef std::vector<double> Vct;

//  un vector sabe como entrarse, mostrarse, calcular
//  su producto punto con otro vector y su norma
```

```
struct Vector {
    //  métodos
    double norma() const;
    double pPunto( Vector &B ) const;
    void entrar( Str id );
    void mostrar( Str id ) const;
    //  miembro
    Vct vect;
};

#endif // VECTORES_H_INCLUDED
```

Definición

Se aplica al flujo de entrada el manipulador **skipws** que evita los espacios en blanco dentro del **cin**. Se leen el paréntesis de apertura y la coma de separación, y se usa un método mejor que el dado antes para limpiar el búfer. Al finalizar las lecturas se repone el flujo. Una vez entrados los datos, se pasan al vector.

El producto punto es calculado por el algoritmo **inner_product** que calcula $0.0 + a_1 * b_1 + a_2 * b_2 + \cdots$ para los correspondientes elementos de ambos vectores. Como el neutro para la suma es cero, ese es el que se pone.




```
inner_product(K.begin(), K.end(), B.begin(), n);
```

Para dejar el flujo en buen estado, se limpia. La norma del vector se calcula tomando la raíz cuadrada del producto interno del vector consigo mismo.

```
#include "vectores.h"
#include <iostream>
#include <ios>
using std::cout;
using std::cin;

void Vector::entrar( Str id ) {
    cout << "Vector " << id << " en formato (x, y): ";
    double a, b;
    char parentesis, coma;
    std::ios::fmtflags oldFlags = cout.flags();
    cin.setf( std::ios::skipws );   // evita los espacios en blanco
    cin >> parentesis;              // el paréntesis
    cin >> a;                       // la 1ra coordenada
    cin >> coma;                    // la coma
    cin >> b;                       // la 2da coordenada
    cin.ignore( 255, '\n' );        // deja limpio el búfer
    cout.flags( oldFlags );
    vect.clear();         // vacía el vector por si acaso...
    vect.push_back( a ); // entra la 1ra. coordenada
    vect.push_back( b ); // entra la 2da. coordenada
}

#include "../../deposito/print.h"

void Vector::mostrar( Str id ) const {
    LISTA( vect, "Vector " + id + " (", 2 );
    cout << "\b)\n";
}

#include <numeric>

double Vector::pPunto( Vector &B ) const {
```

```
    return inner_product( vect.begin(), vect.end(), B.vect.begin(), 0.0 );
}

#include <cmath>

//  la norma del vector; se calcula como
//  el producto punto por sí mismo
double Vector::norma() const {
    Vct tmp( vect ); // mismo vector
    return sqrt( inner_product( vect.begin(), vect.end(), tmp.begin(), 0.0 ) );
}
```

Programa

```
#include "vectores.h"
#include <iostream>
#include <cstdlib>
#include <cmath>
#include <iomanip>

const double RAD2GRAD = 45.0 / atan(1.0);
const int DEC = 2; // 2 decimales

int main() {
    std::cout << "Problema 5: c\240lculo del producto-punto de dos vectores\n\n";
    Vector A, B;

    // entradas
    A.entrar( "A" );
    B.entrar( "B" );
    std::cout << std::fixed << std::showpoint << std::setprecision( DEC ) << '\n';

    // calculos y salidas
    double pPto = A.pPunto( B );
    std::cout << "El producto-punto de A y B vale " << pPto << "\n\n";
    std::cout << "Norma de A = " << A.norma() << " u\n";
    std::cout << "Norma de B = " << B.norma() << " u\n";
    double sita = pPto / ( A.norma() * B.norma() );
    std::cout << "\nEl coseno entre A y B vale "      <<
    std::setprecision( DEC + 1 ) << sita << " rad o " <<
    std::setprecision( DEC )     << sita * RAD2GRAD   << " grados.\n";

    if( 1 == sita )
        std::cout  << "A y B son paralelos.\n\n";
    else if( 0 == sita or M_PI == sita )
        std::cout << "A y B son perpendiculares.\n\n";
    else
        std::cout << '\n';

    system( "pause" );
    return EXIT_SUCCESS;
}
```

Salida

```
Problema 5: cálculo del producto-punto de dos vectores

Vector A en formato (x, y): (9.8,6.5)
Vector B en formato (x, y): (5.4,2.1)

El producto-punto de A y B vale 66.57

Norma de A = 11.76 u
Norma de B = 5.79 u

El coseno entre A y B vale 0.977 rad o 55.98 grados.
```

Su eficiencia es del 40%

Valarrays

Los **valarray**s permiten algunas optimizaciones que dan buen rendimiento al procesamiento de arreglos numéricos, pero no fueron diseñadas muy bien. Es un hecho que nadie trató de determinar si la especificación final trabajaba, porque nadie se sentía responsable de ella. El personal que las introdujo abandonó el comité[13] mucho antes de que la norma fuera aprobada. Por ejemplo, para usarlos muy a menudo se requiere de algunas conversiones entre tipos que son inconvenientes y consumidoras de tiempo. (Josuttis, p. Cap. 12.2)

Entre los varios componentes numéricos que ofrece la biblioteca estándar la clase **valarray** (o arreglo de valores numéricos, en español) está diseñada para proveer un manejo óptimo de los componentes de un arreglo numérico unidimensional (lineal) en grupos y realizar operaciones matemáticas bastante complejas con ellos. En este tipo de arreglo, la indización también comienza obligatoriamente en cero.

Tabla 16. (simplificada) Interfaz de la clase valarray

Operaciones de creación	Resultado
`valarray<T> va;` `va.resize(size_t n)`	Crea un **valarray** tipo **T**. Luego de creado y antes de usarlo hay que darle el tamaño con el método **resize()**
`valarray<T> va(n);`	Crea un **valarray** tipo **T** de **n** elementos puestos a cero.
`valarray<T> va(T &a, n);`	Crea un **valarray** de **n** elementos inicializados al valor **a** del tipo **T**
`va(valarray& va);`	Constructor de copia.

Métodos	Resultado
`n = va.size()`	Cantidad de elementos
`val = va.min()`	El valor mínimo del **valarray**.
`val = va.max()`	El valor máximo del **valarray**.
`val = va.sum()`	La sumatoria de todos sus valores.
`va[i] = val`	Entra el valor **val** al elemento iésimo del valarray **va**
Están definidos los operadores lógicos: `==, !=, <, <=, >, >=`	
Están definidas las operaciones aritméticas: `=, -, *, /, %, +=, -=, *= /=` y `%=`	

[13] Josuttis se refiere a los integrantes del comité de estandarización ANSI del C++, llamado X3J16.

Notas:

- El primer índice de un **valarray** siempre es cero.
- El tipo de dato del número **n** siempre es **size_v**.

La clase **valarray** es un seudocontenedor porque no posee el método **push_back()**, no conoce a **begin** y a **end** para sus iteraciones y al apartarse de la interfaz generalizada de la biblioteca estándar, no es muy fácil de manejar. Usarla a fondo requiere de un nivel mayor al de un curso introductorio, pero para lo que se desea, su tratamiento está bien. Sigue un sumario de sus propiedades principales:

- Operadores unarios: **+**, **-**, **not**. Afectan a todos los valores del arreglo.
- Operadores binarios: **+**, **-**, *****, **/**, **%**:
  - Si se aplican a dos arreglos de valores homólogos, el resultado es otro arreglo de valores con sus elementos afectados —sumados, restados, etc.
  - Si se aplica un número sobre un arreglo de valores, el resultado es otro arreglo de valores con sus elementos afectados — sumados al número, restados del número, etc.
- Operadores lógicos: **==**, **!=**, **<**, **<=**, **>**, **>=**:
  - Si se aplican a dos arreglos de valores homólogos, el resultado es otro arreglo, pero de valores booleanos, con cada elemento resultado de aplicar el operador a los elementos homólogos.
  - Si se aplica un número sobre un arreglo de valores, el resultado es otro arreglo de valores con sus elementos afectados, es decir, comparados al número.
- Operadores de asignación y computación: **+=**, **-=**, ***=**, **/=**, **%=**:
  - Si se aplican a dos arreglos de valores homólogos, el resultado es otro arreglo de valores, con cada elemento resultado de aplicar el operador a los elementos homólogos.
  - Si se aplica un número sobre un arreglo de valores, el resultado es otro arreglo de valores con sus elementos afectados —cada elemento sería **número operador valor**.
- Funciones de la biblioteca matemática: (**abs**, **pow**, **exp**, **sqrt**, etc.) Se aplican a uno o a dos arreglos de valores, según la interfaz de la función; el resultado es otro arreglo de valores con cada elemento afectado por la función.

Subconjuntos de un valarray

Su gestión no fue bien diseñada, porque la biblioteca estándar no especificó que un subconjunto debería poder hacer las mismas operaciones de su **valarray**, y por eso, un subconjunto se puede crear muy fácilmente, pero no es tan fácil el combinarlo con otros y en muchas operaciones se requiere de un encasillamiento. (Josuttis, p. Cap. 12.2.2)

Un subconjunto de un **valarray** se define mediante un índice complejo, especial, llamado **slice** (tajada en español) formado a su vez como una terna de valores con sus propiedades listadas en este orden:

1. El primero de los tres valores es el índice de comienzo del subconjunto.
2. Le sigue la cantidad de elementos que le forman, también llamada tamaño (*size*, en inglés.)
3. Y cierra la distancia que hay entre los elementos de su tamaño, también llamada zancada (*stride*, en inglés.)

Por ejemplo **va[slice(1, 2, 3)]** a partir del índice uno (el segundo valor) define un subconjunto de tamaño dos (**29.27** y **22.62**) y los elementos de cada tajada van apartados entre sí por tres lugares (los que hay entre **29.27** y **22.62**), lo cual es mostrado en la figura extraída de la salida del primar programa. Cualquier cálculo se hace con todos sus decimales, pero se imprime con la precisión requerida:

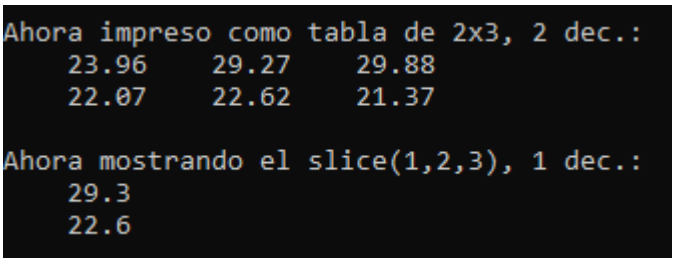


Ilustración 5. Subconjunto de un arreglo de valores

Justificación de las plantillas

Un **valarray**, que es una secuencia numérica lineal puesta en la tienda gratis y manejada por la STL, puede presentarse (y comportarse) como un vector, una tabla o un cubo si se escoge un **slice** adecuado, pero existe un problema cuando se hacen operaciones entre subconjuntos de un **valarray**: no se pueden hacer directas. Por ejemplo, al asignarle a un subconjunto el producto de otros dos, no se puede escribir directamente:

```
va[slice(0,4,3)] = va[slice (1,4,3)] * va[slice (2,4,3)];
```

El compilador "ve" dos **slices** en el lado derecho de la asignación y como el **valarray** no puede hacer operaciones entre sus subconjuntos, da este error:

```
...no match for 'operator*' in 'std::valarray<_Tp>::operator[](std::slice) [with _Tp = double](std::slice(1u, 4u, 3u)) * td::valarray<_Tp>::operator[](std::slice) [with _Tp = double](std::slice(2u, 4u, 3u))'
```

En español: *no hay un casamiento para el operador* en…* Entonces, hay que encasillar ambos subconjuntos:

```
va[slice (0,4,3)] =
   static_cast<valarray<double>>(va[slice (1,4,3)]) * static_cast<valarray<double>>(va[slice (2,4,3)]);
```

Esto no sólo es propenso al error, además es largo y fastidioso de escribir. Y aún es peor: habrá un costo de rendimiento, porque cada encasillamiento primero crea y luego destruye un objeto temporal. (Josuttis, p. Cap. 12.2.2) ofreció una plantilla para aliviar el problema y este autor se tomó la libertad de adaptarla al texto e incorporarla al fichero de plantillas, y añadió una plantilla sobrecargada en dos versiones, para imprimir valores de un **valarray**.

Por último, pero no menos importante, ya se especificó que el índice de un **valarray** es a su vez un conjunto de tres valores y no un simple número, por lo que no se puede usar directamente para mostrar sus valores en ciclos, hay que aplicar el acceso por cálculo de posiciones, técnica mostrada en (Introducción al C++, tomo 1, pág. 156) y vuelta a mostrar aquí:

```
varr[i*columnas + j] = valor; // poniendo un valor en el índice i, j
cout << varr[i*columnas + j]; // mostrando el valor que está en i, j
```

Programa auxiliar a5: uso del valarray

```
#include "../deposito/varray.h"
#include "../deposito/rnd.h"
#include "../deposito/print.h"

using namespace std;
const int FILAS = 3; // filas y columnas; podrían pedirse...
const int COLS  = 4; // ...directamente al usuario

int main() {
    cout << "Programa auxiliar 5: valarray\n\nTrabajando con " << FILAS
         << " filas, " << COLS << " columnas y 2 decimales.\n\n";

    valarray<double>va(FILAS*COLS);      // valarray con 12 números
```

```
    RELLENA_VD(va, FILAS*COLS, 1, 100); // lleno de valores al azar

    cout << std::setprecision(2) << std::fixed << "La tabla:\n";
    PRINTVA(va, COLS);
    cout << "\nEl valor m\240ximo es "   << va.max()
         << "\nEl valor m\241nimo es "   << va.min()
         << "\nEl alcance o rango es "   << va.max() - va.min()
         << "\nLa sumatoria es "         << va.sum()
         << "\nEl valor promedio es de " << va.sum() / va.size() << "\n\n";

    cout << "Extrayendo la ra\241z cuadrada a la tabla:\n";
    va = sqrt(va); PRINTVA(va, COLS);
    cout << "\nLlevando sus valores a la mitad:\n";
    va /= 2; PRINTVA(va, COLS);
    cout << "\nHaciendo la columna 1 = columna 2 x columna 3:\n";
    va[slice(0, 3, 4)] = VA(va[slice(1, 3, 4)]) * VA(va[slice(2, 3, 4)]);
    PRINTVA(va, COLS);
    cout << "\nPoniendo el promedio de las otras en la columna 4:\n";
    va[slice(3, 3, 4)] = (
        VA(va[slice(0, 3, 4)]) +
        VA(va[slice(1, 3, 4)]) +
        VA(va[slice(2, 3, 4)])   ) / static_cast<double>(FILAS);
    PRINTVA(va, COLS);
    cout << '\n';

    system("pause");
    return EXIT_SUCCESS;
}
```

La salida

```
Programa auxiliar 5: valarray

Trabajando con 3 filas, 4 columnas y 2 decimales.

La tabla:
    68.38    32.78    16.38    60.46
    92.51    79.00    76.62    70.59
    83.56    80.40    18.72    46.57

El valor máximo es 92.51
El valor mínimo es 16.38
El alcance o rango es 76.13
La sumatoria es 725.97
El valor promedio es de 60.50
```

```
Extrayendo la raíz cuadrada a la tabla:
     8.27     5.73     4.05     7.78
     9.62     8.89     8.75     8.40
     9.14     8.97     4.33     6.82

Llevando sus valores a la mitad:
     4.13     2.86     2.02     3.89
     4.81     4.44     4.38     4.20
     4.57     4.48     2.16     3.41

Haciendo la columna 1 = columna 2 x columna 3:
     5.79     2.86     2.02     3.89
    19.45     4.44     4.38     4.20
     9.70     4.48     2.16     3.41

Poniendo el promedio de las otras en la columna 4:
     5.79     2.86     2.02     3.56
    19.45     4.44     4.38     9.42
     9.70     4.48     2.16     5.45
```

La eficiencia es del 46%

Notas:

- El valor de los cálculos es completo, aunque se muestren en consola con la precisión dada.
- Debido al factor azaroso, los resultados varían con cada ejecución.

Escalamiento

Escalamiento en este sentido aparece a partir de un programa hecho y funcional, al que se añaden nuevas complejidades, producto de nuevas demandas. Cuando son pocas las alteraciones, basta con integrarlas a lo ya hecho, pero llega muy, muy pronto el momento en que aparecen advertencias del compilador, efectos colaterales molestos cuando menos, y bugs de todo tipo. Entonces se dice que el programa no escaló bien y hay que readaptarlo o sustituirlo por una versión avanzada. (Introducción al C++, tomo 1, pág. 108)

En la vida real, las aplicaciones comerciales por demanda se gestionan en el marco de un protocolo legal que rige la actuación, derechos y deberes tanto del cliente que contrata, como de la empresa que sirve. Pero lo más común durante el desarrollo es que el cliente se aparezca con nuevas demandas, y si se acepta esa petición, al elevarse la complejidad debido a las prestaciones a incluir, no es inusual que cambie todo el panorama y haya que empezar de nuevo, con otras perspectivas, otros diseños y la renegociación del contrato original, condiciones que alargarán el tiempo de entrega y encarecerán el producto final, porque a veces un cambio aparentemente menor implica alteraciones mayores, debido a que el diseño original no escala bien.

Reutilización

Reutilizar las clases existentes cuando se crean nuevas clases y programas es un proceso que ahorra tiempo, dinero y esfuerzo…La reutilización también ayuda a los programadores a crear sistemas más confiables y efectivos, ya que las clases y componentes existentes a menudo han pasado por un proceso extenso de prueba, depuración y optimización del rendimiento. (Deitel & Deitel, 2008, pág. 22)

Cuando en disímiles situaciones se emplea un código previamente hecho, se está en presencia de un fenómeno llamado reutilización, uno de los objetivos a alcanzar mediante el paradigma de la OOP. Como ya se indicó antes, otras vías son el uso de bibliotecas de terceras personas o de las del ISO C++ y en especial, el empleo de la STL con sus componentes, más que suficientes para cubrir un vasto abanico de aplicaciones a implementar de forma relativamente fácil, generando un código minimalista y bueno. La STL es un salto gigante en cuanto al cumplimiento de la visión de la reutilización…Evitar reinventar la rueda usando código existente, siempre que sea posible.

Programa 6: escalamiento y reutilización

En el programa 15 de (Introducción al C++, tomo 1, pág. 104) se tenía como objetivo el trabajar un arreglo numérico lineal estático y se pedía calcular y mostrar la nota promedio (de cero a 100 puntos) de un alumno que cursaba 5 asignaturas y en el programa 16 (pág. 107) se pedía hacer lo mismo, pero para una tabla con seis notas y tres alumnos.

Asignat.	Matemáticas	Español	Física	Química	Inglés
Alumno	63	97	88	77	93

Asignat.	Matemáticas	Español	Física	Química	Inglés	Biología
Alumno 1	63	97	88	77	93	95
Alumno 2	96	87	89	78	80	89
Alumno 3	70	90	86	81	90	82

El atento lector inmediatamente advierte que el segundo programa es una ampliación del primero: es un escalamiento. Aún más, la abstracción general de ambos cubre la creación de tablas con cualquier cantidad de asignaturas y alumnos: eso es reutilización basada en el entorno propio del problema y no en sus particularidades. ¿Resultado? El problema se presta a ser codificado con miras a su reutilización posterior, pero antes hay que precisar, comprender y unificar varios elementos del diseño:

a) La aplicación será un tipo concreto de dato o CDT (*Concrete Data Type*, en inglés.)

- ✓ La unión lógica de las estructuras de datos y sus funciones auxiliares dio origen al concepto de tipo abstracto de dato o ADT. La integración física de datos y funciones del ADT, a su vez dio origen al surgimiento de la OOP, con las clases y sus representaciones, los objetos, dados en un tipo concreto de dato. (Introducción al C++, tomo 1, pág. 17)

Esto significa que el programa estará diseñado para ser aplicado a un entorno específico, pero con datos variables. Aplicado a dicho entorno debe aceptar y procesar todo el conjunto de valores que forma su espacio de datos, aquí compuesto por los nombres de las asignaturas y las notas respectivas.

b) Al pasar a ser una especie de "plantilla" para varios usos, aunque se genera código pitónico, minimalista y bueno, se pierde la innata flexibilidad del hecho a la medida. En la filosofía de la programación moderna de computadoras, es un intercambio muy aceptable.

c) El tipo de dato es importante: bien escogido facilita *enormemente* la tarea. Debido a que ahora las operaciones se hacen sobre una tabla plana, se cambia para un arreglo de valores o **`valarray`** que permite la manipulación de un vector en tajadas.

d) El formato de un solo alumno sale como un vector lineal y no lleva promedios por asignatura. Aunque conceptualmente hablando el problema trabaja con un solo tipo de **`valarray`**, formalmente no: el vector lineal y la tabla plana tienen salidas distintas. Eso hace que se deban separar las tablas planas (con dos o más alumnos) de los vectores lineales (con un alumno.) Hay que particularizar este hecho en el código.

e) Para dar mayor flexibilidad, el usuario decidirá las dimensiones de su tabla (filas y columnas) diciendo la cantidad de alumnos a procesar.

f) Para dar mayor independencia, los datos de cada tabla serán puestos en sendos archivos de texto, (uno para las notas y el otro para los títulos de las asignaturas) desde donde serán leídos.

g) Para tipificar la adquisición de datos, para un alumno sus ficheros se llamarán materias1.txt y notas1.txt; para dos alumnos sus ficheros se llamarán materias2.txt y notas2.txt; y así sucesivamente.

h) Para evitarle al usuario escribir tanto, el programa creará los nombres de ficheros tomando como predefinidos el lugar dónde están y la extensión. Sabiendo que entre el primer código ASCII y el del cero hay una zancada de 48 elementos (**0x30** en hexadecimal), se construye cada nombre de una manera muy simple:

```
const Str PRE("c:\\temp\\bd\\");  // prefacio a los archivos
const Str POS(".txt");            // adenda a los archivos
                ⋮       ⋮       ⋮
std::cout << "\250Cu\240ntos alumnos? ";
(std::cin >> filas).get();
archy = PRE + "materias" + char(filas + 0x30) + POS;  // la ID completa del archivo de materias
                ⋮       ⋮       ⋮
archy = PRE + "notas" + char(filas + 0x30) + POS;     // la ID completa del archivo de notas
```

i) El tema 16 de (Introducción al C++, tomo 1, págs. 190-201) se dedicó completo a conocer cómo hacer operaciones de E/S a ficheros, y en el planteamiento de su problema 40 (pág. 201) se describió el formato CSV (*Comma Separated Values*, o valores separados por comas, en español) para bases de datos sobre ficheros de texto. Ambas técnicas serán aplicadas.

j) Desde luego, no basta con separar por comas los datos numéricos: hay que saber dónde van los elementos que serán los que al final tengan el promedio de los que le anteceden. Por ejemplo, para seis asignaturas más su promedio (siete columnas) los datos, que se recomienda pongan a cero (aunque cualquier valor les sirve), vienen con una zancada de siete lugares, para luego poner los promedios allí:

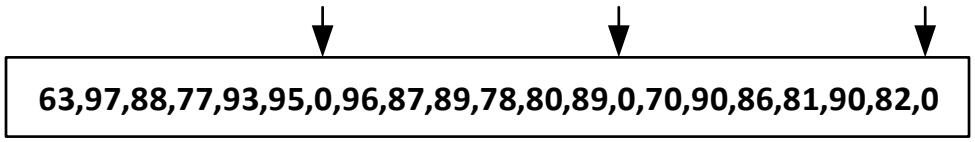


Ilustración 6. Formato de seis notas

Una vez desarrollado y depurado el código, la institución que lo use podrá emitir reportes tabulados de cada aula y grado, solamente creando sus bases de datos y una nomenclatura eficiente para las tablas. Por ejemplo, en este programa y sin mucho esfuerzo puede añadirse otra tabla de tres notas y dos alumnos, para aplicarle el mismo tratamiento:

Asignat.	Matemáticas	Español	Física
Alumno 1	63	97	88
Alumno 2	96	87	89

Sus archivos de datos serían:

```
c:\temp\BD>type materias2.txt
Mat.,Esp.,Fis.,Prom.
c:\temp\BD>type notas2.txt
63,97,88,0,96,87,89,0
c:\temp\BD>
```

Declaración

```
#ifndef OPSES_H_INCLUDED
#define OPSES_H_INCLUDED

// pone una línea doble por omisión o simple, de tamaño n en consola
// input: tamaño
// output: línea en consola
void ponerLinea( int n, bool doble = true );
```

```
#include <string>
typedef std::string Str;
const Str ERR( "El fichero no abri\242. Abortando.\n" );

#include <vector>
typedef std::vector<Str> Vct;

//  carga la BD de materias
//  input: el nombre del fichero
//  output: el vector
Vct cargarMaterias( Str archy );

#include <valarray>
typedef std::valarray<double> Val;

//  carga la BD de notas
//  input: el nombre del fichero y su tamaño
//  output: los valores numéricos
Val cargarNotas( Str archy, int n );

//  input: el valarray y las filas y columnas de la tabla
//  output: la tabla está en consola
Val tabular( const Val & prom, int filas, int cols );

#endif // OPSES_H_INCLUDED
```

Definición

A partir de aquí, el código defensivo se manejará mediante una aplicación sencilla del mecanismo **try-throw-catch**, muy cómodo y seguro: el flujo de entrada nombrado **Leyendo** tiene un método llamado **is_open()** que se pone a **true** si la operación de apertura fue bien y otrosí se pone a **false**. Entonces si no fue bien, se lanza el error adecuado.

Observar como los dos módulos de carga de archivos son muy parecidos. ¿No se podría haber codificado una función que los reuniera? Pues no. Al menos no en este nivel introductorio, porque aquí el vector es un contenedor de cadenas y el **valarray** un seudocontenedor de números, sus tipos de valores son diferentes, sus métodos de carga de valores son otros y no se pueden mezclar. Lo que se hizo fue tratar de estandarizar la forma y el contenido de ambos, conservando sus similitudes y destacando sus incompatibilidades con la marca *// <--*

```
#include "ops_es.h"
#include <iostream>
#include <iomanip>

// pone una línea doble por omisión, o simple, de tamaño n en consola
void ponerLinea( int n, bool doble ) {
    doble ? std::cout << std::setfill( '=' ) : std::cout << std::setfill( '-' );
    std::cout << std::setw( n ) << '\n' << std::setfill( ' ' );
}

#include <cstring>
#include <fstream>

//  nombre de las materias, leídos desde el archivo archy
Vct cargarMaterias( Str archy ) {
    std::ifstream Leyendo( archy );
    if( not Leyendo.is_open() ) throw ERR;  // ¿no abrió? lanza la excepción

    Vct vc;           // <-- vector de cadenas
    Str recd;         // record del archivo
```

```
    char delim = ','; // delimitador CSV de cada campo

    while( Leyendo.good() ) {
        std::getline( Leyendo, recd, delim );
        vc.push_back( recd ); // <-- vector de cadenas
    } // while

    return vc; // <-- vector de cadenas
}

//  valores de las notas leídos desde el archivo
Val cargarNotas( Str archy, int n ) {
    std::ifstream Leyendo( archy );
    if( not Leyendo.is_open() )
       throw ERR;  // ¿no abrió? lanza la excepción

    Val vAry( n );     // <-- valarray de dobles
    Str recd;          // record del archivo
    char delim = ','; // delimitador CSV de cada campo
    int i = 0;         // <-- índice del arreglo

    while( Leyendo.good() ) {
        std::getline( Leyendo, recd, delim );
        vAry[i] = atof( recd.c_str() ); // <-- valarray de dobles
        ++i;
    } // while

    return vAry; // <-- valarray de dobles
}

//  pone los promedios por asignatura en consola
Val tabular( const Val & prom, int filas, int cols ) {
    Val tbl( cols ); // el valarray de promedios

    for( int i = 0; i < cols; ++i ) {
        for( int j = 0; j < filas; j++ )
            tbl[i] += prom[i + j * cols];

        tbl[i] /= filas;
    }

    return tbl;
}
```

Programa

```
#include "../deposito/print.h"
#include "../deposito/varray.h"
#include "ops_es.h"
#include <iostream>
#include <cstdlib>
#include <iomanip>

const Str PRE( "c:/temp/bd/" ); // prefacio a los ficheros
const Str POS( ".txt" );        // adenda a los ficheros

int main() {
    std::cout << "Programa 6: creando tablas, una soluci\242n reutilizable.\n\n";
    Str archy; // la ID de los archivos a leer

    Str filas;//  fijando el archivo de materias
```

```
    std::cout << "\250Cu\240ntos alumnos? ";
    std::getline( std::cin, filas );
    archy = PRE + "materias" + filas + POS; // ID completa
    Vct vMats;                              // vector de las materias

    try {
        vMats = cargarMaterias( archy );
    } catch( Str ) {
        std::cout << ERR;
        return EXIT_FAILURE;
    } // try-catch

    //  fijando el archivo de notas
    int cols = vMats.size();              // las columnas de la tabla
    int ancho = 8 * ( cols + 1 );         // el ancho por columna
    int fils = atoi( filas.c_str() );     // un entero para declarar el valarray
    archy = PRE + "notas" + filas + POS; // ID completa
    Val vAry( fils * cols );              // el arreglo de las notas

    try {
        vAry = cargarNotas( archy, ( fils * cols ) );
    } catch( Str ) {
        std::cout << ERR;
        return EXIT_FAILURE;
    } // try-catch

    // el cálculo del promedio de cada alumno
    for( int i = 0; i < cols - 1; ++i ) {
        vAry[std::slice( cols - 1, fils, cols )] +=
            VA( vAry[std::slice( i, fils, cols )] ) / static_cast<double>( cols - 1 );
    } // for

    // todas las tablas de cualquier tamaño
    std::cout << "\nTabla de resultados\n";
    ponerLinea( ancho );
    MUESTRA( vMats, "Asig." );
    std::cout << std::setprecision( 1 ) << std::fixed << '\n';
    ponerLinea( ancho, false );
    PRINTVA( vAry, cols, "Al. " ); // los promedios por alumno

    if( fils > 1 ) {    // particularizando a más de un alumno ------------
        Val vp( cols ); // el arreglo de los promedios de las notas
        vp = tabular( vAry, fils, cols );
        ponerLinea( ancho, false );
        std::cout << "Prom.";
        PRINTVA( vp, cols - 1 );
    } // if ---------------------------------------------------------------

    std::cout << '\n';
    ponerLinea( ancho ); // cierre de cualquier tabla
    system( "pause" );
    return EXIT_SUCCESS;
}
```

Salida con dos alumnos

```
Programa 6: creando tablas, una solución reutilizable.

¿Cuántos alumnos? 2

Tabla de resultados
=====================================
Asig.    Mat.    Esp.    Fis.   Prom.
-------------------------------------
Al. 1    63.0    97.0    88.0    82.7
Al. 2    96.0    87.0    89.0    90.7
-------------------------------------
Prom.    79.5    92.0    88.5
=====================================
```

La eficiencia es del 55%.

Listas

Crear y mantener manualmente una lista de cualquier tipo requiere un esfuerzo considerable por parte del programador, por lo que modernamente las listas se manejan en C++ con la STL. (Introducción al C++, tomo 1, pág. 166)

En general las listas representan un conjunto de estructuras avanzadas de datos que son autoreferenciadas y están compuestas por nodos enlazados unos a otros. En el primer tomo de esta serie se abordó un estudio elemental del manejo manual de las listas simplemente enlazadas (LSE), con el objetivo pedagógico de fijar la mínima destreza que debe adquirir el programador cuando va a trabajar con punteros, pero ahora la STL viene al rescate.

Una lista STL es tratada en la tienda gratis, de modo que el programador tiene a su disposición su gran cantidad de memoria, pero sin la carga de gestionarla manualmente. El modelo permite una acción rápida de inserción o borrado de un elemento en cualquier posición. Para usarla se debe incluir la clase **`<list>`**, que presenta la siguiente interfaz:

Tabla 17. (simplificada) Interfaz de la clase list

Constructores	**Resultados**
`List<T> l`	Por omisión: crea una lista vacía, del tipo **`T`**.
`List<T> l(l1)`	Crea **`l`** como una copia de la lista **`l1`**
`List<T> l(n)`	Crea una lista con **`n`** nodos con valores al azar.
`List<T> l(n, val)`	Crea una lista con **`n`** nodos, cada uno de valor **`val`**
Métodos	**Resultados**
`l = l1`	Asigna la lista **`l1`** a **`l`**
`l.assign(K.ini, K.fin);`	Construye una lista **`l`** con los valores del contenedor **`K`**, en el rango **`[ini:fin)`**
`l.assign(n, val)`	Asigna a la lista **`n`** nodos con el valor **`val`**
`i = l.insert(p, val)`	inserta el valor **`val`** en la posición **`p`**; devuelve un repetidor al elemento insertado.
`i = l.insert(p, n, val)`	inserta **`n`** copias del valor **`val`** en la posición **`p`**
`i = l.insert(p, K.ini, K.fin)`	Inserta en **`l`**, en la posición **`p`** la secuencia de valores del contenedor **`K`**, en el rango **`[ini:fin)`**
`val = l.front()`	Devuelve el valor de la cabeza. No desenlista.
`val = l.back()`	Devuelve el valor de la cola. No desencola.
`l.push_front(val)`	Enlista al valor **`val`**
`l.pop_front()`	Desenlista la cabeza.
`l.push_back(val)`	Encola al valor **`val`**

`l.pop_back()`	Desencola la cola.
`n = l.size()`	Devuelve la cantidad de nodos en la lista.
`l.clear()`	Vacía la lista.
`b = l.empty()`	**`true`** si la lista está vacía
`l.unique()`	Borra valores duplicados y consecutivos.
`i = l.erase(p)`	Borra el nodo que está en **`p`** y devuelve un puntero al valor que le sigue.
`i = l.erase(l.ini, l.fin)`	Borra los elementos del rango **`[ini:fin)`** y devuelve un puntero al valor que sigue al último elemento borrado.
`l.remove(val)`	Pasa todos los valores iguales a **`v`** al final de la lista.
`l.reverse()`	Vuelve la lista al revés.
`l.swap(l1)`	Intercambia la lista **`l`** con **`l1`**
`l.merge(l1)`	Mezcla la lista **`l`** con **`l1`**. Si están en orden, éste se conserva.
`l.sort()`	Ordena la lista de menor a mayor.
Repetidores	**Resultados**
`list<T>::iterator i` `list<T>::const_iterator i`	Un repetidor constante o no, de cabeza a cola.
`l.begin()`	La cabeza de la lista.
`l.end()`	El final de la lista.

Notas:

- Si los valores asignados no son del mismo tipo que la lista, el sistema hace la conversión, aunque como ya se advirtió, puede perder precisión si va de un número en punto flotante de doble precisión a otro.
- Los repetidores de listas no admiten ni acceso al azar, ni desplazamiento; hay que visitarla para llegar a un elemento.
- Al borrar con **`erase()`** se devuelve un repetidor apuntando al siguiente valor. Si la acción falla, se devuelve **`l.end()`.**
- Para eliminar los valores duplicados usar:

```
l.sort();   // pone consecutivos todos los elementos con el mismo valor
l.unique(); // elimina los duplicados de elementos con el mismo valor consecutivo
```

- Si la lista que llama a mezclar (**`merge`**) no está ordenada, la otra simplemente se añade a partir del punto de entrada (el primer valor de la segunda lista que concuerda con uno de la primera); si está ordenada, la otra se añade en orden. En ambos casos, al trasladar sus valores la otra lista queda vacía.
- El borrado de un solo valor se hace mediante una visita a la lista (que no presente duplicados) que, al encontrar el primer valor deseado, lo elimina y termina la visita.

Programa 7: listas simplemente enlazadas

La solución dada en el problema 24 de (Introducción al C++, tomo 1, pág. 167) se readaptó para manejar la lista en uno de los múltiples modos de C++. En ISO C++ puro el manejo de una lista se hace mediante una clase adecuada, pero la STL permite realizar la tarea con un mínimo de codificación mediante el contenedor **`list`**. Manteniendo la forma con el propósito de comparar ambos programas, éste se adaptó de modo que:

- En la estructura se sustituyó la lista manual por el contenedor STL **`list`**
- Se codificaron las acciones que incluían al contenedor, sus métodos y algoritmos
- Se usó una plantilla para la puesta de sus valores en consola.

Como la lista es simple (es de enteros y un entero es un tipo nativo simple), las acciones del programa principal pueden escribirse directamente y sólo deben sacarse a otro fichero si presentaran uno o más niveles de complejidad que enmascaren la sencillez del código.

Declaraciones

- Menú

```
#ifndef MENU_H_INCLUDED
#define MENU_H_INCLUDED

//  enumera lo que pide el menú
enum { Terminar, Insertar, Enumerar, Buscar, Imprimir, Eliminar, Vaciar };

#include <string>

//  un menú en consola
//  entrada: la información al usuario
//  salida: menú a consola
short menu( std::string info );

#endif // MENU_H_INCLUDED
```

- Lista

```
#ifndef LSE_H_INCLUDED
#define LSE_H_INCLUDED

#include <list>
typedef std::list<int> Lst;

#include <string>
typedef std::string Str;
const Str ERR_V( "Lista vac\241a.\n\n" );
const Str ERR_Y( "Ya est\240.\n\n" );
const Str ERR_N( "No est\240.\n\n" );
const Str HECHO( "Hecho.\n\n" );

struct Nodo {
    Nodo(); // constructor
    // método
    void adiciona( int val, short pos );
    bool enLista( int val );
    //  miembro
    Lst nodo;
};

#endif // LSE_H_INCLUDED
```

Definiciones

- Menú

```
#include "menu.h"
#include <iostream>
#include <cstdlib>
typedef std::string Str;

//  un menú en consola
short menu( Str info ) {
    Str conjunto = "0123456";
    Str opcion;
    Str::size_type idx;
```

```
        while( true ) {
            system( "cls" );
            std::cout << info <<
            "Lista simplemente enlazada\n"
            "1) Insertar un valor\n"
            "2) Contar los nodos\n"
            "3) Comprobar si un valor est\240\n"
            "4) Mostrar la lista\n"
            "5) Eliminar un valor\n"
            "6) Vaciar la lista\n"
            "0) Terminar\n"
            "Entre una opci\242n: ";
            ( std::cin >> opcion ).get();
            idx = conjunto.find( opcion );

            if( Str::npos == idx ) {
                std::cerr << "\nError: opci\242n ilegal.\n\n";
                system( "pause" );
            } else {
                return idx;
            } // if-else
        } // while
    }
```

- Lista: la función de inserción **adiciona** maneja tres opciones con mucha facilidad: a la cabeza, a la cola y en una posición intermedia usando la función **advance** para avanzar al índice requerido; la función **enLista** busca un valor con el algoritmo **count(K.begin, K.end, valor)** y es utilizada varias veces para concluir opciones.

```
#include "lse.h"

Nodo::Nodo() { nodo.clear(); }

void Nodo::adiciona( int val, short pos ) {
    if( 0 == pos )
        nodo.push_front( val );
    else if( unsigned( pos ) > nodo.size() )
        nodo.push_back( val );
    else {
        Lst::iterator i = nodo.begin(); // iterador al inicio
        advance( i, pos - 1 );          // pone al índice en posición
        nodo.insert( i, val );          // se inserta
    } // if-en-cascada
}

#include <algorithm>

bool Nodo::enLista( int val ) { return find( nodo.begin(), nodo.end(), val ) != nodo.end(); }
```

Programa

Incluye la plantilla, que no la hizo el usuario; solamente la utilizó. Usa los mensajes centralizados y trabaja directamente con una lista STL.

```
#include "../deposito/print.h"
#include "menu.h"
#include "lse.h"
#include <iostream>
#include <cstdlib>

int main() {
    Str info( "Problema 7: manejo de una LSE.\n\n" );
```

```
Nodo * head = new Nodo;
short opcion;
int valor, pos, n; // valor, posición y tamaño

while( true ) {
    opcion = menu( info );

    if( Terminar == opcion ) {
        head->nodo.clear();
        delete head;
        return EXIT_SUCCESS;
    } else
        n = head->nodo.size();

    switch( opcion ) {
    case Insertar:
        std::cout << "\250Valor? ";
        ( std::cin >> valor ).get();

        if( head->enLista( valor ) )
            std::cerr << ERR_Y;
        else {
            std::cout << "(0 a la cabeza) \250Posici\242n? ";
            ( std::cin >> pos ).get();
            head->adiciona( valor, pos );
            std::cerr << HECHO;
        }
        break;
    case Enumerar:
        0 == n
            ? std::cerr << ERR_V
            : 1 == n
                ? std::cout << "Un nodo.\n\n"
                : std::cout << n << " nodos.\n\n";
        break;
    case Buscar:
        if( 0 == n )
            std::cerr << ERR_V;
        else {
            std::cout << "\250Valor? ";
            ( std::cin >> valor ).get();
            head->enLista( valor )
                ? std::cout << ERR_Y
                : std::cout << ERR_N;
        }
        break;
    case Imprimir:
        if( 0 == n )
            std::cerr << ERR_V;
        else {
            LISTA( head->nodo, "Lista: " );
            std::cout << "\n\n";
        }
        break;
    case Eliminar:
        if( 0 == n )
            std::cout << ERR_V;
        else {
            std::cout << "\250Valor? ";
            ( std::cin >> valor ).get();

            if( head->enLista( valor ) ) {
```

```
                head->nodo.remove( valor );
                std::cout << HECHO;
            } else
                std::cerr << ERR_N;
        }
        break;
    case Vaciar:
        if( 0 == n )
            std::cout << ERR_V;
        else {
            char procedo;
            std::cout << "Esta operaci\242n destruir\240 la lista.\n\250Procedo? (s/n) ";
            std::cin >> procedo;

            if( 'S' == toupper( procedo ) ) {
                head->nodo.clear();
                std::cout << HECHO;
            } else std::cerr << "Operaci\242n abortada.\n\n";
        }
        break;
    } // switch
    system( "pause" );
  } // while
}
```

Una posible salida

```
Problema 7: manejo de una LSE.

Lista simplemente enlazada
1) Insertar un valor
2) Contar los nodos
3) Comprobar si un valor está
4) Mostrar la lista
5) Eliminar un valor
6) Vaciar la lista
0) Terminar
Entre una opción: 4
Lista: 24, -99, 56

Presione una tecla para continuar . . .
```

La eficiencia es del 67%

Especializaciones de la LSE

De las cuatro especializaciones (pilas, colas, colas prioritarias y bitsets), se describen dos de ellas:

1. **Pilas o LSE Tipo LIFO** (*"last in, first out"* en inglés el último que entra es el primero que sale.) Son aquellas en que la inserción de un nodo siempre es por la cabeza, y esa operación es llamada **`push`** (empujar en español). La remoción de un nodo también siempre es por la cabeza, operación que llaman **`pop`**, sonido onomatopéyico que representa el descorchado de una botella. A la cabeza le llaman **`top`** o tope. Está en la biblioteca **`<stack>`**. Es el tipo más simple, porque el puntero no se mueve: siempre apunta a la cabeza.

   Tabla 18. (simplificada) Interfaz de la clase stack

Constructor	Resultado
`stack<T>p`	Crea una pila vacía `p` del tipo `T`
Métodos	**Resultados**
`p.push(val)`	Apila el valor `val`
`p.pop()`	Desapila el tope
`val = p.top()`	Devuelve el valor del tope
`b = p.empty()`	Devuelve `true` si la pila está vacía
`n = p.size()`	Devuelve la cantidad de nodos en la pila

2. **Colas o LSE Tipo FIFO** (*"first in, first out"* en inglés el primero que entra es el primero que sale) Son aquellas donde la remoción de un nodo siempre es por la cabeza, operación que se llama `dequeue` (desencolar en español) y la adición de un nodo siempre es por el final, operación que se llama `enqueue` (encolar en español), requiriendo de al menos un puntero que pueda viajar desde la cabeza hasta el final. A la cabeza le llaman `top` y a la cola, `tail`. Está en la biblioteca `<queue>`.

Tabla 19. (simplificada) Interfaz de la clase queue

Constructor	Resultado
`queue<T>c`	Crea una cola vacía `c` del tipo **T**
Métodos	**Resultados**
`c.push(val)`	Encola el valor `val`
`c.pop()`	Desencola un nodo
`val = c.front()`	Devuelve el valor de la cabeza
`val = c.back()`	Devuelve el valor de la cola
`b = c.empty()`	Devuelve `true` si la cola está vacía
`n = c.size()`	Devuelve la cantidad de nodos en la cola

Es costumbre representar pictográficamente las colas por la horizontal y las pilas por la vertical, pero es sólo eso: una costumbre.

Mapas y multimapas

Hasta ahora hemos estado estudiando los principales tipos lineales de estructuras de datos: cadenas, arreglos, listas, pilas y colas. Este capítulo define una estructura de datos no lineal llamada árbol. Esta estructura se usa principalmente para representar datos con una relación jerárquica entre sus elementos. (Lipschutz, pág. 245)

Un `map` (llamado *mapa* en español), es un contenedor de la STL que emula a una estructura de árbol guardando pares de valores únicos, tipo clave/valor, ordenados ascendentemente por diseño. Puesto en la tienda gratis, el programador tiene a su disposición esa gran cantidad de memoria, pero sin la carga de gestionarla manualmente.

- ✓ Un árbol es una estructura de datos que gráficamente imita la forma de un árbol invertido mediante nodos conectados entre sí. Las conexiones son las ramas y los nodos son las hojas. Un nodo es la unidad sobre la que se construye el árbol; está conectado a otro nodo llamado su padre y puede tener cero o más puntos de conexión con otros nodos llamados sus hijos. Cuando los puntos son no más de dos al árbol se le llama binario.

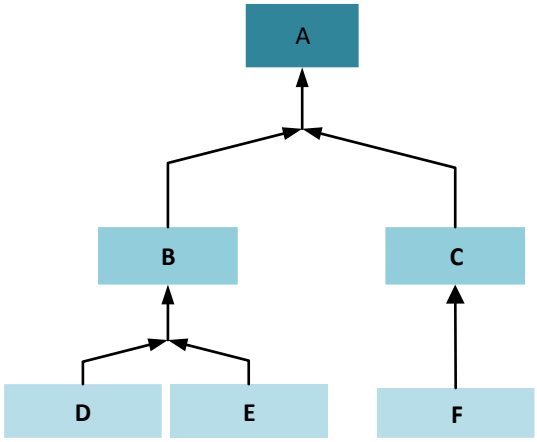


Ilustración 7. Un árbol binario

Un **multimap** (también llamado en español *multimapa* o *mapa múltiple*), es un mapa que puede guardar claves repetidas. Ambos tipos permiten iteradores bidireccionales y en ambos, el valor puede ser recuperado en base a su clave.

Los datos están contenidos en una estructura **pair** (*par* en español) que admite los dos tipos: el de la clave y el del valor, accesibles mediante las llamadas **m->first** para la clave y **m->second** para el valor si el par se pasa como puntero, o **m.first** y **m.second**, si se pasa como referencia. Para usarlos se debe incluir la clase **<map>**. El par puede ser de cualquier tipo legal, pero el par **clave/valor** debe ser asignable, copiable y además la clave debe ser comparable con un criterio de clasificación.

Tabla 20. (simplificada) Interfaz de la clase map

Constructores	Resultados
`map<key, val> m`	Crea un mapa vacío con clave tipo **key** y valor tipo **val**
`map<key, val> m(m1)`	Crea **m** como una copia de **m1**. Ambos son conformes.
`multimap<key, val> mm`	Crea un multimapa vacío con clave tipo **key** y valor tipo **val**
`multimap<key, val> mm(mm1)`	Crea **mm** como una copia de **mm1**. Ambos son conformes.
Métodos	**Resultados**
`n = m.size()`	Devuelve la cantidad de nodos.
`b = m.empty()`	Devuelve **true** si el contenedor está vacío.
`m.clear()`	Vacía el contenedor.
`m.erase(key)`	Borra el/los elementos con clave **key**
`m.erase(i)`	Borra el elemento en la posición **i**
`m.erase(i,j)`	Borra los elementos entre **(i:j)**
Operaciones de comparación	**Resultados**
`b = m == m1;`	**true** si ambos son iguales
`b = m != m1;` `b = m not_eq m1;`	**true** si ambos son distintos
Operaciones de asignación	**Resultados**
`m = m1`	Sobrescribe los elementos de **m1** en **m**
`m.swap(m1)`	Intercambia **m** con **m1**
`[p = ]m.insert(make_pair(key,val));`	Entra la clave **key** y el valor **val** al contenedor **m**. **p** es un par: **p.first** es la posición de inserción; **p.second** es **true** si se insertó.
Operaciones de búsqueda	**Resultados**
`n = count(key)`	Devuelve la cantidad de elementos con la clave **key**
`n = m.count(val)`	Devuelve la cantidad de elementos con el valor **val**
`i = find(key)`	Devuelve el índice del primer elemento con la clave **key**; otrosí devuelve **end()**.
`p = m.find(key);`	Devuelve un par **p**. Si está, la clave va en el índice **p.first** y el valor en **p.second**.

	Si no está, **`i.first`** toma el valor **`m.end()`**
`i = lower_bound(key)`	Devuelve el índice del primer elemento con la clave **`key`**
`i = upper_bound(key)`	Devuelve el índice del último elemento con la clave **`key`**
Repetidores	**Resultados**
`map<key, val>::iterator i;` **`map<key, val>::const_iterator i;`**	Un repetidor constante o no, de cabeza a cola
`m.begin()`	El inicio del mapa.
`m.end()`	El final del mapa.
`multimap<key, val>::iterator i;` **`multimap<key, val>::const_iterator i;`**	Un repetidor constante o no, de cabeza a cola
`mm.begin()`	El inicio del multimapa.
`mm.end()`	El final del multimapa.

Notas:

- Lo que se muestra se aplica para mapas y multimapas, sólo que estos últimos toman entradas repetidas.
- Una vez entrada la clave, nunca deberá alterarse, pues el árbol pierde su sincronía y queda inservible. Para alterarla correctamente, primero hay que borrarla, luego entrar su nuevo valor y por último añadirla al mapa. Es responsabilidad del programador.
- Si se hace una operación con rangos, estos deben quedar entre límites o el programa se cuelga. Es responsabilidad del programador.
- Al insertar un par clave/valor se debe tener en cuenta que los tipos deben ser los correctos o en su defecto, aplicarles encasillamientos adecuados. Es responsabilidad del programador.
- Dos árboles binarios son semejantes si tienen la misma estructura. Si un árbol B es semejante a otro A y sus valores son iguales, se dice que B es copia de A y viceversa.
- En las operaciones de comparación, los contenedores tienen que ser semejantes —mapas con mapas y multimapas con multimapas— y los árboles serán semejantes.
- Se pueden insertar mapas múltiples en mapas y viceversa, siempre que los árboles sean semejantes. El mapa purga los duplicados que se encuentra en el multimapa insertado.
- El destructor sólo borra los elementos del mapa. Si alguno de estos es un puntero a la tienda gratis, la liberación de esa memoria es exclusivamente responsabilidad del codificador.
- Se recomienda hacer una definición por **`typedef`** para evitar la repetición del tipo, o errores en su escritura, cuando se está empleando con mucha frecuencia un tipo largo o confuso de escribir. Por ejemplo:

```
typedef map<string, float> mapaSF;
typedef multimap<string, float> bolsaSF;
```

- En los mapas múltiples, las operaciones de muestra de claves se hacen mediante ciclos adecuados. Por ejemplo:

```
for ( auto x: bolsaSF )
    cout << x.first << ": " << x.second << '\n';
```

Los mapas facilitan el conteo o la agrupación por categorías de elementos de un conjunto dado, pero sobre todo sirven para acceder rápidamente a un elemento mediante una clave de búsqueda pareada a un valor, como en los directorios telefónicos, registros, listas de favoritos, árboles genealógicos, tablas de contenido, anotaciones, etc.

Programa 8: países y monedas

Las monedas nacionales son parte de la identidad de los países, pero hay monedas que son compartidas por más de uno. Las principales divisas en el mundo son el euro, el dólar estadounidense, el yen japonés, la libra esterlina, el yuan chino y el dólar canadiense. También se pueden considerar divisas el Franco Suizo y el Rublo Ruso.

El estándar internacional ISO 4217 fue creado con el objetivo de definir códigos de tres letras para todas las divisas del mundo, eliminando las confusiones introducidas por algunas, como el dólar, el franco o la libra, que son utilizados en numerosos países, pero tienen cambios muy diferentes. La tabla que sigue muestra la moneda y los países que la comparten, compilada y editada de (Wikipedia en español), artículos Divisa e ISO 4217.

Tabla 21. Países y monedas

ISO	Nombre	Países
EUR	**Euro**	Alemania, Andorra, Austria, Bélgica, Chipre, Ciudad del Vaticano, Eslovaquia, Eslovenia, España, Estonia, Finlandia, Francia, Grecia, Guadalupe, Guyana Francesa, Irlanda, Italia, Kosovo, Letonia, Lituania, Luxemburgo, Malta, Martinica, Mayotte, Montenegro, Países Bajos, Polinesia Francesa, Portugal, Reunión, San Bartolomé, San Martin, San Pedro y Miquelón, y Wallis y Futuna.
USD	**Dólar USA**	Estados Unidos, Área insular de Estados Unidos, Caribe Neerlandés, Ecuador, El Salvador, Estados Federados de la Micronesia, Guam, Islas Marianas del Norte, Islas Marshall, Islas Turcas y Caicos, Islas Ultramarinas de Norteamérica, Islas Vírgenes Británicas, Islas Vírgenes norteamericanas, Palaos, Panamá, Puerto Rico, Samoa Americana, Territorio Británico en el Océano Indico, Timor Oriental y Zimbabue.
CAD	**Dólar Canadiense**	Canadá, Territorios del Noroeste, Territorio del Yukón y Territorio de Nunavut.
GBP	**Libra inglesa**	Reino Unido, Acrotiri y Dhekelia, Anguila, Bailía de Guernesey, Bailía de Jersey, Bermudas, Gibraltar, Isla de Mann, Islas Caimán, Islas Georgia del Sur y Sándwich del Sur, Islas Malvinas, Islas Pitcairn, Islas Santa Helena, Ascensión y Tristán de Cuña, Islas Turcas y Caicos, Islas Vírgenes Británicas, Montserrat, Territorio Antártico Británico, y el Territorio Británico en el Océano Indico.
AUD	**Dólar Australiano**	Australia, Territorios Antárticos Australianos, Islas Christmas, Islas Cocos, Islas Heard y McDonald, Islas Norfolk, Kiribati, Nauru y Tuvalu
JPY	**Yen**	Archipiélago japonés y Zimbabue.
CNY	**Yuan**	La República Popular China
CHF	**Franco Suizo**	Suiza, Busingen am Hochrhein , Campione d'Italia y Liechtenstein.
RUB	**Rublo**	La Federación Rusa, Abjasia, Bielorrusia, Osetia del Sur y Transnistria.

¿Cuándo el lector ha de hacer algo de diseño al resolver un ejercicio y cuándo no? EODA, eso es impredecible. Entre otras muchas cosas, depende del planteamiento del problema y de la habilidad innata o adquirida del estudiante.

En este caso, el planteamiento es sencillo y la codificación no debe dar problemas, pero hay una situación algo inesperada: hasta el momento, sólo se ha pedido el dato al usuario cuando era necesario, pues para no distraer al ocupado lector con detalles superfluos, la mayoría de las veces los valores se embebieron en el código, técnica conocida como *hard-coding* en inglés y que no es recomendada en la práctica, pues vincula demasiado el problema a su solución y se desentiende del entorno, pero tiene sus ventajas pedagógicas. Más acá se manejan decenas de pares que unen un tipo de divisa con los países que las emplean como moneda nacional —el *hard-coding* es totalmente impráctico y pedir el dato al usuario no tiene caso: *es el usuario quien interroga al programa*.

Se pide codificar un programa que permita al usuario (mediante un menú) saber cuáles son los países que comparten estas monedas. ¿Solución? Recordar que el tema 16 de (Introducción al C++, tomo 1, págs. 190-201) se dedicó completo a conocer cómo hacer operaciones de E/S a ficheros, y en el planteamiento de su problema 40 (pág. 201) se describió el formato CSV para bases de datos sobre ficheros de texto. Ambas técnicas serán aplicadas. Esos conocimientos se emplearán para hacer un fichero de base de datos en formato CSV, que será puesto en la dirección C:\temp\BD\monedas.txt, desde donde el programa cargará sus datos. Un extracto de la representación de ese fichero sería algo así como:

```
c:\temp\BD>type monedas.txt
EUR,Alemania
EUR,Andorra
EUR,Austria
EUR,Bélgica
EUR,Chipre
EUR,Ciudad del Vaticano
EUR,Eslovaquia
EUR,Eslovenia
EUR,España
EUR,Estonia
EUR,Finlandia
```

Ilustración 8. Un archivo CSV

Declaraciones

- **menú.h**

```
#ifndef MENU_H_INCLUDED
#define MENU_H_INCLUDED

//  enumera lo que pide el menú
enum { FIN, EUR, USD, CAD, AUD, GBP, JPY,  CNY, CHF, RUB };

#include <string>

//  un menú en consola
//  entrada: la información al usuario
//  salida: menú a consola
int menu( std::string info );

#endif // MENU_H_INCLUDED
```

- **monedas.h**

```
#ifndef MONEDAS_H_INCLUDED
#define MONEDAS_H_INCLUDED

#include <string>
typedef std::string Str;
const Str ERR( "El fichero no abri\242. Abortando.\n" );

#include <map>
typedef std::multimap<Str, Str> Moneda;

//  carga la BD
//  input: el nombre del fichero
//  output: el mapa cargado
Moneda cargar( Str archy );

//  muestra los países por moneda
//  input: el código y el mapa
//  output: la respuesta en consola
void mostrar( Str iso, Str nombre, const Moneda & moneda );

#endif // MONEDAS_H_INCLUDED
```

Definiciones

- **menú.cpp**

```
#include "menu.h"
#include <iostream>
#include <cstdlib>
#include <string>

//  un menú en consola
int menu( std::string info ) {
    std::string opcion, conjunto = "0123456789";
    std::string::size_type idx;

    while( true ) {
        system( "cls" );
        std::cout << info <<
        "Divisa\n" <<
        "1) Euro.\n"
        "2) D\242lar americano.\n"
        "3) D\242lar canadiense.\n"
        "4) D\242lar australiano.\n"
        "5) Libra esterlina.\n"
        "6) Yen japon\202s.\n"
        "7) Yuan chino.\n"
        "8) Franco suizo.\n"
        "9) Rublo ruso.\n"
        "0) Terminar.\n"
        "Entre una opci\242n: ";
        ( std::cin >> opcion ).get();
        idx = conjunto.find( opcion );

        if( std::string::npos == idx ) {
            std::cerr << "\nError: opci\242n ilegal.\n\n";
            system( "pause" );
        } else {
            return idx;
        } // if-else
    } // while
}
```

- La función **rellena** carga la base de datos y hace la defensa ya clásica de cerciorarse de que el fichero se abrió; la extracción se hace con la forma completa de **getline()** y en la medida en que va leyendo, va empaquetando los dos valores con un **make_pair**, permitiendo tratarlos como una unidad[14]. En la función de vista **mostrar**, **count** cuenta la cantidad presente de claves y se discrimina entre la libra esterlina (género femenino) y el resto de las monedas (género masculino.) El **for** basado en rangos se encarga de la salida a consola.

```
#include "monedas.h"
#include <fstream>

//  carga la BD
Moneda cargar( Str archy ) {
    std::ifstream Leyendo( archy );
    if ( not Leyendo.is_open() ) throw ERR; // abortar si falló

    Moneda temp;      // un multimapa temporal
```

[14] Empleada en varias partes de la biblioteca estándar, **make_pair** apoya a los contendores **tuple**, **map** y **multimap**, pero puede ser aplicada allí donde se necesite, especialmente en funciones que deban retornar dos valores.

```
        Str iso, nombre; // las variables

        while ( Leyendo.good() ) {
            std::getline( Leyendo, iso, ',' );         // una lectura
            std::getline( Leyendo, nombre, '\n' );     // la siguiente
            temp.insert( std::make_pair( iso, nombre ) ); // inserta el par
        } // while

        return temp;
    }

    #include <iostream>
    using std::cout;

    //  muestra los países por moneda
    void mostrar( Str iso, Str nombre, const Moneda & moneda ) {
        int n = moneda.count( iso );
        cout << '\n';

        if ( "GBP" == iso )
            cout << n << " pa\241ses usan la libra esterlina:\n\n";
        else
            1 == n
                ? cout << "Un pa\241s usa el " << nombre << ":\n\n"
                : cout << n << " pa\241ses usan el " << nombre << ":\n\n";

        for ( auto x : moneda )
            if( x.first == iso ) cout << "  " << x.second << '\n';

        cout << '\n';
    }
```

Programa

```
#include "monedas.h"
#include "menu.h"
#include <iostream>
#include <cstdlib>

const Str ARCHY( "c:/temp/BD/monedas.txt" );

int main() {
    Str info( "Programa 8: pa\241ses que usan divisas\ncomo moneda nacional.\n\n" );
    Moneda paises;
    Str nombre, iso;
    int opcion;

    try {
        paises = cargar( ARCHY );
    } catch( Str ) {
        std::cout << ERR;
        return EXIT_FAILURE;
    } // try-catch

    while( true ) {
        opcion = menu( info );

        if( FIN == opcion ) return EXIT_SUCCESS;

        switch( opcion ) {
        case EUR:
```

```
            nombre = "Euro";
            iso = "EUR";
            break;
        case USD:
            nombre = "D\2421ar americano";
            iso = "USD";
            break;
        case CAD:
            nombre = "D\2421ar canadiense";
            iso = "CAD";
            break;
        case JPY:
            nombre = "Yen japon\202s";
            iso = "JPY";
            break;
        case CNY:
            nombre = "Yuan chino";
            iso = "CNY";
            break;
        case GBP:
            nombre = "Libra esterlina";
            iso = "GBP";
            break;
        case CHF:
            nombre = "Franco suizo";
            iso = "CHF";
            break;
        case RUB:
            nombre = "Rublo ruso";
            iso = "RUB";
            break;
        case AUD:
            nombre = "D\2421ar Australiano";
            iso = "AUD";
            break;
        } // switch

        mostrar( iso, nombre, paises );
        system( "pause" );
    } // while
}
```

Salida

```
Programa 8: países que usan divisas
como moneda nacional.

Divisa
1) Euro.
2) Dólar americano.
3) Dólar canadiense.
4) Dólar australiano.
5) Libra esterlina.
6) Yen japonés.
7) Yuan chino.
8) Franco suizo.
9) Rublo ruso.
0) Terminar.
Entre una opción: 6

2 países usan el Yen japonés:

  Archipiélago japonés
  Zimbabue
```

La eficiencia es del 32%

Tuplas

El contenedor **`tuple`** es la generalización del utilitario **`pair`**. Este último sólo contiene dos valores que pueden ser de diferentes tipos, pero el **`tuple`** se declara recursivamente y por ello puede tener muchísimos elementos, aunque lo usual es una cantidad razonable[15]. El autor.

El contenedor **`<tuple>`** es una generalización que extiende el concepto del **`pair`** más allá del par de valores. La tupla (que así es como se llama en español) permite que se asocien entre sí una cantidad arbitraria de distintos tipos y que puedan ser tratados como ente único, algo así como una **`struct`** POD[16] barata y restringida, pero mucho más liviana al SO, con las ventajas de la STL. Por ejemplo, **`tuple<string, string, double>tri("gal", "l", 3.78)`** es una instrucción que incluye las cadenas **`gal`** y **`l`**, y el valor de punto flotante **`3.78`** en el trío **`tri`**.

Por lo común este contenedor es utilizado para devolver limpiamente múltiples valores de una función y también como sustituto de mapas y conjuntos en problemas que requieren de éstos, pero cuyos datos no se les ajustan. Para usarle se debe incluir la clase **`<tuple>`**. Se aclara al lector que, en matemáticas una tupla es una lista ordenada de elementos. Una n-tupla es una lista ordenada de n elementos, siendo n un número entero no negativo. El término es una generalización de: dupla, dúo o par; tripla, trío, tripleta o terna; cuádrupla, cuarteto o cuarteta; quíntupla o quinteto; séxtuplo o sexteto, séptuplo o septeto; óctuplo u octeto, nónuplo, décuplo y de ahí en adelante, n-tupla (tupla enésima.)

---

[15] Siempre teniendo presente que razonable es un término muy subjetivo.

[16] Recordar que para el propósito de esta serie un POD es un tipo nativo (**`int`**, **`double`**, etc.) o una estructura sin constructores, métodos o miembros. La idea es tener en C++ un tipo al que se le pueda hacer una copia fiel (byte-por-byte.)

También se puede aplicar la tupla para empacar los valores de múltiples variables, puesto que su pase como parámetro es muy limpio. Como ninguno de sus valores se considera una clave para búsquedas, este contenedor no posee repetidores ni accesos aleatorios. Si hay que hacer algo con algún valor, hay dos caminos: desempacar la tupla y obrar en consecuencia, o extraer directamente el valor requerido para la operación.

Tabla 22. (simplificada) Interfaz de la clase tuple

Constructor	Resultado
`tuple<T1, T2, ...> t`	Crea una tupla **t** vacía con los tipos **T1**, **T2**, …
`tuple<T> t {v1, v2, ...};`	Crea una tupla **t** con el tipo **T** y le empaca los valores **v1**, **v2**, …
Operaciones de asignación	**Resultados**
`t = t1`	Sobrescribe los elementos de **t** con los de **t1**
`[auto | T] t = make_tuple(v1, v2, ...);`	Llena la tupla **t** con los valores **v1**, **v2**, … Debe haber correspondencia entre tipos de variables y tupla.
`tie(v1, v2, ...) = t;`	Desempaca los valores de la tupla **t** a las variables **v1**, **v2**, …. Debe haber correspondencia entre tipos de variables y tupla.
`v = get<IDX>(t)`	Extrae el valor iésimo de la tupla **t** y lo pone en **v**. **IDX** es un índice constante, la tupla está indizada a cero y los tipos de datos de la tupla y la variable son iguales.
Operaciones de comparación	**Resultados**
`b = t1 == t2`	**true** si ambas tuplas son iguales.
`b = t1 != t2;` `b = t1 not_eq t2`	**true** si ambas tuplas son distintas.

Programa 9: calculadora de conversión

Escoger un modelo de datos que convenga al problema a solucionar no es trivial; de hecho, esos tópicos se estudian a fondo en cursos avanzados de Estructuras de Datos. No obstante, las preferencias personales junto a las experiencias del diseñador o del codificador muchas veces pesan bastante sobre el modelo escogido, facilitando o entorpeciendo la producción del código. Por ejemplo, la calculadora de conversión dada en el problema 26 de (Introducción al C++, tomo 1, pág. 126), que emula una muy pequeña parte de la función de conversión de la calculadora HEXelon MAX 6, por su nivel de dificultad fue resuelta en el tomo anterior mediante estructuras C++, pero ahora, haciendo una composición con una tupla, que con su gran poder de sintetización se adapta más a una solución eficiente, dando como resultado un código menor, más rápido de escribir y más claro de interpretar.

El problema decía: en las calculadoras modernas se puede buscar la equivalencia en el cambio de unidades entre magnitudes físicas. Codificar un programa que emule parte de la función de conversión de la calculadora HEXelon MAX 6, hecha por Jerzy Znamirowski (www.HEXelon.com) y buscar la equivalencia para: galón(gal)/litro(l); pie cúbico(ft^3)/metro cúbico(m^3); libra(lb)/kilogramo(kg); acre(ac)/hectárea(ha); grado Fahrenheit(F)/grado Celsius(C); radián(rad)/grado (grado); el radián se debe dar al menor ángulo posible.

Declaraciones

- Menú

```
#ifndef MENU_H_INCLUDED
#define MENU_H_INCLUDED

//  enumera lo que pide el menú
enum { Fin, VolLiq, VolSol, Masa, Area, Largo, Temp, Ang };

#include <string>
```

```
typedef std::string Str;

//  un menú en consola
//  entrada: la información al usuario
//  salida: menú a consola
short menu( Str info );

#endif // MENU_H_INCLUDED
```

- Se define un quinteto llamado **Datos** y se añade a la estructura **Conver** formando una clase por composición. Ahora la estructura puede manejar los métodos de la tupla y gestionar sus componentes mediante el miembro **factorizar**, no importa si está en la pila o en la tienda libre.

```
#ifndef CONVERSION_H_INCLUDED
#define CONVERSION_H_INCLUDED

#include "menu.h"
#include <tuple>
typedef std::tuple<double, double, double, Str, Str> Datos; // datos

struct Conver {
    //  métodos:
    void factorizar( short opcion, double valor );
    void mostrar   () const;
    //  miembros
    Datos param;  // un paquete con los datos
};

#endif // CONVERSION_H_INCLUDED
```

Definiciones

- Menú

```
#include "menu.h"
#include <iostream>
#include <cstdlib>

short menu( Str info ) {
    Str opcion, conjunto = "01234567";
    Str::size_type opc;

    while ( true ) {
        system( "cls" );
        std::cout << info <<
        "1) Gal\242n USA(gal) y Litro(l)\n"
        "2) Pie c\243bico(ft3) y metro c\243bico(m3)\n"
        "3) Libra(lb) y Kilogramo(kg)\n"
        "4) Acre (ac) y Hect\240rea(ha)\n"
        "5) Milla(mi) y Kil\242metro(km)\n"
        "6) Grados Fahrenheit(F) y Celsius(C)\n"
        "7) Radi\240n(rad) y grado(grado)\n"
        "0) Terminar\n\n"
        "Entre una opci\242n: ";
        std::cin >> opcion;
        opc = conjunto.find( opcion );

        if ( Str::npos == opc ) {
            std::cerr << "\nError: opci\242n ilegal.\n\n";
            system( "pause" );
        } else {
```

```
            return opc;
        } // if-else
    } // while
}
```

- Conversión: según sea la opción tomada, la tupla empaca los datos necesarios en un paquete. Para mostrarlos, los desempaca a sendas variables conformes y las usa. El pase de parámetros es ahora más limpio. Como se declara una variable dentro de un **case**, éste va de último para no provocar un error de violación de salto:

```
|45|error: jump to case label [-fpermissive]       // 45 es la línea donde está case Temp
|39|error: crosses initialization of 'double vmt'  // 39 es la línea donde se declara vmt
```

  Con un factor **k** de conversión de unidades inglesas a métricas, prácticamente el cálculo de todos los datos es lo mismo: el valor convertido a unidades inglesas es **valor/k** y el valor convertido a unidades métricas es **valor*k**. Los factores fueron tomados de la misma HEXelon MAX 6. Caso aparte son el cálculo de la temperatura que no es tan simple (el valor en unidades inglesas es **1.8*valor+k** y el valor en unidades métricas es **(valor-k)/1.8**; y el cálculo de los ángulos pues hay que calcular el valor de grados a radián y aplicar al valor obtenido la función matemática **fmod** para reducirlo al mínimo de vueltas.

```
#include "conversion.h"
#include <cmath> // para fmod()
using std::make_tuple;

//  dispone todos los valores
void Conver::factorizar( short opcion, double valor ) {
    if ( Fin == opcion ) return; //  el final no se procesa

    double k; // factor primario de conversión

    switch ( opcion ) {
    case VolLiq:
        k = 3.785412;
        param = make_tuple( valor, valor / k, valor * k, "gal", "L  " );
        break;
    case VolSol:
        k = 0.028317;
        param = make_tuple( valor, valor / k, valor * k, "ft3", "m3 " );
        break;
    case Masa:
        k = 0.453593;
        param = make_tuple( valor, valor / k, valor * k, "lb ", "kg " );
        break;
    case Area:
        k = 0.404686;
        param = make_tuple( valor, valor / k, valor * k, "ac", "ha" );
        break;
    case Largo:
        k = 1.609344;
        param = make_tuple( valor, valor / k, valor * k, "mi", "km" );
        break;
    case Temp:
        k = 32.0;
        param = make_tuple( valor, 1.8 * valor + k, ( valor - k ) / 1.8, "F", "C" );
        break;
    case Ang:
        k = 45.0 / atan( 1.0 );
        // eliminar las vueltas completas
        double vmt = valor * k;
        if ( vmt > 360.0 ) vmt = fmod( vmt, 360.0 );
        param = make_tuple( valor, valor / k, vmt, "rad  ", "grado" );
        break;
```

```
        } // switch
    }

    #include <iostream>
    #include <iomanip>

    void Conver::mostrar() const {
        int ancho   = 8;
        int decimal = 4;
        double  val,   // valor a convertir
                valEN, // valor convertido a inglés
                valMT; // valor convertido a métrico
        Str     uniEN, // unidad inglesa
                uniMT; // unidad métrica

        tie( val, valEN, valMT, uniEN, uniMT ) = param;

        std::ios::fmtflags oldFlags = std::cout.flags(); // formato original
        std::cout << setiosflags( std::ios::fixed | std::ios::showpoint )
                  << std::setprecision( decimal );
        //  primera línea a consola
        std::cout << '\n' <<
        std::setw( ancho )  << val   << ' ' << uniEN << " --> " <<
        std::setw( ancho )  << valMT << ' ' << uniMT;
        //  segunda línea a consola
        std::cout << '\n' <<
        std::setw( ancho )  << val   << ' ' << uniMT << " --> " <<
        std::setw( ancho )  << valEN << ' ' << uniEN;
        std::cout.flags( oldFlags ); // formato original restaurado
    }
```

Programa

```
#include "conversion.h"
#include <iostream>
#include <cstdlib>

int main() {
    Str info( "Problema 9: calculadora de conversi\242n.\n\n" );

    //  variables
    short opcion;
    double valor;
    Conver * pKonv = new Conver;

    //  programa
    while ( true ) {
        system( "cls" );
        opcion = menu( info );

        if ( Fin == opcion ) {
            delete pKonv;
            return EXIT_SUCCESS;
        } // if

        std::cout << "Unidades a convertir: ";
        std::cin >> valor;

        pKonv->factorizar( opcion, valor );
        pKonv->mostrar();
        std::cout << "\n\n";
```

```
        system( "pause" );
    } // while
}
```

Una posible salida

```
Problema 9: calculadora de conversión.

1) Galón USA(gal) y Litro(l)
2) Pie cúbico(ft3) y metro cúbico(m3)
3) Libra(lb) y Kilogramo(kg)
4) Acre (ac) y Hectárea(ha)
5) Milla(mi) y Kilómetro(km)
6) Grados Fahrenheit(F) y Celsius(C)
7) Radián(rad) y grado(grado)
0) Terminar

Entre una opción: 7
Unidades a convertir: 39

 39.0000 rad   -->  74.5354 grado
 39.0000 grado -->   0.6807 rad

Presione una tecla para continuar . . .
```

La eficiencia es del 32%

Conjuntos y bolsas

Son contenedores asociativos. La fortaleza de estos dos contenedores estriba en el acceso sumamente rápido a sus elementos, asociados con la clave que los identifica. Implementados casi siempre como un tipo de árbol balanceado, aplican un algoritmo especializado para disponer dónde almacenar los elementos entrantes, mostrarlos o eliminarlos, por lo que el usuario no controla estas acciones (Josuttis, p. Cap. 6.5.1).

Un **`set`** (también llamado conjunto) es un contenedor asociativo cuyos dos elementos (llamados **`key`** y **`value`**) son únicos y por diseño están en orden ascendente, aunque se pueden construir en orden descendente usando el functor **`greater`**. Un **`multiset`** (también llamado bolsa) es un **`set`** que permite duplicar el **`value`**. Para usar el conjunto o la bolsa se debe incluir la clase **`<set>`**.

Tabla 23. (simplificada) Interfaz de las clases set y multiset

Constructores	Resultados
`set<T> s`	Crea un **`set`** vacío , del tipo **`T`**
`set<T> s(s1)`	Crea en **`s`** una copia de **`s1`**
`multiset<T> ms`	Crea un **`multiset ms`** vacío
`multiset<T> ms(ms1)`	Crea en **`ms`** una copia de **`ms1`**
Métodos	**Resultados**
`n = s.size()`	Devuelve la cantidad de elementos presentes
`s.insert([p,] val)`	Inserta una copia del valor **`val`** y devuelve el nuevo valor (**`p`** es una sugerencia de dónde debe comenzar a buscar la inserción.)
`s.insert(K.ini, K.fin)`	Inserta una copia de todos los elementos del contenedor **`K`**, en el rango **`[ini:fin)`**

`s.erase(p)`	Borra el valor **`val`** que está en la posición **`pos`**
`n = s.erase(val)`	Borra los elementos con el valor **`val`** y devuelve la cantidad de elementos borrados
`s.erase(K.ini, K.fin)`	Borra todos los elementos del contenedor **`K`**, en el rango **`[ini:fin)`**
`s.clear()`	Vacía el contenedor
`b = s.empty()`	Devuelve **`true`** si el **`conjunto/bolsa`** está vacío
`n = s.count(v)`	Devuelve la cantidad de elementos con el valor **`val`**
`i = s.find(val)`	Devuelve la posición del primer elemento con el valor **`val`**. Si no está, devuelve el repetidor **`s.end()`**
`s == s1`	Devuelve **`true`** si ambos son iguales
`s != s1` `s not_eq s1`	Devuelve **`true`** si ambos son distintos
`s = s1`	Sobrescribe los elementos de **`s1`** en **`s`**
`s.swap(s1)`	Intercambia los elementos de **`s`** con los de **`s1`**
Iteradores	**Resultados**
`set<T>::iterator i` `set<T>::const_iterator i`	Un repetidor constante o no.
`s.begin()`	El inicio del set.
`s.end()`	El final del set.
`multiset<T>::iterator i` `multiset<T>::const_iterator i`	Un repetidor constante o no.
`ms.begin()`	El inicio del multiset.
`ms.end()`	El final del multiset.

Notas:

- Los repetidores son pares de valores homólogos a la declaración del contenedor.
- Otras comparaciones que las dos mostradas producen respuestas sin significación.

Álgebra de conjuntos

En matemáticas, un conjunto o **`set`** es una colección de objetos, considerando un objeto "per se" como algo que comparte al menos una cualidad en común con los demás. Por extensión y con el mismo significado se utiliza el término en otros campos, como son la arqueología, las ciencias de la computación, la ciencias de lo jurídico, la construcción de chips de computadores, las artes musicales, etc. (Wikipedia en español)

La teoría de conjuntos es una rama de las matemáticas que estudia las propiedades y relaciones de los conjuntos (sets), herramienta básica en la formulación de cualquier teoría en ese campo. Los conjuntos numéricos usuales son el de los números naturales $\mathbb{N}$, el de los números enteros $\mathbb{Z}$, el de los números racionales $\mathbb{Q}$, el de los números reales $\mathbb{R}$ y el de los números complejos $\mathbb{C}$, y cada uno de ellos, a partir de los naturales, es un subconjunto del que le sigue.

El álgebra de conjuntos presenta unas operaciones, similares a las operaciones aritméticas, que permiten manipular los conjuntos y sus elementos.

Funciones STL del álgebra de conjuntos

La STL provee seis algoritmos generales que calculan la mezcla, unión, intersección, pertenencia, diferencia y diferencia simétrica de dos conjuntos, pero pueden ser aplicados a cualquier par de contenedores homólogos (del mismo tipo) que estén ordenados. Todos menos **`includes`**, toman cinco parámetros: los cuatro primeros son repetidores que definen sendos rangos (el primer par define el rango de aplicación del primer conjunto y el segundo par, el del segundo conjunto); el último parámetro es un insertor. Pero **`includes`**, sólo toma los cuatro primeros y no lleva insertor.

Tabla 24. Algoritmos STL del álgebra de conjuntos

`b = merge(K1.begin(), K1.end(), K2.begin(), K2.end(), K3.begin())`: retorna si `K1` es un subconjunto de `K2`. Si todos los elementos son iguales, es un subconjunto propio.
`set_union(K1.begin(), K1.end(), K2.begin(), K2.end(), K3.begin())`: en `K3` está la unión de `K1` con `K2`.
`set_intersection(K1.begin(), K1.end(), K2.begin(), K2.end(), K3.begin())`: en `K3` está la intersección de `K1` con `K2`.
`set_difference(K1.begin(), K1.end(), K2.begin(), K2.end(), K3.begin())`: ambos están ordenados con un mismo criterio y `K3` posee su diferencia.
`set_symmetric_difference(K1.begin(), K1.end(), K2.begin(), K2.end(), K3.begin())`: mezcla ambos conjuntos y en `K3` están los elementos comunes que no están en ambos.
`b = includes(K1.begin(), K1.end(), K2.begin(), K2.end())`: Verdadero si todos los elementos de `K1` están en `K2`.

Notas:

- Todas las formas de estos algoritmos pueden sustituir el último término por un predicado binario, o por un insertor.
- Todos los elementos en el conjunto-destino están ordenados con un mismo criterio.
- El programador garantiza que los rangos de trabajo estén ordenados por el mismo criterio.
- El rango de B no solapará al de A.
- Los rangos de partida no se modifican.

Insertores

Los insertores son punteros adaptados a partir de repetidores para insertar valores en la secuencia, ya que estos últimos, los sobrescriben. Usualmente son accionados por algoritmos. La biblioteca estándar presenta tres clases de repetidores de inserción, o insertores: los que lo hacen por la cola (*back inserters* en inglés), los que lo hacen por la cabeza (*front inserters* en inglés) y los que lo hacen en cualquier posición (*inserters* en inglés.) Todos son repetidores de salida y como cada uno de ellos utiliza su método propio para insertar, cada uno de ellos debe ser inicializado junto con su contenedor.

Tabla 25. Insertores

N	Clase	Contenedor adecuado
1	`back_insert_iterator`	`vector, string, deque, list`
2	`front_insert_iterator`	`deque, list`
3	`insert_iterator`	Todos, menos los seudocontenedores

Como en este caso se usa el contenedor `set`, la llamada de inicialización adecuada sería:

```
std::set<int>C;
std::insert_iterator<std::set<int>>listar(C, C.begin());
```

donde `listar` es el nombre del insertor, `C` el contenedor donde serán insertados los valores y `C.begin()` es el punto de entrada de los mismos.

Programa 10: álgebra de conjuntos

Dados dos conjuntos de enteros A y B, mostrar las operaciones para conjuntos que suministra la STL. Los valores de ambos conjuntos pudieran haber sido preguntados al usuario, pero se decidió que fueran generados aleatoriamente, pues EODA, la esencia del programa se capta mejor así.

Programa

Las plantillas generadoras de valores aleatorios no funcionan con el **set/multiset** porque sus iteradores son incompatibles con los de los contenedores no asociativos, ya que están optimizados para recorrer un árbol. Entonces antes que manipular directamente la generación, se utilizaron vectores auxiliares y se pasaron sus valores para los respectivos conjuntos.

```
#include "../deposito/print.h"
#include "../deposito/rnd.h"
#include <iostream>
#include <cstdlib>
#include <string>
#include <set>

//typedef std::string Str;
//typedef std::multiset<Str> mmSet;

const int LIM_SUP = 50; // número mayor
const int LIM_INF = 1;  // número menor
const int TOP_LIM = 10; // set mayor
typedef std::multiset<int> mmSet;

int main() {
    using std::cout;
    cout << "Programa 10: \240lgebra de conjuntos\n\n";
    srand( time( nullptr ) );           // una semilla aleatoria
    std::array<int, TOP_LIM>aryA, aryB; // un arreglo auxiliar
    mmSet C;                            // la bolsa
    std::insert_iterator<mmSet>insertar( C, C.begin() ); // el insertor

    //  los conjuntos
/*
    mmSet A;
    A.insert("Platon");
    A.insert("Arquimedes");
    A.insert("Zorba");
    A.insert("Pitagoras");
    LISTA(A, "Conjunto A { "); cout << "}\n";

    mmSet B;
    B.insert("Manuel");
    B.insert("Zorba");
    B.insert("Leonela");
    LISTA(B, "Conjunto B { "); cout << "}\n\n";
*/
    RELLENA_I( aryA, LIM_INF, LIM_SUP );
    RELLENA_I( aryB, LIM_INF, LIM_SUP );
    std::set<int> A( begin( aryA ), end( aryA ) );
    std::set<int> B( begin( aryB ), end( aryB ) - 6 );
    LISTA( A, "Conjunto A { " ); cout << "}\n";
    LISTA( B, "Conjunto B { " ); cout << "}\n\n";

    //  las respuestas:
    //  mezcla
    merge(A.begin(), A.end(), B.begin(), B.end(), insertar);
    LISTA(C, "\tMezcla  { "); cout << "}\n";
    //  unión
    C.clear();
    set_union(A.begin(), A.end(), B.begin(), B.end(), insertar);
    LISTA(C, "\v Uni\242n  { "); cout << "}\n";
```

```
    //  intersección y pertenencia de B respecto a A
    C.clear();
    set_intersection(A.begin(), A.end(), B.begin(), B.end(), insertar);

    if ( C.empty() ) {
        cout << "  Intersecci\242n  Conjunto vac\241o.\n";
        cout << "  Pertenencia\tB no es subconjunto de A\n";
    } else {
        LISTA(C, "  Intersecci\242n  { "); cout << "}\n";
        includes(A.begin(), A.end(), B.begin(), B.end())
            ? cout << "   Pertenencia\tB es subconjunto propio de A\n"
            : cout << "   Pertenencia\tB es subconjunto de A\n";
    }

    //  igualdad
    A == B
        ? cout << "      Igualdad\tSon iguales\n."
        : cout << "      Igualdad\tNo son iguales.\n";
    //  diferencia A\B
    C.clear();
    set_difference(A.begin(), A.end(), B.begin(), B.end(), insertar);

    if ( 0 == C.size() )
        cout << "Diferencia A\\B: conjunto vac\241o.\n";
    else {
        LISTA(C, "Diferencia A\\B  { ");
        cout << "}\n";
    }

    //  diferencia B\A
    C.clear();
    set_difference(B.begin(), B.end(), A.begin(), A.end(), insertar);

    if ( 0 == C.size() )
        cout << "Diferencia B\\A: conjunto vac\241o.\n";
    else {
        LISTA(C, "Diferencia B\\A  { ");
        cout << "}\n";
    }

    //  diferencia simétrica
    C.clear();
    set_symmetric_difference(A.begin(), A.end(), B.begin(), B.end(), insertar);

    if ( 0 == C.size() )
        cout << "Dif. sim\202trica: conjunto vac\241o.\n";
    else {
        LISTA(C, "Dif. sim\202trica  { ");
        cout << "}\n\n";
    }

    // fin
    system("pause");
    return EXIT_SUCCESS;
}
```

Una posible salida

Las salidas dependerán de las combinaciones **set/multiset** usadas. Por ejemplo, todo a **multiset** da una salida y con los mismos datos todo a **set** da otra.

```
Programa 10: álgebra de conjuntos

Conjunto A { 6, 7, 10, 19, 29, 31, 34, 37, 42 }
Conjunto B { 10, 19, 31, 37, 42 }

       Mezcla  { 6, 7, 10, 10, 19, 19, 29, 31, 31, 34, 37, 37, 42, 42 }
        Unión  { 6, 7, 10, 19, 29, 31, 34, 37, 42 }
 Intersección  { 10, 19, 31, 37, 42 }
  Pertenencia  B es subconjunto propio de A
     Igualdad  No son iguales.
Diferencia A\B  { 6, 7, 29, 34 }
Diferencia B\A: conjunto vacío.
Dif. simétrica  { 6, 7, 29, 34 }
```

Aquí la eficiencia es del 48%

Los contenedores no "saben" qué contienen, sólo lo almacenan. Este programa se adaptó a cadenas C++, se le entraron nuevos datos y se ejecutó sin más cambios. Comentando los contenedores numéricos y descomentando los de cadenas se obtiene esta otra salida:

```
Programa 10: álgebra de conjuntos

Conjunto A { Arquimedes, Pitagoras, Platon, Zorba }
Conjunto B { Leonela, Manuel, Zorba }

       Mezcla  { Arquimedes, Leonela, Manuel, Pitagoras, Platon, Zorba, Zorba }
        Unión  { Arquimedes, Leonela, Manuel, Pitagoras, Platon, Zorba }
 Intersección  { Zorba }
  Pertenencia  B es subconjunto de A
     Igualdad  No son iguales.
Diferencia A\B  { Arquimedes, Pitagoras, Platon }
Diferencia B\A  { Leonela, Manuel }
Dif. simétrica  { Arquimedes, Leonela, Manuel, Pitagoras, Platon }
```

Números complejos

El concepto de número complejo es una extensión del de los números reales y puede representarse como la suma simbólica de un número real y un número imaginario, múltiplo real de la unidad imaginaria que es indicada con la letra i. (Wikipedia en español)

Otro componente numérico que ofrece la STL es la clase **<complex>**, que representa los números complejos y define los métodos que operan sobre ellos. Un número complejo z en forma algebraica se escribe como un par ordenado de números reales: $z = (a, b)$ y en forma binomial como una suma simbólica: $z = a + bi$ — Ambas notaciones representan al mismo número complejo en formatos diferentes

Tabla 26. (simplificada) Interfaz de la clase complex

Constructores	**Resultado**
complex<T>z	Crea un complejo **z** de tipo **T** con valores **(0,0i)**
complex<T>z(r, i)	Crea un complejo **z** de tipo **T** con valores **r** (real), e **i** (imaginario)
complex<T>z2(z1)	Un complejo **z2** de tipo **T**, a partir del complejo **z1**
complex<T>p(polar(ro))	Un complejo **p** en base a un valor polar real **ro** y ángulo de **0** radianes
complex<T>p(polar(ro, sita))	Un complejo **p** en base a un valor polar real **ro** y ángulo de **sita** radianes
Operador de asignación	**Resultado**
z = z1	**z** recibe los valores de **z1**

`v = real(z)`	**v** recibe al componente real de **z**
`v = imag(z)`	**v** recibe al componente imaginario de **z**
Operadores de comparación	**Resultado**
`b = (z1 == z2)`	`true` si `z1` y `z2` son iguales
`b = (z1 ¡= z2)`	`true` si `z1` y `z2` son diferentes
`b = (z1 not_eq z2)`	
Operadores aritméticos	**Resultado**
`z += escalar`	Número complejo a la izquierda y escalar a la derecha
`z + escalar`	Sólo afecta la parte real
`z - escalar`	
`escalar * z`	Afecta ambos términos
`z * escalar`	
`z / escalar`	
`escalar / z`	
`z = z1 + z2`	Números complejos en ambos lados
`z1 += z2`	
`z1 = z1 + z2`	
Funciones	**Resultado**
`a = abs(z)`	Valor de la magnitud polar de **z**: $\sqrt{(\mathbf{real}^2 + \mathbf{imag}^2)}$
`a = arg(z)`	Valor del ángulo de fase **f** de **z**
`a = norm(z)`	Valor de la norma de **z**: $(\mathbf{real}^2 + \mathbf{imag}^2)$
`z2 = conj(z1)`	Complejo conjugado de **z1**
`z1 = pow(z, exp)`	Complejo de $\mathbf{z}^{\mathbf{exp}}$
`z1 = pow(exp, z)`	Complejo de $\mathbf{z}^{\mathbf{exp}}$
`z1 = sqrt(z)`	Complejo de $\sqrt{\mathbf{z}}$
`z1 = exp(z)`	Complejo de $\mathbf{e}^{\mathbf{z}}$
`z1 = log(z)`	Complejo del logaritmo natural de **z**
`z1 = log10(z)`	Complejo del logaritmo base 10 de **z**
`z1 = sin(z)`	Complejo del seno de **z**
`z1 = cos(z)`	Complejo del coseno de **z**
`z1 = tan(z)`	Complejo de la tangente de **z**

Notas:

- Menos la división módulo (**&**), todas las operaciones aritméticas del tipo **op=** (ej. **+=**) están definidas.
- La función auxiliar **`polar()`** permite crear un número complejo desde su constructor, a partir de la magnitud de su radio-vector ($\rho = rho$) y su ángulo de fase ($\theta, = sita$) en radianes, pero NO desde el operador de asignación. Esto significa que, por ejemplo:

```
complex<float>c2(polar(4.2,0.75));  // OK con el constructor
complex<float>c2 = polar(4.2,0.75); // ERROR con una asignación
```

  La segunda forma produce error de compilación

```
...error: conversion from 'complex<double>' to non-scalar type complex<float>' requested
```

- Cualquier comparación que no sean una de las mostradas en la tabla carece de sentido.
- Notar que la clase no posee un método para escribir el formato binomial; hay que codificarlo.

Programa 11: números complejos

En el programa 23 de (Introducción al C++, tomo 1, pág. 158) se creaba una ADT para números complejos hecha manualmente. Aquí se resuelven y amplían sus proposiciones usando la clase **<complex>**, lo cual posibilita eliminar la compilación separada de la clase compleja que se hizo manualmente.

Programa

```
#include <iostream>
#include <cstdlib>
#include <complex>
#include <iomanip>
#include <cmath>
#include <ios>

const double RAD2GRAD = 45.0 / atan( 1.0 );

//  Escribe un número complejo en formato z = a+/-bi
//  input: un número complejo
//  output: salida formateada a consola
void fmtSuma( std::string ID, const std::complex<float> &z );

//  Escribe un número complejo en formato (M,A)
//  M: módulo del complejo; A: angulo de frase en grados
//  input: un número complejo
//  output: salida formateada a consola
void fmtPolar( std::string ID, const std::complex<float> &z );

int main() {
    using std::cout;
    using std::polar;
    cout << "Programa 11: aritm\240tica compleja.\n\n";
    // los números
    std::complex<float> z1, z2;
    float esc;
    //  entrada
    cout << "Entrar datos en formato (a,b):\n";
    cout << "Complejo z1 = "; std::cin >> z1;
    cout << "Complejo z2 = "; std::cin >> z2;
    cout << "Escalar: ";      std::cin >> esc;
    //  coordenadas
    cout << "\nEn formato z = a+bi\n";
    fmtSuma( "z1 = ", z1 );
    fmtSuma( "z2 = ", z2 );
    cout << "\nEn formato z = (Mod,Grad)\n";
    fmtPolar( "z1 = ", z1 );
    fmtPolar( "z2 = ", z2 );
    //  formato general
    cout << std::setprecision( 2 ) << std::fixed;
    //  conjugados
    cout << "\nConjugados:";
    cout << "\n~z1 = " << conj( z1 );
    cout << "\n~z2 = " << conj( z2 );
    //  igualdad
    cout << "\n\nIgualdad:\n";
    z1 == z2
        ? cout << "z1 y z2 son iguales.\n"
        : cout << "z1 y z2 no son iguales.\n";
    // algunas operaciones aritméticas
```

```
    cout << "\nOperaciones aritm\202ticas:\n";
    cout << "(z1+z2) = " << z1+z2 << '\n';
    cout << "(z1-z2) = " << z1-z2 << '\n';
    cout << "(z1*z2) = " << z1*z2 << '\n';
    cout << "(z1/z2) = " << z1/z2 << '\n';
    //  con escalares
    cout << "\nOperaciones con escalares:\n";
    cout << "e+z1 = " << esc+z1 << '\n';
    cout << "e-z1 = " << esc-z1 << '\n';
    cout << "z1-e = " << z1-esc << '\n';
    cout << "e*z1 = " << esc*z1 << '\n';
    cout << "z1*e = " << z1*esc << '\n';
    cout << "z1/e = " << z1/esc << '\n';
    cout << "e/z1 = " << esc/z1 << '\n';
    // aplicación de funciones
    cout << "\nOperaci\242n con tres funciones:";
    cout << "\nCuadrado: z1" << pow( z1, 2.0 );
    cout << "\nRa\241z: z1"  << sqrt( z1 );
    cout << "\nSeno: z1"     << sin( z1 );
    // módulos
    cout << std::setprecision( 2 ) << std::fixed
         << "\n\nLos m\242dulos son:"
         << "\n|z1| = " << abs(z1)
         << "\n|z2| = " << abs(z2) << "\n\n";
    //  fin
    system( "pause" );
    return EXIT_SUCCESS;
}

// muestra un número complejo en formato suma
void  fmtSuma( std::string id, const std::complex<float> &z ) {
    std::ios::fmtflags oldFlags = cout.flags();
    cout << id << std::setprecision( 2 ) << std::fixed << z.real()
         << std::showpos << z.imag() << "i\n" << std::setprecision( 6 );
    cout.flags( oldFlags );
}

// muestra un número complejo en formato polar
void fmtPolar( std::string id, const std::complex<float> &z ) {
    double sita = arg(z) * RAD2GRAD;
    std::ios::fmtflags oldFlags = cout.flags();
    cout << id << std::setprecision( 2 ) << std::fixed
         << "(" << abs(z) << "," << sita << ")\n"
         << std::setprecision( 6 );
    cout.flags( oldFlags );
}
```

Salida

```
Programa 11: aritmática compleja.

Entrar datos en formato (a,b):
Complejo z1 = (9.8,6.5)
Complejo z2 = (5.4,2.1)
Escalar: 1.33

En formato z = a+bi
z1 = 9.80+6.50i
z2 = 5.40+2.10i

En formato z = (Mod,Grad)
z1 = (11.76,33.55)
z2 = (5.79,21.25)

Conjugados:
~z1 = (9.80,-6.50)
~z2 = (5.40,-2.10)

Igualdad:
z1 y z2 no son iguales.

Operaciones aritméticas:
(z1+z2) = (15.20,8.60)
(z1-z2) = (4.40,4.40)
(z1*z2) = (39.27,55.68)
(z1/z2) = (1.98,0.43)

Operaciones con escalares:
e+z1 = (11.13,6.50)
e-z1 = (-8.47,-6.50)
z1-e = (8.47,6.50)
e*z1 = (13.03,8.65)
z1*e = (13.03,8.65)
z1/e = (7.37,4.89)
e/z1 = (0.09,-0.06)

Operación con tres funciones:
Cuadrado: z1(53.79,127.40)
Raíz: z1(3.28,0.99)
Seno: z1(-121.88,-309.43)
```

La eficiencia es del 24%

Barajado

El algoritmo Fisher-Yates, es un algoritmo de permutaciones que técnicamente encaja en la categoría de los algoritmos de ordenamiento, pero su objetivo es el opuesto: desordenar los ítems que contiene. Es el que se usa típicamente para barajar en los juegos de azar o sacar, en las rifas, los premios desde un sombrero. Editado de (Wikipedia en español)

En el problema 22 de (Introducción al C++, tomo 1, pág. 124) se aplicó directamente una forma básica del algoritmo de Fischer-Yates[17]. Acá se tomaron las facilidades dadas por la STL para generar barajados ¿Resultado? Un código más fácil de escribir, más fiable y más robusto.

Programa 12: barajado aleatorio

Dice el problema: la baraja española clásica consiste en un mazo de 40 naipes, divididos en cuatro palos que son: oros, copas, espadas y bastos. Los números son As, 2, 3, 4, 5, 6, 7, Sota, Caballo y Rey. Se le pide barajar el mazo. Se conoce que los principales juegos de azar con esta baraja se hacen entre 4 jugadores. Al sustituir el arreglo por un **`array`** y utilizar el algoritmo **`random_shuffle`**, se robustece, simplifica y sintetiza el código, uniendo el llenado y el barajado.

Definiciones

- **`cartas.h`**

```
#ifndef CARTAS_H_INCLUDED
#define CARTAS_H_INCLUDED

const int MAZO  = 40; // un mazo trae 40 cartas
const int PALOS =  4; // son 4 palos en la baraja
const int CAMPO = 18; // ancho del campo de salida

#include <string>
typedef std::string Str;

#include <array>
typedef std::array<Str, MAZO> Mazo; // la baraja

//  crea un mazo de cartas y lo baraja
//  input: el array de las cartas
//  output: el mazo está barajado
void cargar( Mazo & cartas );

//  reparte la baraja entre 4 jugadores
//  input: el array de cartas y el de posición
//  output: el mazo está repartido
void repartir( Mazo & cartas );

#endif // CARTAS_H_INCLUDED
```

Declaraciones

- **`cartas.cpp`**: sustituye el arreglo original por un **`array STL`** y los **`switchs`** por un conjunto de instrucciones **`if`**, mucho más compactas. Usa el algoritmo **`random_shuffle()`** directamente sobre la baraja y se le debe alimentar una semilla nueva, para obtener nuevos barajados por ejecución. Una variable estática en la función **`mostrar`** existe por la duración del programa y mantiene su valor entre llamadas, posibilitando una salida visualmente placentera. Cuando **`mostrar`** es llamado por primera vez para poner la primera baraja, **`static int j`** se pone a cero y al terminar se incrementa a uno. Cuando es llamada por segunda vez, ya está en uno y al terminar se incrementa a dos. Y así sucesivamente hasta terminar el reparto. Al llegar al final se interrumpe el ciclo evitando la última nueva línea.

```
#include "cartas.h"
```

[17] Ver el artículo *Algoritmo Fisher-Yates* en (Wikipedia en español).

```
//  las figuras de la baraja
enum { Oros, Copas, Espadas, Bastos };

Str ponerPalo( int palo ) {
    if ( Oros    == palo ) return "oros";
    if ( Copas   == palo ) return "copas";
    if ( Espadas == palo ) return "espadas";
    return "bastos";
}

//  los números de la baraja
enum { As, Dos, Tres, Cuatro, Cinco, Seis, Siete, Sota, Caballo, Rey };

Str ponerValor( int valor ) {
    if ( As      == valor ) return "As de ";
    if ( Dos     == valor ) return "Dos de ";
    if ( Tres    == valor ) return "Tres de ";
    if ( Cuatro  == valor ) return "Cuatro de ";
    if ( Cinco   == valor ) return "Cinco de ";
    if ( Seis    == valor ) return "Seis de ";
    if ( Siete   == valor ) return "Siete de ";
    if ( Sota    == valor ) return "Sota de ";
    if ( Caballo == valor ) return "Caballo de ";
    return "Rey de ";
}

#include <algorithm>
#include <ctime>

//  carga y baraja un mazo de cartas
void cargar( Mazo & cartas ) {
    int i = 0;

    do {
        for ( int fig = Oros; fig <= Bastos; ++fig ) {
            for ( int val = As; val <= Rey; ++val ) {
                cartas[i] = ( ponerValor( val ) + ponerPalo( fig ) );
                ++i;
            } // for As hasta Rey
        } // for Oros hasta Bastos
    } while ( i < MAZO ); // do-while

    srand( time( nullptr ) ); // garantiza distintas secuencias por ejecución
    std::random_shuffle( cartas.begin(), cartas.end() ); // baraja el arreglo
}

#include <iostream>
#include <iomanip>

//  función auxiliar: pone una línea divisoria
void ponerLinea() {
    // CAMPO x PALOS + un blanco entre cada 2 jugadores
    int ancho = CAMPO * PALOS + 3;
    std::cout << std::left
    << std::setw( ancho ) << std::setfill( '-' ) << '\n'
    << std::setfill( ' ' ) << '\n';
}

//  muestra las barajas desde for_each
void mostrar( const Str & carta ) {
    static int j = 0; // lleva la cuenta entre llamadas
```

```
        std::cout << std::setw( CAMPO ) << carta;                    // pone la carta
        if ( j == MAZO - 1 ) return;                                 // ¿última línea?
        0 == ( j + 1 ) % PALOS ? std::cout << '\n' : std::cout << ' '; // reparte
        ++j;
    }

    //  reparte entre 4 jugadores
    void repartir( Mazo & cartas ) {
        //  encabezado para 4 jugadores
        for ( int i = 0; i < 4; ++i )
            std::cout << "Jugador " << i + 1 << std::setw( CAMPO );

        //  tabla
        ponerLinea();
        for_each( cartas.begin(), cartas.end(), mostrar );
        ponerLinea();
    }
```

Programa

```
#include "cartas.h"
#include <iostream>
#include <cstdlib>

int main() {
    std::cout << "Programa 12: un barajado de cartas espa\244olas.\n\n";
    Mazo cartas;
    cargar( cartas );
    repartir( cartas );
    system( "pause" );
    return EXIT_SUCCESS;
}
```

Una posible salida

```
Programa 12: un barajado de cartas españolas.

Jugador 1          Jugador 2          Jugador 3          Jugador 4
-------------------------------------------------------------------------
Caballo de oros    Sota de oros       Siete de oros      Dos de bastos
Rey de bastos      Rey de copas       Cuatro de bastos   Cuatro de copas
Sota de espadas    Tres de copas      Siete de espadas   Cinco de bastos
Sota de copas      Cinco de oros      Tres de bastos     As de bastos
Caballo de espadas Dos de oros        Siete de bastos    Tres de espadas
Sota de bastos     Cinco de copas     Seis de oros       Dos de espadas
Siete de copas     Cuatro de oros     Caballo de copas   Seis de copas
Dos de copas       As de copas        Rey de oros        As de espadas
Rey de espadas     Cuatro de espadas  As de oros         Tres de oros
Seis de espadas    Cinco de espadas   Caballo de bastos  Seis de bastos
-------------------------------------------------------------------------
```

La eficiencia es del 38%

Ordenamiento

La ordenación y búsqueda son operaciones fundamentales en informática. La ordenación se refiere a la operación de organizar datos…creciente o decrecientemente para números o alfabéticamente para caracteres. (Lipschutz, pág. 361)

El ordenamiento rápido o Quicksort es un algoritmo que tiene varias formas de ser codificado. Fue inventado por el británico Anthony Hoare en 1960, y está basado en la técnica recursiva de "divide-y-vencerás", pero además es el algoritmo interno más rápido conocido hasta hoy.

Tabla 27. Prototipo de qsort()

Prototipo: `void qsort(void *ary, size_v cant, size_v tam, int *comparar(const void *p,const void *q));`	
Parámetro	**Significado**
`void *ary`	Puntero a un arreglo unidimensional de valores.
`size_v cant`	Cantidad de valores del arreglo.
`size_v tam`	Tamaño en bytes de cada elemento del arreglo.
`int *comparar(const void *p const void *q)`	Función de comparación codificada manualmente, que toma dos punteros genéricos a sendos elementos del arreglo.

El algoritmo ordena un arreglo apuntado por **`ary`**, de **`cant`** elementos residentes en memoria, de tamaño **`tam`**. El programador codifica la función de comparación generalizada.

Programa 13: comparativa de ordenamientos

Idealmente, realizaríamos las pruebas cierta cantidad de veces y promediaríamos los tiempos, para obtener un resultado más preciso. (Halterman, pág. 366)

Quicksort es un algoritmo recursivo que fija un valor como pivote, dividiendo la secuencia en dos disjuntas, una con valores menores que el pivote y otra con valores mayores, y va repitiendo el proceso sobre ambas partes hasta que quedan n listas de un solo elemento, todas ordenadas. Desde su publicación en 1960 ha tenido múltiples implementaciones, unas mejores que otras, mayormente basadas en la selección del primer pivote y su posicionamiento en la lista. Por ejemplo, (Williams) cita los siguientes en orden de importancia: la versión Fat-Pivot (para él, la mejor), la de R. Sedgewick, la original de A. Hoare, la realización bajo ANSI C y las versiones propietarias de Microsoft™ y Embarcadero™. Aquí se usa una versión C++ implementado en la suite GNU, para las series 4.5/4.6 de compiladores C/C++.

Toda implementación de Quicksort bajo C++ se llama **`qsort()`**

En el programa 26 de (Introducción al C++, tomo 1, pág. 182) se pedía probar la función **`qsort()`** de la biblioteca estándar creando un arreglo de 150,000 valores enteros generados al azar y marcarle el tiempo tardado en ordenarlos. Escalando este problema, ahora se trabajarán medio millón de valores enteros aleatorios por pase y se comparará el rendimiento promedio de **`qsort()`** por cinco pases contra los métodos básicos de ordenamiento total provistos por la STL.

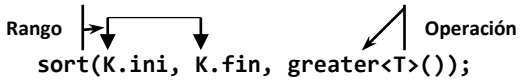
Nota: actualmente todos los SO populares son multitareas, lo cual significa que el tiempo de ejecución lo distribuyen otorgando una parte del mismo a cada tarea bajo ejecución. Por eso el SO constantemente está interrumpiendo una tarea para dar una porción de tiempo a las demás. Aunque la mayoría de las veces estos cambios son imperceptibles al usuario, pueden influir al medir resultados cronometrados en un programa de larga duración.

Tomando a **`qsort()`** como patrón, se comparan los siguientes algoritmos:

- **`sort(K.ini, K.fin)`**. Ordena todos los elementos de cualquier contenedor no-asociativo, pero el ordenamiento no es estable, significando que no se preserva el orden original de los elementos de igual valor.

- **`stable_sort(K.ini, K.fin)`**. Ordena todos los elementos de cualquier contenedor no-asociativo y el ordenamiento es estable, significando que se preserva el orden original de los elementos de igual valor.
- **`partial_sort(K.ini, K.fin_orden, K.fin)`**. En el rango **`[ini:fin)`** ordena los elementos de cualquier contenedor no-asociativo desde **`K.ini`** hasta **`K.fin_orden`**, deteniéndose allí; el resto queda como estaba. Si **`K.ini`** apunta al primer elemento y **`K.fin_orden`** es igual al último, ordena todo el contenedor, y así se usa aquí.
- **`make_heap(K.ini, K.fin)`**. Pone todos los elementos de cualquier contenedor no-asociativo en el rango **`[ini:fin]`** en un bulto[18] directamente puesto en memoria.
- **`sort_heap(K.ini, K.fin)`**. Devuelve los valores ordenados en el rango **`[ini:fin]`** al contenedor. El ordenamiento no es estable. Este algoritmo se usa conjuntamente con el anterior.
- **`multiset.insert(K.ini, K.fin)`**. El método inserta en una bolsa los valores dados en el rango **`[ini:fin]`**, y en la medida en que los adiciona, los va ordenado. Al final la bolsa está ordenada.
- **`list.sort()`**. El método ordena todos los valores de la lista. El ordenamiento es estable.

Todos estos algoritmos ordenan de menor a mayor por omisión y así se usarán, pero todos tienen una segunda versión que permite el ordenamiento contrario supliéndoles un último parámetro. Por ejemplo:



ordena de mayor a menor al contenedor **`K`** del tipo **`T`**

Programa

```
#include "../deposito/rnd.h"
#include "auxiliar.h"
#include <iostream>
#include <iomanip>
#include <algorithm>

const int PASE = 5;

// función de comparación de enteros para qsort()
int comparar( const void * m, const void * n ) {
    int p = *static_cast<const int *>( m );
    int q = *static_cast<const int *>( n );
    return p - q;
}

int main() {
    int m = 1, M = N * 0.5; // límites de los números generados

    std::cout <<
    "Programa 13: comparativa de ordenamiento\n\n"
    "Comparaci\242n entre qsort() y las formas de ordenamiento\n"
    "total de la STL, haciendo " << PASE << " pases sobre " << N <<
    " enteros,\nentre " << m << " y " << M <<  ", generados al azar.\n\n"
    "Tenga un poco de paciencia. Trabajando...\n\n";

    // pedido de memoria
    vInt vc1( N ), vc2( N ); // dos vectores
    lInt lst;                // una lista
    sInt mms;                // una bolsa
    aInt ary;                // un arreglo estático STL
    int * pAry = new int[N]; // un arreglo en la tienda libre
```

[18] Un **heap** (bulto) en este contexto es una estructura de datos que se implementa en C++ como un tipo de árbol binario.

```
    // las marcas del tiempo y los totales
    clock_t ini, fin, totalIni, totalFin;
    double total_R  = 0, total_qs = 0,
           total_st = 0, total_ss = 0, total_ps = 0,
           total_ls = 0, total_ms = 0, total_hs = 0,
           total_ar = 0, tiempo_total = 0;

    totalIni = clock();

    for ( int i = 0; i < PASE; ++i ) {
        std::cout << "Pase #" << i + 1 << '\r';

        // el tiempo de llenar nuevos valores
        ini = clock();
        RELLENA_I( vc1, m, M );                        // un juego nuevo de valores
        vc2 = vc1;                                     // se pasan al otro vector
        lst.assign( vc1.begin(), vc1.end() );          // se pasan a la lista
        copy( vc1.begin(), vc1.end(), ary.begin() ); // se pasan al arreglo estático
        copy( vc1.begin(), vc1.end(), pAry );          // se pasan al arreglo dinámico
        fin = clock();
        total_R += difftime( fin, ini ) * 0.001;
        // el tiempo de qsort() sobre un vector en la tienda libre
        ini = clock();
        qsort( pAry, N, sizeof( int ), cmpia );
        fin = clock();
        total_qs += difftime( fin, ini ) * 0.001;
        // el tiempo de sort() sobre un arreglo STL
        ini = clock();
        std::sort( ary.begin(), ary.end() );
        fin = clock();
        total_ar += difftime( fin, ini ) * 0.001;
        // el tiempo de sort() sobre un vector STL
        ini = clock();
        std::sort( vc2.begin(), vc2.end() );
        fin = clock();
        total_st += difftime( fin, ini ) * 0.001;
        vc2 = vc1; // vc2 está azarosa de nuevo
        // el tiempo de stable_sort() sobre un vector STL
        ini = clock();
        stable_sort( vc2.begin(), vc2.end() );
        fin = clock();
        total_ss += difftime( fin, ini ) * 0.001;
        vc2 = vc1; // vc2 está azarosa de nuevo
        // el tiempo de sort_heap()
        ini = clock();
        make_heap( vc2.begin(), vc2.end() ); // convierte la secuencia en árbol
        sort_heap( vc2.begin(), vc2.end() ); // convierte el árbol en secuencia
        fin = clock();
        total_hs += difftime( fin, ini ) * 0.001;
        vc2 = vc1; // vc2 está azarosa de nuevo
        // el tiempo de partial_sort() sobre un vector STL
        ini = clock();
        partial_sort( vc2.begin(), vc2.end(), vc2.end() );
        fin = clock();
        total_ps += difftime( fin, ini ) * 0.001;
        vc2 = vc1; // vc2 está azarosa de nuevo
        // el tiempo de la lista
        ini = clock();
        lst.sort();
        fin = clock();
        total_ls += difftime( fin, ini ) * 0.001;
```

```
        // el tiempo del multiset sobre una lista azarosa
        ini = clock();
        mms.clear();
        mms.insert( vc2.begin(), vc2.end() );
        fin = clock();
        total_ms += difftime( fin, ini ) * 0.001;
    } // for

    totalFin = clock();
    tiempo_total = difftime( totalFin, totalIni ) * 0.001;

    // la salida
    std::cout << std::fixed << std::setprecision( 4 ) <<
    "\n\nTiempo total consumido: " << tiempo_total << " s" <<
    "\nTiempo promedio consumido por:\n"
    "  genera valores = " << total_R  / PASE << " s/pase.\n"
    "  qsort()        = " << total_qs / PASE << " s/pase.\n"
    "  array: sort()  = " << total_ar / PASE << " s/pase.\n"
    "  lista: sort()  = " << total_st / PASE << " s/pase.\n"
    "  stable_sort()  = " << total_ss / PASE << " s/pase.\n"
    "  sort_heap()    = " << total_hs / PASE << " s/pase.\n"
    "  partial_sort() = " << total_ps / PASE << " s/pase.\n"
    "  list.sort()    = " << total_ls / PASE << " s/pase.\n"
    "  multiset       = " << total_ms / PASE << " s/pase.\n\n";

    system( "pause" );
    delete [] pAry;
    return EXIT_SUCCESS;
}
```

Salida

```
Programa 13: comparativa de ordenamiento

Comparación entre qsort() y las formas de ordenamiento
total de la STL, haciendo 5 pases sobre 500000 enteros,
entre 1 y 250000, generados al azar.

Tenga un poco de paciencia. Trabajando...

Pase #5

Tiempo total consumido: 9.4600 s
Tiempo promedio consumido por:
  genera valores = 0.0790 s/pase.
  qsort()        = 0.2022 s/pase.
  array: sort()  = 0.0444 s/pase.
  lista: sort()  = 0.0456 s/pase.
  stable_sort()  = 0.0622 s/pase.
  sort_heap()    = 0.1182 s/pase.
  partial_sort() = 0.1176 s/pase.
  list.sort()    = 0.4642 s/pase.
  multiset       = 0.7546 s/pase.
```

La eficiencia fue del 35%.

Para esta salida, las operaciones tomaron unos 9½ s en total, y el programa lo consumió casi todo en la comparativa, porque la PC estaba dedicada a la tarea. La generación de valores consumió un 4% del tiempo de trabajo. En lo tocante a la eficiencia, dando a **`Quicksort`** el 100%, el algoritmo **`sort`** fue entre un 22 y un 30% más eficiente, el llenado de una bolsa o el ordenamiento parcial fueron alrededor de un 58% más eficientes, mientras que **`qsort`** fue un 43% más eficiente que el método **`sort`** de una lista, y un 27% más que un conjunto múltiple.

Del análisis se desprende que los algoritmos de ordenamiento **`sort()`** y **`stable_sort()`** vencen con ventaja al **`qsort()`** suministrado por esta versión de GCC, mientras que los ordenamientos parcial y por bulto son algo más eficientes. Salen en desventaja de tiempo consumido el **`multiset`** y la función-miembro **`sort`** de la clase **`list`**.

En cada ejecución los valores variarán, pero se mantendrán cercanos a los dados, y cada equipo en particular tendrá su conjunto promedio de valores.

En la vida real, con la actualidad tecnológica de hoy día, absolutamente se preferirá la simpleza frente al desempeño, independientemente del concepto de rapidez: a mayor sencillez, menor probabilidad de deslizar un error lógico (*bug*) en el programa, mayor rapidez al elaborar código y menor gasto de recursos.

Dice (Stroustrup) en una hipotética conversación con un aprendiz de programador:

> *Para un programador novicio, qsort es extraño. ¿Por qué dar el número de elementos? (Porque el arreglo no lo sabe.) ¿Por qué dar el tamaño de un entero? (Porque qsort no sabe que ordena enteros.) ¿Por qué tiene que escribir esa fea función de comparación? (Porque qsort necesita de un puntero a la función, ya que no sabe el tipo de los elementos que está ordenando.) ¿Por qué la función de comparación de qsort toma argumentos const void* en vez de argumentos actuales, como char*? (Porque el qsort puede ordenar basado en valores que no sean de cadena.) ¿Qué es un void* y cuál es el significado de ser const? ("Eh, hmmm, llegaremos a eso más adelante".)*

Menos el uso de un conjunto/bolsa, o el método integrado a las listas, los algoritmos de ordenamiento total aportados por la STL fueron muy superiores a la implementación del **`Quicksort`** suministrada por el sistema, y todos, todos ellos son mucho más fáciles y rápidos de codificar. Indiscutiblemente, C++ no sólo contribuye con un altísimo grado de abstracción, pero con el uso de la STL aporta desempeño con ventajas.

Independientemente de todo, siempre se preferirá la simpleza del código frente al desempeño

Búsqueda

Actualmente existen aplicaciones —mayormente bases de datos— que trabajan con cantidades cuantiosas, verdaderamente masivas de registros (decenas, centenas, millones y millares de millones de ellos) y se requiere de una búsqueda que tarde un tiempo razonable…Para ello se aplican múltiples técnicas, y una bien sencilla y bastante usada es localizar la clave con un algoritmo binario y si está, extraerla entonces con un algoritmo lineal. (Introducción al C++, tomo 1, pág. 184)

Programa 14: búsqueda visual

El programa 27 de (Introducción al C++, tomo 1, pág. 185) dice: Para probar la función **`bsearch()`**, se crea un arreglo de 80 valores enteros entre uno y 100, generados al azar, sobre el cual puede comprobarse visualmente la operación de búsqueda. Una vez decidido que el valor clave está, mostrar sus posiciones.

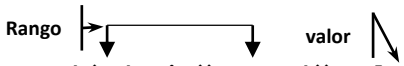
Aquí se aplica la STL para resolver los dos problemas: si la clave está y dónde se ubica.

Definición

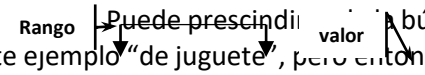
La respuesta consiste en un par de valores: las veces que está y el lugar que ocupa (el elemento de índice cero ocupa el primer lugar, el elemento de índice uno, el segundo, etc. El lugar es el índice más uno.) Se toma un par como dato básico. Los valores a buscar se ponen en un arreglo STL.

```
#ifndef BUSCA_H_INCLUDED
#define BUSCA_H_INCLUDED

const int N = 80;

#include <array>
typedef std::array<int, N> Ary;
typedef std::pair<int, int> Par;

//  busca las posiciones de los valores
//  input: clave y el arreglo
//  output: las posiciones y el lugar de la clave en el arreglo
Par posiciones( int clave, Ary & ary );

#endif // BUSCA_H_INCLUDED
```

Declaración



Para saber si la clave está se usa el algoritmo **binary_search(K.begin(), K.end(), clave)**, que retorna un valor booleano. Si el resultado es false, se devuelve el par **(0,0)**. Si devuelve **true** la cl



Puede prescindi búsqueda binaria, ya que **equal_range** puede hacer todo el trabajo perfectamente en este ejemplo "de juguete", pero entonces la secuencia se visitaría totalmente si la clave no está, o casi en su totalidad para un valor de últimas, y eso traería notable ineficiencia en el proceso de decenas de millones de registros o más.

Para avanzar el índice hasta la primera posición de la clave se usa **equal_range(K.begin(), K.end(), clave)**, que retorna un **pair** que contiene dos iteradores marcando los límites donde aparece la clave. Hay que declarar el par de iteradores de retorno y todo eso se hace en la instrucción comentada como "*los límites donde está la clave*". Entonces se calculan directamente los valores del lugar donde por primera vez aparece la clave y la cantidad de veces que aparece, y se devuelve el par.

```
#include "busca.h"
#include <algorithm>
using std::pair;
using std::binary_search;
using std::equal_range;

//  busca las posiciones de los valores: la estrategia es primero localizar si
//  la clave está; luego localizar dónde está y contar las veces que aparece
Par posiciones( int clave, Ary & ary ) {
    int lugar = 0, veces = 0; // de entrada no está

    if ( binary_search( ary.begin(), ary.end(), clave ) ) { // ¿está?
        // los límites donde está la clave
        pair<Ary::iterator, Ary::iterator>pos = equal_range( ary.begin(), ary.end(), clave );
        // devuelve el LUGAR (uno más que el índice) y las veces
        lugar = pos.first - ary.begin() + 1; // la posición
        veces = pos.second - pos.first;      // las veces
    }

    return std::make_pair( lugar, veces );
}
```

Programa

La respuesta está en el par: cuando el segundo componente vale cero, el valor no está; otrosí el valor está en el primer componente y la cantidad de veces en el segundo. Eso se discrimina en aras de una respuesta más “inteligente”.

```
#include "../deposito/print.h"
#include "../deposito/rnd.h"
#include "busca.h"
#include <iostream>
#include <cstdlib>

int main() {
    using std::cout;
    cout <<
    "Programa 14: b\243squeda visual; si el valor est\240, dice "
    "cu\240ntas veces y d\242nde.\n\n";
    srand( time( nullptr ) );            // una semilla por ejecución
    Ary ary;                             // un arreglo STL
    int clave;                           // el valor a buscar
    Par resp;                            // las respuestas
    int m = 1, M = 100;                  // límites - podrían pedirse
    RELLENA_I( ary, m, M );              // la lista con valores azarosos
    std::sort( ary.begin(), ary.end() ); // la lista ordenada
    TABULA( ary, "Arreglo:\n\n" );       // su tabla a consola

    while ( true ) {
        cout << "T para terminar. \250Clave? ";
        std::cin >> clave;

        if ( std::cin.fail() )
            return EXIT_SUCCESS;

        if ( clave < 0 ) {
            cout << "Un entero positivo, por favor.\n";
            continue;
        } // if

        // resp.first: lugar; resp.second: veces
        resp = posiciones( clave, ary );

        if ( 0 == resp.second )
            cout << "No est\240\n\n";
        else
            1 == resp.second
                ? cout << "Est\240 una vez en la posici\242n " << resp.first << "\n\n"
                : cout << "Est\240 " << resp.second << " veces a partir de la posici\242n "
                       << resp.first << "\n\n";
    } // while
}
```

Una posible salida

```
Programa 14: búsqueda visual; si el valor está, dice cuántas veces y dónde.

Arreglo:

     1       3       3       4       4       4       6       8       8       8
    10      12      14      15      16      17      17      20      22      23
    24      24      25      25      26      29      29      31      32      33
    36      36      42      42      43      43      44      44      46      47
    48      48      48      50      51      58      60      60      61      62
    62      66      66      66      66      67      67      69      69      74
    76      77      78      80      81      81      82      82      87      88
    89      92      95      95      96      96      97      97     100     100

T para terminar. ¿Clave? 2
No está

T para terminar. ¿Clave? 1
Está una vez a partir de la posición 1

T para terminar. ¿Clave? 4
Está 3 veces a partir de la posición 4

T para terminar. ¿Clave? t
```

La eficiencia fue del 46%.

Persistencia

Para que una aplicación sea útil de verdad debe permitir guardar datos e información. Por regla general las aplicaciones profesionales pueden guardar y retomar datos a voluntad del usuario, permitiéndole crearlos desde el mismo programa o tomarlos desde un fichero, procesarlos y grabarlos a otro fichero para que más adelante el usuario pueda recargarlos y proseguir con su labor.

Una aplicación posee persistencia si los datos que ha creado sobreviven a su ejecución. Sin esta capacidad, los datos sólo existen en memoria y se pierden cuando la ejecución termina o cuando la computadora pierde energía eléctrica por corte o por apagado normal. Esto requiere que los datos sean almacenados en un medio secundario, no volátil, tal como un archivo en disco.

Clase de flujos de E/S a ficheros

Las clases para E/S son parte importante de la biblioteca estándar de C++; un programa sin E/S no es de mucho uso. En la actualidad, las clases de E/S de la biblioteca estándar no están restringidas para los archivos o para consola y teclado. En realidad, forman un marco extensible para el formateo de datos arbitrarios y el acceso a las representaciones externas arbitrarias. (Josuttis, p. Cap. 13)

Tal como hay diferentes tipos de E/S, también hay diferentes clases que los manejan, siendo una de las más importantes, la de salida a ficheros **<fstream>**[19], quien define el flujo que puede ser usado:

- Para escribir a un archivo: **ofstream**
- Para leer desde un archivo: **ifstream**

Usualmente hay que crear un código defensivo para comprobar si un archivo abrió para una operación de E/S, tal como se hizo en los programas 6 y 8. Esto eleva la robustez del código. Si la defensa será intolerante o correctora depende puntualmente de la situación, que dictará al programador cuál es la mejor opción. Por ejemplo, con la técnica aquí mostrada la clase que escribe a disco siempre lo hace si el camino existe, pues si el fichero ya está, lo sobrescribe y si no, lo crea. Pero si el camino no se encuentra, entonces lo obvia y no avisa. Por eso a este problema hay que hacerle un código defensivo intolerante pues en este caso, si no puede escribir ¿cuál es el propósito de continuar?

Un nuevo método para generar números aleatorios

En el primer tomo de esta serie se vio la forma estándar de tratar con números seudoaleatorios. El uso de estos números para problemas científicos o comerciales, que surgen mayormente en campos especializados de la matemática y las ingenierías, hace indeseable el mecanismo típico de C++, rudimentario en este sentido, puesto que hay al menos tres cosas a considerar:

1. La aleatoriedad de la función **rand()**, que entrega un patrón de números con un (relativamente) corto periodo de duración, lo que significa que la secuencia eventualmente se repetirá si es llamada con suficiente tenacidad, cuestión que muy bien puede suceder en programas de simulaciones intensivas.
2. El algoritmo de producción de la función **rand()** es delegado enteramente al proveedor del compilador, incidiendo en la calidad del producto, que dependerá del fabricante del IDE usado, ya que unos se conforman más estrictamente que otros a su implementación según la norma. En este sentido es notable el caso de Microsoft™ que trata de integrar todos sus productos en un escenario muy propio, necesariamente alejándose de los estándares establecidos por la comunidad.
3. El valor que entrega la función **rand()** sólo devuelve valores enteros distribuidos linealmente en el rango **[0:RAND_MAX]** esta última constante definida en la biblioteca estándar usualmente como **0x7FFF** hexadecimal o **32,767** decimal. Para obtener números mayores o de punto flotante hay que manipularla, incidiendo negativamente en el efecto seudoaleatorio del resultado, o muy posiblemente perdiéndolo.

Para generar estos números con mayor facilidad y eficacia, la norma C++11 divide su funcionalidad en un motor generador de números seudoaleatorios y una distribución matemática que reparte esos valores en la forma de la distribución y el rango estipulado.

El motor generador empleado en el texto es un Mersenne Twister alimentado con semillas aleatorias dadas por la función estándar **rand()**[20], que devuelve un número tomado al azar del rango **[0:RAND_MAX]** cada vez que se solicita. Eso garantiza una secuencia diferente de números seudoaleatorios cada vez que se invoque el mecanismo. Todo junto queda como: **std::mt19937 mt(rand())**.

[19] **fstream** es **file_stream** o flujo a fichero, en español, **ofstream** es **output_file_stream** o flujo de salida, en español, e **ifstream** es **input_file_stream** o flujo de entrada, en español.

[20] (Halterman) plantea que el mecanismo trabaja impecablemente cuando el motor se alimenta de una semilla brindada por la clase **random_device**, pero genera el mismo valor aun cuando cambie la ejecución. EODDA el método mostrado es mejor.

Se usa la distribución correspondiente en dos pasos. Para la Distribución Lineal Uniforme se creó la función **`dist`** que distribuye los valores generados en el rango dado por p y q: **`std::uniform_int_distribution<int>dist(p, q)`**. Lo mismo se hizo con la distribución uniforme para valores de punto flotante y la de Gauss.

Seguidamente, el tercer parámetro de la función generadora se arma con **`bind()`**, un functor de adaptación que une la distribución y el motor como un todo en un tercer parámetro dado por una función de usuario: **`auto g = std::bind(distro, motor)`**

Por último, el algoritmo generador **`gen()`** espera tres parámetros: el punto de partida del contenedor que se rellena, el punto de culminación y la función que realmente genera esos números, ensamblada en el paso anterior: **`gen(begin(K), end(K), g)`**[21]

Tabla 28. Motor y distribuciones usados en el texto para generar números seudoaleatorios

Motor generador
`mersenne_twister_engine`: Motor generador de números seudoaleatorios desarrollado por Matsumoto y Nishimura. Reputado por su calidad, su nombre proviene del hecho que la longitud del periodo corresponde a un número primo de Mersenne. Viene en dos variantes: la más usada (y más portable) es MT19937, de 32 b; la otra, de 64 b es MT19937-64
Distribución matemática
`uniform_int_distribution<int>Dist(mín, max)`: Distribución Uniforme Discreta, que asume en el dominio un número finito de valores, ***producto de un conteo*** hecho con la misma probabilidad de ocurrencia. El dominio está definido por dos parámetros indicados por los valores mínimo y máximo deseados. Su rango es **`[min:max]`**
`uniform_real_distribution<double>Dist(min, max)`: Distribución Uniforme Continua, en la que todos los intervalos de igual longitud en el rango **`[min:max)`**, ***producto de una medición***, tienen la misma probabilidad de ocurrencia. El dominio está definido por dos parámetros indicados por los valores mínimo y máximo deseados, incluyendo al primero y excluyendo al último.
`normal_distribution<double>Dist(media, desv)`: Distribución Estándar $\boldsymbol{Z \sim N(\mu, \sigma)}$. Cuando el promedio es cero $(\boldsymbol{\mu = 0})$ y la desviación típica es uno $(\boldsymbol{\sigma = 1})$, se dice que ***está normalizada*** y sus valores vienen dados en tablas estadísticas *ad hoc*.

Programa 15: distribución de Gauss

En estadística y probabilidad se llama distribución normal o distribución de Gauss, a una familia de variables numéricas continuas que frecuentemente aparece muy aproximada en fenómenos reales. (Wikipedia en español)

En el programa 28 de (Introducción al C++, tomo 1, pág. 192) se pedía generar 200 números enteros aleatorios y ponerlos en un archivo llamado C:\temp\datos.txt. Acá se le da un giro, estipulando que los valores sean tomados de una distribución normalizada de Gauss (algo que simplemente, la función **`rand()`** no puede hacer) y puestos en un archivo llamado C:\temp\gauss.txt

Definición

```
#ifndef SALIDA_H_INCLUDED
#define SALIDA_H_INCLUDED

#include <string>
typedef std::string Str;
const Str ERR( "No pude abrir el archivo; abortando.\n" );

#include <vector>
typedef std::vector<double> Vct;

//  escribe los valores del arreglo ary al archivo archy
//  entrada: nombre del archivo, y nombre y tamaño del arreglo
```

[21] Este es un buen momento para revisar el código de las plantillas generadoras de números aleatorios.

```
//  salida: los valores están en el archivo
void alArchivo( Str archy, const Vct & vct );

#endif // SALIDA_H_INCLUDED
```

Declaración

Se comprueba la apertura del archivo: si todo va bien, se graban los valores; otrosí se emite un mensaje y se aborta.

```
#include "salida.h"
#include <fstream>

//  valores al archivo
void alArchivo( Str archy, const Vct & vct ) {
    std::ofstream Escribiendo( archy );

    if ( Escribiendo.is_open() ) {
        for ( auto x : vct )
            Escribiendo << x << '\n';
    } else {
        throw ERR;
    } // if-else
}
```

Programa

```
#include "../deposito/print.h"
#include "../deposito/rnd.h"
#include "salida.h"

const int N      = 200;                 // N números
const short COLS = 10;                  // tabla de COLS columnas
const Str archy( "C:/temp/gauss.txt" ); // la salida - podría pedirse

int main() {
    std::cout <<
    "Programa 15: escribe " << N << " valores gaussianos\n"
    "en el archivo \"" << archy << "\"\n\n";
    Vct data( N ); // un vector de 200 dobles
    GAUSS( data ); // números gaussianos normalizados

    try {
        alArchivo( archy, data );                      // se escriben a disco
        TABULA( data, "Valores:\n\n", 3, true, COLS ); // valores a consola
    } catch( Str ) {
        std::cerr << ERR;
        return EXIT_FAILURE;
    } // try-catch

    system( "pause" );
    return EXIT_SUCCESS;
}
```

Dos salidas

- Con error: en este sistema la torre J no existe. Si no se posee la ubicación C:/temp, la ejecución dará el mismo error. Una vez que la localización está dispuesta, el programa no falla.

```
Programa 15: escribe 200 valores gaussianos
en el archivo "J:/temp/gauss.txt"

No pude abrir el archivo; abortando.

Process returned 1 (0x1)   execution time : 0.021 s
```

- Sin error

```
Programa 15: escribe 200 valores gaussianos
en el archivo "C:/temp/gauss.txt"

Valores:

-0.177  -0.861  -0.754  +1.020  -1.067  +0.387  -1.930  +1.698  +0.310  -0.709
+0.216  +0.898  -0.052  +2.811  -1.416  -1.885  +0.215  -1.310  -0.668  +0.017
-2.002  +1.237  +0.259  +1.087  +0.481  +1.772  +0.586  +1.467  +0.145  +0.917
+0.625  -0.046  +0.553  -0.726  -0.079  +0.067  +0.498  -0.548  -0.519  +0.714
-0.482  +1.772  +0.398  +0.665  -0.024  -0.168  -0.433  +1.508  +0.613  +0.590
-0.712  +1.438  +0.622  -2.156  -0.885  -0.275  +0.516  +1.182  -0.137  -0.436
-1.623  -0.505  -0.552  -0.171  +0.422  +0.216  -0.436  +0.605  -1.753  -1.057
-1.282  +2.068  +0.350  -0.760  +0.142  -0.646  -1.005  +0.417  -0.784  -0.328
-2.015  -0.847  +0.880  -1.817  -0.724  +0.100  -1.808  -0.727  +0.990  -0.444
-1.099  -0.707  -1.109  -0.385  -0.331  -1.049  -0.824  -1.444  +0.469  +0.085
+0.969  -1.904  -0.842  -0.362  +0.364  -0.591  +1.873  -0.219  -1.848  -0.164
+1.386  -0.793  +1.372  -0.125  -0.118  +0.207  -0.267  -0.699  +0.057  -0.949
-0.292  -0.115  +0.720  +0.314  +0.387  +0.540  +0.648  -2.070  +0.244  -2.053
-1.326  -0.174  -0.098  +0.459  +0.653  -0.372  +0.352  +0.433  +1.162  -0.173
+2.667  -0.226  +0.489  +1.403  +1.542  -1.730  +0.101  +1.505  +2.075  +0.275
+0.047  -1.074  +0.300  +0.043  -1.185  -0.077  -0.085  +0.294  -1.043  +0.851
-0.253  +0.603  -0.854  +0.808  +0.381  +0.141  -0.806  -0.013  +1.235  -0.612
-0.973  +0.327  +0.229  +0.424  -0.040  +1.585  +1.216  +0.019  +0.989  -0.037
-1.906  -0.046  -0.133  -0.828  -1.457  +0.926  +0.017  -0.840  -1.535  +0.906
+1.846  -1.143  +0.242  -0.556  -1.238  -1.121  -0.261  +1.073  +2.289  -0.084
```

La eficiencia fue del 47%

Manipuladores hechos a la medida

C++ proporciona varios manipuladores de flujos que realizan tareas de formato. Los manipuladores de flujos proporcionan herramientas para establecer las anchuras de los campos, establecer la precisión, establecer y quitar el formato de estado, establecer el carácter de relleno en los campos, vaciar flujos, insertar una nueva línea en el flujo de salida (y vaciar el flujo), insertar un carácter nulo en el flujo de salida y omitir el espacio en blanco en el flujo de entrada. (Deitel & Deitel, pág. 658)

Los manipuladores hechos a medida son importantes porque un manipulador puede agrupar una secuencia de varias operaciones de E/S independientes. Además, no es inusual encontrar situaciones en las que se repite frecuentemente la misma secuencia de operaciones de E/S dentro de un programa. En estos casos puede utilizarse un manipulador a medida para realizar estas operaciones, simplificándose de este modo el código fuente e impidiendo que se produzcan errores accidentales.

La OOP admite manipuladores hechos a la medida, pero estos también pueden aplicarse fuera de ese marco, en programas que no están orientados a objetos, tal y como se ha venido haciendo hasta ahora. Ellos pueden, con buena eficiencia, simplificar el código de programas que hagan un uso intensivo de operaciones de E/S.

Existen dos tipos básicos de manipuladores: los que operan sobre flujos de entrada y los que lo hacen sobre flujos de salida y que pueden o no tener argumentos. ¿La buena noticia? La codificación de las funciones de los manipuladores no parametrizados es bastante sencilla. Todas responden a este esquema:

- De salida: retorna una referencia a un flujo de salida:

```
ostream & nombreSale(ostream &flujo) {
      ⋮ // aquí va el código
   return flujo;
}
```

- De entrada: retorna una referencia a un flujo de entrada:

```
istream & nombreEntra(istream &flujo) {
      ⋮ // aquí va el código
   return flujo;
}
```

Notas:

- Aun cuando por su definición parece que los manipuladores reciben un parámetro, en las operaciones de E/S se llaman sin argumento. Por ejemplo, **`nombreEntra >> unValor;`**
- Los manipuladores reciben una referencia al flujo para el que han sido invocados y deben devolverlo, pues si no lo hacen, no podrían encadenarse en una secuencia de operaciones E/S. Por ejemplo, **`cout << unValor << otroValor;`**

Programa 16: estadísticas

En el programa 29 de (Introducción al C++, tomo 1, pág. 196) se pedía leer los números contenidos en un archivo cuyo nombre se definió como C:\temp\datos.txt, y calcularles sus estadísticos descriptivos. Pero la solución hallada tenía dos problemas: sólo leía enteros y sólo calculaba hasta 200 valores antes de tener que recompilar. Esta solución resuelve el problema de la memoria, calcula dobles y usando el **`narrowing`**[22], también calcula enteros.

Como se sabe que en este caso los valores generados son gaussianos normalizados, una comprobación QAD de la verdadera forma de esa distribución particular, es verificar lo que se dice en el recuadro que sigue.

> Una *distribución gaussiana normalizada perfecta* es simétrica, no tiene moda, la media y la mediana valen cero, la desviación típica vale uno, el 99,74% de sus valores están en los límites ± tres, y cuando el alcance tiene un valor absoluto muy aproximado de seis su gráfica es simétrica y estándar en forma de campana, llamada justamente “Campana de Gauss”, tal como se muestra en la ilustración.

Hay tres amplitudes gaussianas: la platicúrtica o amplitud plana, la mesocúrtica o amplitud mediana y la leptocúrtica o amplitud aguda. En la medida en que sean realmente masivos los datos, más se acerca la gráfica de la distribución real a

[22] Esto se debe a un fenómeno de conversión automática entre tipos llamado **`narrowing`** (estrechamiento, en español) que permite al programa cambiar el tipo de un entero por el de uno doble sin perder precisión. Lo contrario se llama **`wideninig`** (ensanchamiento, en español) y en general no garantiza mantener la precisión, porque si el número de punto flotante trae decimales, al pasar a entero se trunca y si el número es tipo **`double`** o mayor, al pasar a **`float`** también se trunca. Hay una solución mejor, pero es una historia del tercer tomo.

la gráfica de la distribución teórica. En la medida en que se incumplen las premisas resaltadas en el recuadro, debido al azar más se aleja la distribución real de la teórica.

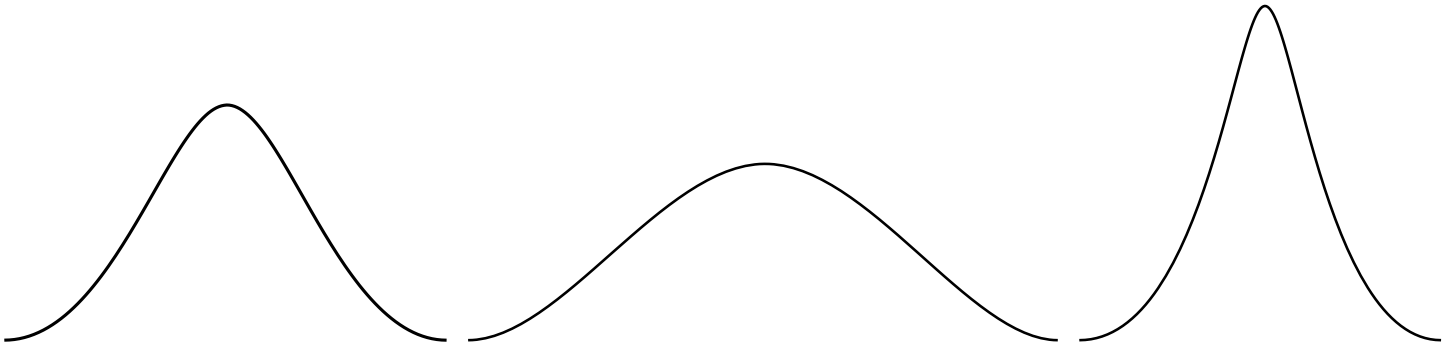


Ilustración 9. Distribuciones normalizadas de Gauss

Definiciones

Se usará una salida formateada para los números reales mediante el manipulador **muestra()** A la estructura se le añade el contenedor **vector**, formando una clase por composición, se le adicionan los estadígrafos y un método que carga los valores desde un archivo. Ahora la estructura puede manejar sus métodos y gestionar sus componentes, no importa si está en la pila o en la tienda libre.

```
#ifndef ENTRADA_H_INCLUDED
#define ENTRADA_H_INCLUDED

#include <algorithm>
#include <ostream>
#include <string>
typedef std::string Str;
const Str ERROR( "\nNo pude abrir el archivo; abortando.\n" );
typedef std::vector<double> Vct;

struct Estads {
    //  interfaz
    double suma     ();
    double mediana  ();
    double moda     ();
    double promedio ();
    double varianza ();
    double miMax    ();
    double miMin    ();
    double cuenta   ();
    void leerArchivo( Str archy );
    // miembro
    Vct vec;
};

//  flujo de salida especializado
void muestra( std::ostream & escribe, Str ID, double stat );

#endif // ENTRADA_H_INCLUDED
```

Declaraciones

Los estadígrafos fueron codificados según las definiciones dadas por la Estadística elemental; ante la duda, consultar un texto afín. No obstante, se advierte que el cálculo de la varianza se hizo mediante la transformación algebraica del binomio al cuadrado perfecto, eliminando el ciclo que conlleva la fórmula clásica por tres instrucciones lineales. Se usa el algoritmo

```
transform(K1.begin(), K1.end(), // el rango de transformación
          K2.begin()            // la entrada del segundo contenedor
          K3.begin(),           // la entrada del resultado
          oper);                // la operación a realizer
```

que puede trabajar con uno, dos o tres contenedores realizando la operación deseada con los dos primeros y poniendo el resultado en el tercero. Si todo se hace con el mismo contenedor, la operación es de multiplicación y se vuelve a poner en el mismo, el resultado es de elevar el vector al cuadrado. Y si luego se suma todo, se obtiene la suma de cuadrados o SDC.

La moda presenta un problema: si su secuencia es uno para una distribución de valores, simplemente y por definición, no existe. Entonces se devuelve el valor **HUGE_VAL**, reservado por C++ para errores de desbordamiento y se examina la respuesta en el flujo de salida. Como siempre, se hace programación defensiva al abrir el archivo.

```
#include "entrada.h"
#include <iostream>
#include <fstream>

//  valores leídos desde el archivo
void Estads::leerArchivo( const Str archy ) {
    // aquí hace falta código defensivo, porque
    // si el fichero no está, no puede abrirlo
    std::ifstream Leyendo;
    Leyendo.open( archy );

    if ( Leyendo.is_open() ) {
        double val;

        while ( Leyendo >> val )
            vec.push_back( val );

        sort( vec.begin(), vec.end() );
    } else {
        throw ERROR;
    } // if-else
}

double Estads::suma()     { return accumulate(vec.begin(), vec.end(), 0.0); }
double Estads::promedio() { return suma() / vec.size(); }
double Estads::miMax()    { return *max_element(vec.begin(), vec.end()); }
double Estads::miMin()    { return *min_element(vec.begin(), vec.end()); }

// mediana
double Estads::mediana() {
    double med;
    int n = vec.size();

    int idx = 0.5 * n; // se buscó el índice del medio
    n % 2              // se calculó la mediana
        ? med = vec[idx]
        : med = 0.5 * ( vec[idx - 1] + vec[idx] );
    return med;
}

// moda
```

```
double Estads::moda() {
    double laModa, unaModa;
    int laRacha, unaRacha;
    laModa = unaModa = vec[0];
    laRacha = unaRacha = 0;
    unsigned i = 0;           // el índice de la secuencia
    unsigned n = vec.size(); // el tamaño

    do {
        if ( unaModa not_eq vec[i] ) {  // ¿cambió la moda?
            unaModa = vec[i];           // SI: actualizarla

            if ( unaRacha > laRacha ) { // ¿cambió la racha?
                laRacha = unaRacha;     // SI: actualizarla y...
                laModa  = vec[i - 1];   // ...actualizar la moda
            } // if-interior

            unaRacha = 0; // contar otra racha
        } // if-exterior

        ++unaRacha; // incrementa la racha
        ++i;        // incrementa el índice
    } while ( i not_eq n ); // do-while

    if ( 1 == laRacha )
        laModa = static_cast<double>( HUGE_VAL );

    return laModa;
}

// varianza muestral
double Estads::varianza() {
    std::vector<double>aux( vec );
    int n = vec.size();
    transform( aux.begin(), aux.end(),     // el rango a multiplicar
               aux.begin(),                // la 2da. entrada
               aux.begin(),                // a partir de aquí
               std::multiplies<double>()  // multiplica 1ra. por 2da.
             );
    double SDC = accumulate( aux.begin(), aux.end(), 0.0 ); // suma de cuadrados
    return (SDC - suma() * suma() / n) / ( n - 1 ) ;
}

#include <iomanip>
#include <ios>

// la escritura
void muestra( std::ostream & escribe, Str ID, double stat ) {
    using std::cout;
    std::ios::fmtflags oldFlags = escribe.flags();
    escribe.setf( std::ios::fixed );
    escribe << std::setprecision( 4 );

    if ( "Cuenta = " == ID ) {
        escribe << std::setprecision( 0 );
        escribe << std::setw( 12 ) << ID << std::setw( 4 ) << stat << '\n';
    } else if ( HUGE_VAL == stat ) {
        escribe << "     " + ID + " #N/A" << '\n';
    } else {
        escribe << std::setw( 12 ) << ID << std::setw( 9 ) << stat << '\n';
    } // if-else
    escribe.flags(oldFlags);
```

```
}
```

Programa

```
#include "entrada.h"
#include <iostream>
#include <fstream>
#include <iomanip>
#include <cmath>

int main() {
    using std::cout;
    cout << "Programa 16: lee los n\243meros de un archivo\n"
           "y calcula sus principales estad\241sticos\n\n";
    Estads est; // los estadísticos
    Str archy;  // nombre de los archivos de lectura y escritura
    cout << "Nombre del archivo a leer: ";
    std::cin >> archy;

    //  código defensivo aquí
    try {
        cout << "Leyendo del archivo " << archy;
        est.leerArchivo( archy );
        cout << "...hecho.\n\n";
    } catch( Str ) {
        std::cerr << ERROR;
        return EXIT_FAILURE;
    } // try-catch

    //  respuestas a consola
    cout << "Estad\241sticos a consola:\n\n";
    muestra( cout, "Media = ",     est.promedio() );
    muestra( cout, "Mediana = ",   est.mediana()  );
    muestra( cout, "Moda = ",      est.moda()     );
    muestra( cout, "Varianza = ",  est.varianza() );
    muestra( cout, "Desviac. = ",  sqrt( est.varianza() ) );
    muestra( cout, "M\240ximo = ", est.miMax()    );
    muestra( cout, "M\241nimo = ", est.miMin()    );
    muestra( cout, "Rango = ",     est.miMax() - est.miMin() );
    muestra( cout, "Suma = ",      est.suma()     );
    muestra( cout, "Cuenta = ",    est.vec.size() );
    cout << "\n";

    //  respuestas al disco
    cout << "Nombre del archivo a escribir: ";
    std::cin >> archy;

    // un flujo de salida
    std::ofstream Escribiendo;
    Escribiendo.open( archy );

    if ( Escribiendo.is_open() ) {
        cout << "Escribiendo al archivo " << archy;
        muestra( Escribiendo, "Media = ",     est.promedio() );
        muestra( Escribiendo, "Mediana = ",   est.mediana()  );
        muestra( Escribiendo, "Moda = ",      est.moda()     );
        muestra( Escribiendo, "Varianza = ",  est.varianza() );
        muestra( Escribiendo, "Desviac. = ",  sqrt( est.varianza() ) );
        muestra( Escribiendo, "M\240ximo = ", est.miMax()    );
        muestra( Escribiendo, "M\241nimo = ", est.miMin()    );
        muestra( Escribiendo, "Rango = ",     est.miMax() - est.miMin() );
```

```
        muestra( Escribiendo, "Suma = ",      est.suma()     );
        muestra( Escribiendo, "Cuenta = ",    est.vec.size() );
        cout << "...hecho.\n\n";
    } else {
        std::cerr << ERROR;
        return EXIT_FAILURE;
    } // if-else

    system( "pause" );
    return EXIT_SUCCESS;
}
```

Dos salidas

- Con error: la aplicación detecta si el fichero no existe, porque por cualquier causa no está donde se dice o su nombre está incorrectamente escrito. Una vez que se pone la ubicación o el nombre correctos, el programa no falla.

```
Programa 16: lee los números de un archivo
y calcula sus principales estadísticos

Nombre del archivo a leer: c:\temp\gaussian.txt
Leyendo del archivo c:\temp\gaussian.txt
No pude abrir el archivo; abortando.

Process returned 1 (0x1)   execution time : 11.265 s
```

- Sin error, datos gaussianos:

```
Programa 16: lee los números de un archivo
y calcula sus principales estadísticos

Nombre del archivo a leer: c:\temp\gauss.txt
Leyendo del archivo c:\temp\gauss.txt...hecho.

Estadísticos a consola:

    Media =   -0.0469
  Mediana =   -0.0459
    Moda =  #N/A
 Varianza =    0.9794
 Desviac. =    0.9897
   Máximo =    2.8108
   Mínimo =   -2.1563
    Rango =    4.9671
     Suma =   -9.3852
   Cuenta =  200

Nombre del archivo a escribir: c:\temp\respD.txt
Escribiendo al archivo c:\temp\respD.txt...hecho.
```

La eficiencia es del 28%. Al ser contrastados los datos en Excel™, los resultados fueron iguales:

Media	Mediana	Moda	Varianza	Deviación	Máximo	Mínimo	Rango	Suma	Cuenta
-0.0469	-0.0459	#N/A	0.9794	0.9897	2.8108	-2.1563	4.9671	-9.3852	200

Un análisis indica que la media y la mediana están cerca del cero, ambas al lado izquierdo del eje de valores. La desviación, cerca de uno, pero por la derecha (≈0.9897), sugiriendo una concentración de valores alrededor del eje central; el extremo menor está en –2.1563 y el extremo mayor en 2.8108, indicando una leve asimetría también a la

derecha. El rango está alrededor de 5, sugiriendo una distribución leptocúrtica. En este caso, y como era de esperar, la distribución no tiene moda, porque ninguno de sus valores se repite.

Una salida extra

El programa puede trabajar también con números enteros.

```
Programa 16: lee los números de un archivo
y calcula sus principales estadísticos

Nombre del archivo a leer: c:\temp\datos.txt
Leyendo del archivo c:\temp\datos.txt...hecho.

Estadísticos a consola:

    Media =   48.7650
  Mediana =   46.5000
     Moda =    6.0000
 Varianza =  821.5174
 Desviac. =   28.6621
   Máximo =   99.0000
   Mínimo =    1.0000
    Rango =   98.0000
     Suma = 9753.0000
   Cuenta =  200

Nombre del archivo a escribir: c:\temp\respI.txt
Escribiendo al archivo c:\temp\respI.txt...hecho.
```

El programa no “sabe” que trabaja con enteros: todo lo ve como **`double`** y le aplica su precisión; el remedio está en el tercer tomo. Al ser el resultado de una distribución lineal, no procede hacer el análisis contra una curva de Gauss.

Refinamiento de las E/S

¿Qué tal si se desea añadir el último dato obtenido al archivo? Es decir, conservar los datos generados entre sucesivas aplicaciones del programa, e ir enriqueciendo el archivo con los últimos datos obtenidos. Hasta ahora, al escribir a un archivo, lo que se hace es sobrescribirlo, sin retener los datos previos. Eso ha servido a las soluciones halladas hasta el momento, pero la persistencia es realmente útil cuando el trabajo entre sesiones se guarda y recupera más tarde.

Entonces hay que establecer el modo de apertura y el tipo de acceso adecuados. La función **`open()`** es miembro de las clases de los tres tipos de flujos (entrada, salida y error) y su prototipo es:

```
void open(const char *nombre, int acceso);
```

donde **`*nombre`** es el puntero a una cadena ANSI C con el nombre del archivo y **`acceso`** es una constante simbólica que determina el modo de su apertura. La siguiente tabla muestra sus valores.

Tabla 29. (simplificada) Modos de acceso a un archivo

Constantes	Resultado
`ios::app`	Añade los datos al final del archivo: **`append`**
`ios::ate`	Abre por el final del archivo: **`at end`**
`ios::binary`	Abre el archivo en modo binario: **`binary`** Por omisión lo abre en modo texto
`ios::in`	Abre el archivo para entrada: **`input`**. Por omisión si se usa un **`istream`**
`ios::nocreate`	Da error si el archivo aún no existe: **`do not create it`**

ios::noreplace	Da error si el archivo ya existe: **do not replace it**
ios::out	Abre el archivo para salida: **output** Por omisión si se usa un **ostream**
ios::trunc	Trunca el archivo, eliminando sus datos: **truncate it**

Al especificar el modo pueden combinarse dos o más valores aplicándoles operador *o-binario* **(|)** Por ejemplo, para escribir datos añadiéndolos a un archivo **archy**:

```
ofstream escribe(archy, ios::out | ios::app);
```

y para leer del archivo **archy**:

```
ifstream lee(archy);
```

Programa 17: acuñando el momento

El autor es diabético y confrontó esta situación: en Cuba la medición de la tasa de azúcar en sangre se realiza en milimoles por litro (mmol/l), pero en muchas otras naciones los glucómetros empleados lo hacen en miligramos por decilitro (mg/dl.) Desafortunadamente, su glucómetro es de estos últimos y no posee uno estándar para el país. Entonces, creó un conversor de lecturas, cuyos resultados son mantenidos en un archivo llamado C:\temp\glucosa.txt, que lleva el historial de seguimiento del paciente[23].

A cada valor le asoció la fecha y hora de la recogida de la muestra de sangre; el programa hace la conversión y muestra el historial hasta ese momento.

Definición

Se usan dos variables para los dos errores posibles: de lectura, de escritura. La lectura baja se controla por **assert()**

```
#ifndef CONVERSOR_H_INCLUDED
#define CONVERSOR_H_INCLUDED

#include <string>
typedef std::string Str;
const Str ERR_E( "No pude escribir. Abortando.\n" );
const Str ERR_L( "No pude leer. Abortando.\n" );

#include <fstream>
#include <vector>
typedef std::vector<Str> Vct;
typedef std::pair<int, float> Par;

//  la clase toma un valor en mg/dL y lo entrega como mmol/L
//  escribe la información a disco y a consola
struct Lector {
    Lector(); // constructor
    // métodos
    Par tomarLectura();
    void mostrarInfo( std::ostream & escribir, Par dat );
    void leerData( std::istream & leer );
    //  miembros
    Str archy; // archivo de datos
    Str fecha; // fecha y hora actuales
    Vct vct;   // vector que carga del archivo
```

[23] Los datos mostrados son tomados de la realidad.

```
};

#endif // CONVERSOR_H_INCLUDED
```

Declaración

La función **time(puntero)** devuelve el tiempo calendario actual del sistema (fecha y hora.) Si el puntero es a una variable del tipo **time_v**, ella toma el tiempo. La función **ctime()** devuelve una cadena de la forma **día_de_semana día_de_mes hora:minuto:segundo año**. Su búfer es sobrescrito cada vez que se llama; si se necesita preservar su contenido, hay que ponerla en otra variable.

```
#include "conversor.h"

const int ANCHO = 5; // ancho del campo de salida

//  convierte los mg/dl en mmol/l; fórmula de Wikpedia
double inline convierte( int valor ) { return ( valor - 3.75 ) / 17.5; }

#include <ctime>

Lector::Lector() {
    archy = "C:/temp/glucosa.txt";
    time_v laFecha;             // variable para la fecha
    time( &laFecha );           // obteniendo la fecha actual
    fecha = ctime( &laFecha );  // preservando la fecha
    vct = vct.clear();          // el vector
}

#include <iostream>
#include <cassert>
Par Lector::tomarLectura() {
    int mg; // mg/dl
    std::cout << "Programa 17: GLUC\340METRO conversor de lecturas.\n\nLectura (mg/dl > 56): ";
    ( std::cin >> mg ).get();
    assert( mg > 56 );
    float mmol = convierte( mg ); // mmol/l
    return std::make_pair( mg, mmol );
}

#include <iomanip>
void Lector::mostrarInfo( std::ostream & escribir, Par dat ) {
    if ( escribir.fail() ) throw ERR_E;

    escribir  << std::setw( ANCHO ) << dat.first << " mg/dl equivalen a "
              << std::fixed << std::right << std::setprecision( 1 )
              << std::setw( ANCHO - 1 ) << dat.second << " mmol/l; " << fecha;
}

//  por simplicidad hace dos cosas: toma los datos
// del historial y los pone en consola
void Lector::leerData( std::istream & leer ) {
    if ( leer.fail() ) throw ERR_L;

    Str linea;

    while ( getline( leer, linea ) )
       vct.push_back( linea );
    std::cout << "\nHistorial:\n";

    for ( auto x : vct )
```

```
        std::cout << x << '\n';

    std::cout << '\n';
}
```

Programa

```
#include "conversor.h"
#include <iostream>
#include <cstdlib>
#include <ios>

int main() {
    Lector * pLect = new Lector;
    Par dat;

    try {
        dat = pLect->tomarLectura();
    } catch ( Str ) {
        std::cerr << ERR_L;
        return EXIT_FAILURE;
    } // try-catch

    try {
        std::ofstream aDisco( pLect->archy, std::ios::app );
        pLect->mostrarInfo( std::cout, dat );
        pLect->mostrarInfo( aDisco, dat );
    } catch ( Str ) {
        std::cerr << ERR_E;
        return EXIT_FAILURE;
    }; // try-catch

    std::ifstream deDisco( pLect->archy );
    pLect->leerData( deDisco );
    delete pLect;

    system( "pause" );
    return EXIT_SUCCESS;
}
```

Salidas

- Error de lectura: se le dio un valor ilegal para probar la acción.

```
Programa 17: GLUCÓMETRO conversor de lecturas.

Lectura (mg/dl > 56): 50
Assertion failed!

Program: E:\Texto\Tomo 2\Programas\p17-glucosa\bin\Release\prog17.exe
File: E:\Texto\Tomo 2\Programas\p17-glucosa\conversor.cpp, Line 31

Expression: mg > 56

This application has requested the Runtime to terminate it in an unusual way.
Please contact the application's support team for more information.
```

- Salida normal: los datos son tomados de la vida real.

```
Programa 17: GLUCÓMETRO conversor de lecturas.

Lectura (mg/dl > 56): 156
  156 mg/dl equivalen a  8.7 mmol/l; Wed Sep 30 19:44:45 2020

Historial:
  119 mg/dl equivalen a  6.6 mmol/l; Sat Aug 08 08:57:02 2020
  140 mg/dl equivalen a  7.8 mmol/l; Mon Aug 17 07:58:16 2020
  103 mg/dl equivalen a  5.7 mmol/l; Sun Aug 23 07:53:35 2020
  174 mg/dl equivalen a  9.7 mmol/l; Tue Sep 01 06:17:46 2020
  179 mg/dl equivalen a 10.0 mmol/l; Sun Sep 06 05:10:34 2020
  169 mg/dl equivalen a  9.4 mmol/l; Sat Sep 12 08:44:29 2020
  156 mg/dl equivalen a  8.7 mmol/l; Wed Sep 23 06:07:14 2020
  156 mg/dl equivalen a  8.7 mmol/l; Wed Sep 30 19:44:45 2020
```

La eficiencia es del 30%

5 –ESTRUCTURAS COMPLEJAS DE DATOS

> *Hasta el momento se han visto contenedores que usan como nodos a los tipos básicos del lenguaje (**`int`**, **`float`**, etc…), pero ¿qué pasa si la secuencia envuelve algo más complejo, por ejemplo, nodos que son otros contenedores o son estructuras? Aquí se hace necesario utilizar más a fondo el potencial que brinda la STL, pues mientras más se ajuste un contenedor a la estructura de datos del problema, más fácil, conciso y claro resultará el código y más eficiente será el trabajo. El Autor.*

En el texto se analizarán tres tipos de datos complejos: una estructura que posee uno o más miembros de la STL, un contenedor que almacena una estructura, que a su vez puede ser compleja, y un contenedor que almacena otro contenedor.

El secreto de manejar las indirecciones en C++ estriba en ver el tipo de dato que alberga el contenedor principal; para trabajar el dato, se descarga su nodo. Si el nodo a su vez contiene otras capas, se descargan estas y así sucesivamente…

Ordenando al contenedor

Por su tipo, el ordenamiento podría ser:

1. De orden estricto. Cuando dos elementos tienen los mismos campos y sus datos homólogos son iguales: se dice que los elementos bajo comparación son idénticos o iguales y su orden es estricto.
2. De orden débil. Cuando dos elementos tienen los mismos campos, pero difieren en algún valor, que no necesariamente en todos, entonces se dice que los elementos bajo comparación son similares o equivalentes y su orden es débil.

Pero surge un problema: si los datos son estructurados, ¿cómo se obtendría un tipo de ordenamiento sobre un campo complejo? Porque los métodos de ordenamiento no pueden trabajar directamente con contenedores no asociativos cuyos nodos estén estructurados. ¿La solución? O usar el algoritmo de comparación lexicográfica, o programar el operador de comparación.

a) El código prototipo de una función de ordenación con ordenamiento débil, de menor a mayor es:

```
//  función de ordenamiento débil, de menor a mayor
bool menor(const K &d1, const K &d2) {
    if ( d1.valor1 < d2.valor1 ) return true;
    return false;
}
```

donde `K` es el tipo de estructura C++ a ordenar, y `d1.valor1`, `d1.valor2`, `d2.valor1` y `d2.valor2` son los campos homólogos por dónde será débilmente ordenado el contenedor, y para ordenamiento fuerte es

```
//  función de ordenamiento fuerte, de menor a mayor
bool menor(const K &d1, const K &d2) {
    if ( d1.valor < d2.valor ) return true;
    if ( d1.valor1 == d2.valor1 and d1.valor2 < d2.valor2 ) return true;
    return false;
}
```

donde `K` es el tipo de estructura C++ a ordenar, y `d1.valor` y `d2.valor` son los campos homólogos por dónde será fuertemente ordenado el contenedor. Pero sólo son prototipos. El codificador debe tener presente que están afectados por los niveles de indirección que aporta el grado de complejidad de la estructura de datos. Para invertir el ordenamiento, se cambia el signo.

b) En los contenedores no asociativos el algoritmo `sort()` es llamado como: `sort( K.begin(), K.end(), menor );`

c) En la lista, el ordenamiento está integrado y la llamada es: `list.sort( menor );`

En la mayoría de los casos un ordenamiento débil basta y desde luego, es algo más sencillo de codificar, pero el programador debe conocer totalmente este concepto, para poder aplicar una solución adecuada a cualquier complicación que tenga entre manos.

Una estructura que posee algún miembro de la STL

En C++ la estructura es una especie de clase paralela (hasta el momento en que esto se escribe, ***ES*** una clase) y puede contener entre otros tipos de datos a otras estructuras, o a uno o más contenedores, lo cual se llama composición por agregación, permitiendo la creación de estructuras complejas y dejando ver al paciente lector de una manera…gentil el comienzo de la programación orientada a objetos.

En el texto se han usado para manejar complejidades en los programas 5, 7, 9 y 16. EODA y luego de terminar este quinto tema, será el momento adecuada para que el lector vuelva a ellos y los analice con otra perspectiva, más profunda y mejor enfocada.

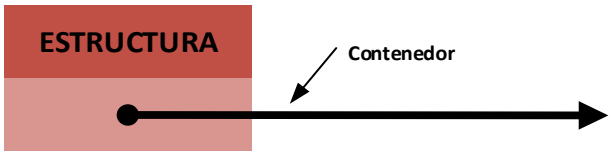


Ilustración 10. Una estructura con agregación de un contenedor

Una colección de estructuras C++

Si llamamos nodo a un tipo indeterminado de dato que esté representado en una secuencia tal como lo es un vector, entonces, al usar una secuencia cualquiera, las seis operaciones básicas (no importa su orden) son mostrar la secuencia, contar los nodos, insertar un nodo, editar un nodo, borrar un nodo y verificar si un nodo está presente. Además, a veces es de interés ordenar la secuencia.

- ✓ Nodo: estructura dividida lógicamente en dos secciones: una de datos, que puede ser compleja y/o extensa; y otra de enlace, que tiene al menos un puntero para referenciar a otro nodo similar. (Introducción al C++, tomo 1, pág. 165)

(Sutter, Exceptional C++: 47 Engineering Puzzles, Programming Problems, and Solutions, p. Item 6) señala que en la mayoría de los contenedores, en muchas ocasiones la salida de un ciclo se establece llamando al método **`end()`**; como en **`i ¡= miObjeto.end()`**. Entonces un objeto temporal será construido, la instrucción completada y dicho objeto, destruido en cada ciclo.

Como ese valor no cambia (se dice que es un valor invariante dentro del ciclo), con cada realización esa secuencia se volverá a efectuar, siendo tanto innecesaria, como ineficiente. Por lo tanto, el valor deberá ser almacenado en una variable antes del lazo y esa es la variable que será implementada.

Como ya se vio, **`end()`** devuelve un puntero al valor que va inmediatamente después del final de la secuencia del contenedor **`K`**. La instrucción genérica sería:

```
typename T::iterator i;           // un repetidor de inicio
typename T::iterator end(K.end()); // un repetidor de avance que marca el final
```

y la llamada sería:

```
for ( i = K.begin(); i != end; ++i ) { ... }
```

Nunca declarar los valores invariantes dentro del ciclo

A continuación, un ejemplo muy simple, pero que ilustra esta técnica y de paso le permite ver una forma de ordenar la secuencia compleja.

Programa 18: lista de estructuras

En el programa 7 se adaptó una solución STL al programa 24 de (Introducción al C++, tomo 1, pág. 167) que decía: se pide manejar por menú un listado de enteros mediante una LSE. El programa debe ser capaz de crear una lista, insertar un nodo, enumerar los nodos, buscar un valor en la lista, imprimirla, eliminar un nodo con determinado valor y vaciarla.

Ahora se readapta pidiendo que la colección esté fuertemente ordenada y sea de nodos tipo [cadena, cadena, valor], donde valor es un número entero.

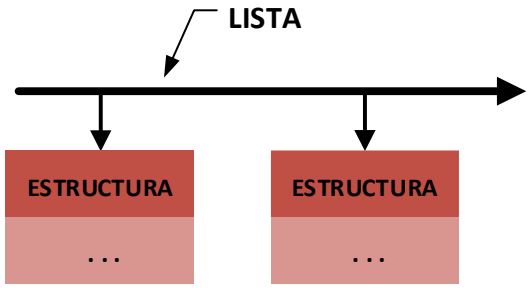


Ilustración 11. Un listado de estructuras

¿Cuál es el mejor contenedor? Cualquiera que no sea asociativo sirve al problema. ¿Por qué no los asociativos? Pues porque van ordenando los valores en el momento de entrar, estando diseñados por omisión para ordenamientos débiles. Prácticamente, cualquiera de los contenedores no asociativos sirve para la tarea, pero escoger apropiadamente el tipo correcta facilita enormemente la tarea. Por la naturaleza del problema el autor escogió la lista.

Entonces, el nodo será una lista cuyos miembros serán una estructura con algunos métodos, un par de miembros tipo cadena y uno entero como valor. La secuencia debe estar estrictamente ordenada de menor a mayor y permitirá un fácil acceso a sus nodos.

¿Cómo se maneja el contenedor secuencial en estos casos? Cuando los nodos son "inteligentes", como es el caso las estructuras y las clases, lo más lógico es dividir las operaciones en dos partes, delegando responsabilidades:

1. Los datos que maneja y las tareas que hace la estructura o clase. En este caso sabe entrarse un valor, cambiárselo y mostrarse en consola. Observar que un nodo, siendo parte de una secuencia, no puede borrarse por sí mismo.
2. La tarea propia del contenedor escogido, dictada por el menú, delega aparte las tareas complejas como es buscar el índice de un nodo o eliminar un nodo de la secuencia y el resto las hace el contenedor.

Declaraciones

El arreglo es una lista de estructuras. Los mensajes al usuario se tipifican y para poder emplear el algoritmo de ordenamiento es necesario codificar una función de comparación, porque el algoritmo *per se* no "sabe" como ordenar desde la secuencia compleja: lo pierde ese nivel extra de indirección que aportan las estructuras.

- Menú

```
#ifndef MENU_H_INCLUDED
#define MENU_H_INCLUDED

//  enumera lo que pide el menú
enum { Terminar, Insertar, Enumerar, Buscar, Imprimir, Eliminar, Vaciar };

#include <string>
typedef std::string Str;

//  un menú en consola
//  entrada: la información al usuario
//  salida: menú a consola
short menu( Str info );

#endif // MENU_H_INCLUDED
```

- Cola

```
#ifndef COLA_H_INCLUDED
#define COLA_H_INCLUDED

#include <string>
typedef std::string Str;

//  tipifica las respuestas
const Str ERR_N( "El nodo no est\240.\n\n" );
const Str ERR_Y( "El nodo ya est\240.\n\n" );
const Str ERR_V( "La lista est\240 vac\241a.\n\n" );
const Str HECHO( "Hecho.\n\n" );

//  un nodo de la lista
struct Nodo {
    // interfaz
    void mostrar() const;             // se muestra
    void entrar( bool dato = true ); // recibe/busca un nodo
    //  miembros:
    Str nombre;
    Str apellido;
    int valor;
```

```
};

#include <list>
typedef std::list<Nodo> Lst; // un listado de nodos

//  operador menor-que: función de ordenamiento
//  fuerte (lexicográfico)
bool menor( const Nodo & n1, const Nodo & n2 );

//  localiza el índice del dato en la lista
int buscar( const Lst * lista, const Nodo & dato );

//  elimina el dato que está en la posición pos
void borrar( Lst * lista, int pos );

#endif // COLA_H_INCLUDED
```

Definiciones

- Menú

```
#include "menu.h"
#include <iostream>
#include <cstdlib>

//  un menú en consola
short menu( Str info ) {
    Str conjunto = "0123456";
    Str opcion;
    Str::size_type idx;

    while ( true ) {
        system( "cls" );
        std::cout << info <<
        "Lista compleja\n"
        "1) Insertar un nodo\n"
        "2) Contar los nodos\n"
        "3) Comprobar si un valor est\240\n"
        "4) Mostrar la lista\n"
        "5) Eliminar un nodo\n"
        "6) Vaciar la lista\n"
        "0) Terminar\n"
        "Entre una opci\242n: ";
        ( std::cin >> opcion ).get();
        idx = conjunto.find( opcion );

        if ( Str::npos == idx ) {
            std::cerr << "\nError: opci\242n ilegal.\n\n";
            system( "pause" );
        } else {
            return idx;
        } // if-else
    } // while
}
```

- Cola: las dos funciones complejas se encargan de localizar el índice de un dato en la secuencia y de eliminar un nodo de la secuencia, donde aparece la función auxiliar **advance**. Se emplea el **goto** sin miedo.

```
#include "cola.h"

//  función de ordenamiento fuerte
```

```
bool menor( const Nodo & n1, const Nodo & n2 ) {
    if ( n1.apellido < n2.apellido ) return true;
    if ( n1.apellido == n2.apellido and n1.nombre < n2.nombre ) return true;
    return false;
}

#include <iostream>
//  entra un nodo nuevo
void Nodo::entrar( bool dato ) {
    std::cout << "\nEntre un nombre: ";
    std::getline( std::cin, nombre );
    std::cout << "Ahora su apellido: ";
    std::getline( std::cin, apellido );

    if ( true == dato ) {
        std::cout << "Su entero asociado = ";
        ( std::cin >> valor ).get();
    }
}

#include <iomanip>
//  muestra un nodo en consola
void Nodo::mostrar() const {
    std::cout << std::left      <<
    std::setw( 16 ) << apellido <<
    std::setw( 16 ) << nombre   << std::right <<
    std::setw(  4 ) << valor    << '\n';
}

//  localiza el índice del dato en la lista
int buscar( const Lst * lista, const Nodo & dato ) {
    int i = 0;

    for ( auto x: *lista ) {
        if ( x.nombre   == dato.nombre and
             x.apellido == dato.apellido ) {
            return i;
        } // if

        ++i;
    } // for

    return -1;
}

//  elimina el dato que está en la posición pos
void borrar( Lst * lista, int pos ) {
    Lst::const_iterator i = lista->begin();
    advance( i, pos ); // avanza el índice
    lista->erase( i );
}
```

Programa

Como siempre, el programa hace su defensa. Las operaciones directas son hechas aquí y solamente las dos más complejas se hacen en módulos aparte. Todos los algoritmos de ordenamiento pueden aceptar un tercer parámetro que gobierna su acción según deseo del programador; el ejemplo presenta cómo llevar a cabo esta técnica.

```
#include "menu.h"
#include "cola.h"
```

```
#include <iostream>
#include <algorithm>

int main() {
    using std::cout;
    using std::cin;
    using std::cerr;
    Str info( "Problema 18: manejando una secuencia ordenada\n" );
    Lst *pLst = new Lst; // una colección de datos
    Nodo dato;           // su estructura
    short opcion;        // la opción del menú
    int n;               // utilitario

    while ( true ) {
        opcion = menu( info );

        if ( Terminar == opcion ) {
            pLst->clear();
            delete pLst;
            return EXIT_SUCCESS;
        }

        switch ( opcion ) {
        case Insertar:
            dato.entrar();

            if ( -1 == buscar( pLst, dato ) ) { // busca por duplicados
                pLst->push_back( dato );        // si no está la cola lo añade...
                cout << HECHO;                  // ...y avisa...
            } else {
                cerr << ERR_Y; // otrosí, avisa que ya está
            }
            break;
        case Enumerar:
            n = pLst->size();

            if ( 0 == n )
                cerr << ERR_V;
            else
                1 == n
                    ? cout << "\nHay un nodo en la lista.\n\n"
                    : cout << "\nHay " << n << " nodos en la lista.\n\n";
            break;
        case Imprimir:
            if ( pLst->empty() )
                cerr << ERR_V;
            else {
                pLst->sort( menor );
                cout << '\n';

                for ( auto x: *pLst )
                    x.mostrar();

                cout << '\n';
            }
            break;
        case Buscar:
            if ( pLst->empty() )
                cerr << ERR_V;
            else {
                dato.entrar( false );
                 -1 == buscar( pLst, dato ) ? cerr << ERR_N : cerr << ERR_Y;
```

```
            }
            break;
        case Eliminar:
            if ( pLst->empty() ) {
                cerr << ERR_V;
                break;
            }

            dato.entrar( false );
            n = buscar( pLst, dato );

            if ( -1 == n )
                cerr << ERR_N;
            else {
                borrar( pLst, n );
                cout << HECHO;
            }
            break;
        case Vaciar:
            if ( pLst->empty() )
                cerr << ERR_V;
            else {
                char sn;
                cout << "\nOperaci\242n definitiva. \250Procedo? (s/n)";
                cin >> sn;

                if ( 'S' == toupper( sn ) ) {
                    pLst->clear();
                    cout << HECHO;
                } else
                    cout << "Abortado.\n\n";
            }
            break;
        }

        system( "pause" );
    }
}
```

Salida

```
Problema 18: manejando una secuencia ordenada
Lista compleja
1) Insertar un nodo
2) Contar los nodos
3) Comprobar si un valor está
4) Mostrar la lista
5) Eliminar un nodo
6) Vaciar la lista
0) Terminar
Entre una opción: 4

Aldorso         Santos          163
Almudena        Fabia           148
Avogadro        Carlos          161
Avogadro        Maricusa        158
Mancuso         Salvador        167
Montero         Daniel          170

Presione una tecla para continuar . . .
```

La eficiencia es del 29%

Una colección de algún miembro de la STL

El siguiente ejemplo trata de un arreglo de listas. Lo novedoso es la forma de ordenar ambos contenedores.

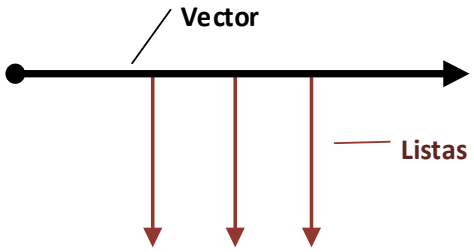


Ilustración 12. Un vector de listas

Programa 19: un vector de listas

Se muestra un par de técnicas interesantes:

1. La creación automatizada de listas en un ciclo, haciendo que la identificación de cada lista al ser creada la construya C++. Este es uno de esos casos en que se requiere de nombres de ficheros temporales y lo mejor es dejar que el programa mismo sea quien lo haga.

   ¿Cómo? En este sistema, en el archivo C:\MinGW32\include\stdio.h están declaradas dos directivas de pre procesamiento: **`TMP_MAX = 32767`**, que da la cantidad de nombres sin repetición que puede generar el sistema y

`L_tmpnam = 16`, que da la cantidad máxima de caracteres en cada nombre; y la función `tmpnam( char* )`, que produce un nombre temporal. En el caso que ocupa, este nombre es generado cada vez que se llama a la función.

2. Ordenamiento de mayor a menor con un algoritmo sobre el vector y en las listas con su método `sort`, ambos usando el mismo functor `greater` aplicado adecuadamente.

Declaración

Se ponen los prototipos usados por el algoritmo `for_each()`

```
#ifndef AUXILIAR_H_INCLUDED
#define AUXILIAR_H_INCLUDED

#include <list>
typedef std::list<int> Lst; // una lista

#include <vector>
typedef std::vector<Lst> Vct; // un vector de listas

//  operación de escritura para for_each:

//  listas originales
void imprimirOriginal( const Lst & l );

//  listas ordenadas z->a
void imprimirOrdenada( Lst & l );

#endif // AUXILIAR_H_INCLUDED
```

Definición

Como el vector es de listas, eso es lo que se imprime. Al método `sort` de las listas se le añade el functor `greater`.

```
#include "auxiliar.h"
#include "../deposito/print.h"
#include <iostream>

//  operación de escritura para for_each
//  listas originales
void imprimirOriginal( const Lst & l ) {
    MUESTRA( l );
    std::cout << '\n';
}

//  listas ordenadas z->a
void imprimirOrdenada( Lst & l ) {
    l.sort( std::greater<int>() );
    imprimirOriginal( l );
}
```

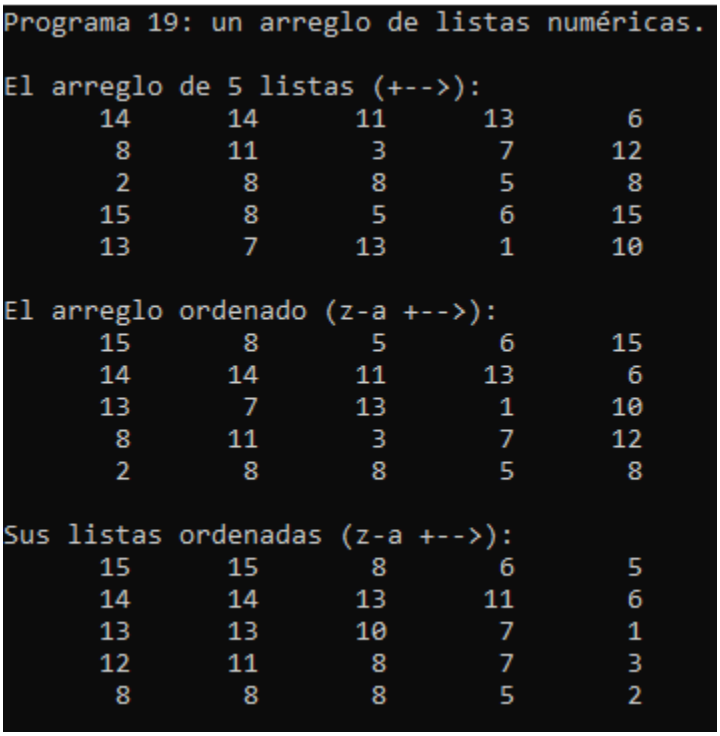
Programa

En cada ciclo se crea un nombre, se enlaza con una lista y ésta se puebla. Al usar el functor con el algoritmo, funciona; al usarlo con las listas, ¡también funciona!

```
#include "../deposito/rnd.h"
```

```
#include "auxiliar.h"
#include <iostream>
#include <cstdlib>
#include <ctime>

const int N = 5; // cantidad de elementos por lista
const int m = 1; // límites de los valores
const int M = N * 3;

int main() {
    std::cout << "Programa 19: un arreglo de listas num\202ricas.\n\n";
    srand( time( nullptr ) );

    //  crear el arreglo
    Vct ary;           // un vector STL
    char ID[L_tmpnam]; // nombres temporales automáticos

    //  poblar el arreglo de listas
    for ( int i = 0; i < N; ++i ) {
        tmpnam( ID );          // un nombre temporal
        Lst ID( N );           // una lista STL
        RELLENA_I( ID, m, M ); // la rellena
        ary.push_back( ID );   // la inserta en el vector
    } // for

    //  imprimir el arreglo
    std::cout << "El arreglo de " << N << " listas (+-->):\n";
    for_each( ary.begin(), ary.end(), imprimirOriginal );
    std::cout << '\n';

    //  imprimir el arreglo ordenado a->z
    std::cout << "El arreglo ordenado (z-a +-->):\n";
    sort( ary.begin(), ary.end(), std::greater<Lst>() );
    for_each( ary.begin(), ary.end(), imprimirOriginal );
    std::cout << '\n';

    //  imprimir las listas ordenadas
    std::cout << "Sus listas ordenadas (z-a +-->):\n";
    for_each( ary.begin(), ary.end(), imprimirOrdenada );
    std::cout << '\n';

    system( "pause" );
    return EXIT_SUCCESS;
}
```

Salida



Un ejemplo complejo

El problema 40 de (Introducción al C++, tomo 1, pág. 201) decía: la administración de nóminas es una importante tarea en el ámbito de la producción y los servicios. La CPA Santos García paga a su obrero agrícola un salario básico de 500 CUP por 40 h de trabajo semanal, más "tiempo y medio" por cada hora extra (no se consideran fracciones de hora.)

Programa 20

Se desea codificar un programa que tome las horas totales trabajadas en el mes por los obreros de la CPA, y emita una nómina mensual de pago, sabiendo que la empresa toma cada mes como de 4 semanas. La aplicación será manejada por menú. Hay restricciones:

- La nómina debe ir a consola y al archivo C:/temp/BD/nomina.txt
- La estructura será montada en la tienda libre.
- El usuario puede escoger entre subir los datos desde una nominilla que está en el archivo C:/temp/BD/plantilla.txt, o escribirlos manualmente.
- Nuevos obreros pueden ser insertados en cualquier momento.
- La nómina estará fuertemente ordenada

a) Para el desarrollo y puesta a punto de esta solución, usar los datos siguientes:

Trabajador	Horas
Salvador Mancuso	167
María Avogadro	158
Daniel Montero	170
Carlos Avogadro	161
Santos Aldorso	163
Fabia Almudena	148

b) El fichero CSV sería algo así como:

```
c:\temp\BD>type plantilla.txt
Salvador,Mancuso,167
Maricusa,Avogadro,158
Daniel,Montero,170
Santos,Aldorso,163
Carlos,Avogadro,161
Fabia,Almudena,148
c:\temp\BD>
```

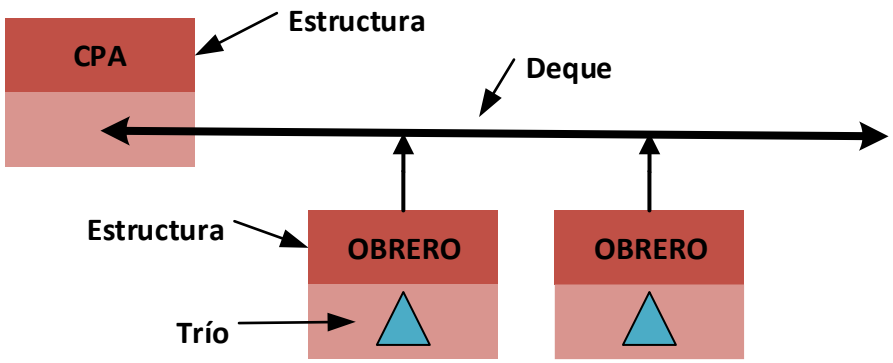


Ilustración 13. La estructura compleja

La ilustración muestra que la CPA es una estructura con una doble cola por composición. Cada nodo de la lista es otra estructura que guarda los datos de cada obrero en una tripleta. En muchas ocasiones se requiere del pase de dos parámetros (nombre y apellido del obrero agrícola) y se hace como un par.

Esta complejidad es un ejemplo claro de una máxima del Zen de Python: *lo práctico gana a lo puro* ya que muchos módulos no están muy bien formados y algunos son excesivamente largos. Por otra parte, permite una gran abstracción y EODA, bien vale la pena leer la solución atentamente.

Tabla 30. (simplificada) La doble cola

Constructor	Resultados
`deque<T> d`	Crea un deque vacío del tipo `T`
`deque<T> d(d1)`	Crea una copia de otro deque del mismo tipo `T`
`deque<T> d(n)`	Crea un deque de `n` elementos tipo `T`, puestos a cero
`deque<T> d(K.begin(), K.end())`	Crea un deque inicializado con los elementos de otro contenedor `K` en el rango `[begin, end)`
Métodos	**Resultados**
`n = d.size()`	El número actual de nodos
`d.clear()`	Hace al deque vacío
`b = d.empty()`	`true` si el deque está vacío

`d1 == d2`	`true` si ambos deques son iguales
`d1 != d2` `d1 not_eq d2`	`true` si ambos deques son distintos
`d1 < d2`	`true` si `d1 < d2`
`d1 > d2`	`true` si `d1 > d2`
`d1 <= d2`	`true` si `d1 <= d2`
`d1 >= d2`	`true` si `d1 >= d2`
`val = d.at(idx)`	Regresa el elemento de índice `idx`; lanza una excepción si se va de límites.
`val = d[idx]`	Regresa el elemento de índice `idx` y no chequea los límites; más rápido.
`val = d.front()`	Regresa el primer elemento, pero no chequea si existe.
`val = d.back()`	Regresa el último elemento, pero no chequea si existe.
`d.push_front(val)`	Encabeza al elemento `val`
`d.push_back(val)`	Encola al elemento `val`
`d.pop_front()`	Remueve al primer elemento, pero no lo devuelve.
`d.pop_back()`	Remueve el último elemento, pero no lo devuelve.
Asignación	**Resultados**
`d = d1`	Asigna todos los nodos de `d1` a `d`
`d.assign(n, val)`	Asigna `n` copias del elemento `val`
`d.swap(d1)`	Intercambia los datos de `d` con los de `d1`
`[idx = ]d.insert (pos, val)`	Inserta el elemento `val` en la posición `pos` y regresa su posición
`[idx = ]d.erase(pos)`	Remueve el elemento en la posición `pos` y regresa la del próximo
Repetidores	**Resultados**
`deque<T>::iterator i` `deque<T>::const_iterator i`	Un repetidor constante o no, de cabeza a cola.
`d.begin()`	La cabeza del deque.
`d.end()`	El final del deque.

Declaraciones

- Menú

```
#ifndef MENU_H_INCLUDED
#define MENU_H_INCLUDED

//  enumera lo que pide el menú
enum { Terminar, Insertar, Cargar, Contar, Checar, Editar, Crear, Borrar, Vaciar };

#include <string>
typedef std::string Str;

//  un menú en consola
//  input: la información al usuario
//  output: menú a consola
int menu( Str info );

#endif // MENU_H_INCLUDED
```

- CPA: debido a que la estructura Obrero es simple y corta, se unió al archivo cabecera de la nómina. Observar cómo se repartieron las tareas para los nodos y para la lista. Usualmente las tareas de los listados se resuelven con los métodos del contenedor empleado y sólo van aparte las acciones complejas, extensas o de ambos tipos, pero debido a los niveles extras de indirección, se pasan aparte todas las definiciones de la nómina.

```
#ifndef CPA_H_INCLUDED
#define CPA_H_INCLUDED

#include <string>
typedef std::string Str;

//  tipificación de mensajes
const Str HECHO( "\n\255Hecho!\n\n" );
const Str MSG_N( "\nNo est\240.\n\n" );
const Str MSG_V( "\nN\242mina vac\241a.\n\n" );
const Str MSG_Y( "\nYa est\240.\n\n" );
const Str ERR_L( "\nNo puedo leer el archivo.\n\n" );
const Str ERR_E( "\nNo puedo escribir al archivo.\n\n" );

#include <tuple>
//  nombre, apellido, horas
typedef std::tuple<Str, Str, int> Datos;

//  es una estructura que tiene el nombre completo
//  del obrero y las horas trabajadas en el mes.
//  sabe entrar sus datos y devolverlos
struct Obrero {
    bool yaEsta( Datos unDato );
    void entrarObrero( Str nombre, Str apellido, int horas );
    Datos sacarDatos();
    // miembro
    Datos dato;
};

#include <fstream>
#include <deque>
typedef std::deque<Obrero> Dek;
//  nombre, apellido
typedef std::pair<Str, Str> Par;

//  la CPA se encarga de manejar la nominilla en disco y de atender
//  los pedidos del menú. Tiene un listado de sus obreros y es capaz
//  de calcularles el salario y poner la nómina a consola y a disco
struct CPA {
    void escribirNomina( std::ostream & escribe );
    void leerPlantilla( std::istream & lee );
    void adicionarObrero( Par dat );
    void editarObrero( Par dat );
    void borrarObrero( Par dat );
    double verDatos( Datos & d, std::ostream & escribe ) const;
    bool datoEsta( Par dat ) const;
    // miembro
    Dek nomina;
};

#endif // CPA_INCLUDED
```

- Funciones auxiliares que van fuera de las estructuras

```
#ifndef EXTRAS_H_INCLUDED
#define EXTRAS_H_INCLUDED

#include "cpa.h"
//  función de ordenamiento fuerte: trabaja sobre
//  los nodos del contenedor, añade un nivel extra
//  de indirección
bool menor( const Obrero & n1, const Obrero & n2 );
```

```
//  función auxiliar: devuelve el salario devengado
float calcSalario( int hs );

//  función auxiliar para cargar nombre y apellido
Par cargarDato();

#endif // EXTRAS_H_INCLUDED
```

Definiciones

- Menú

```
#include "menu.h"
#include <iostream>
#include <cstdlib>

//  un menú en consola
int menu( Str info ) {
    //  variables
    Str conjunto = "012345678";
    Str opcion;
    Str::size_type idx;

    while ( true ) {
        system( "cls" );
        std::cout << info <<
        "   MEN\351\n"
        "--------------------------\n"
        "1) Insertar registros\n"
        "2) Cargar una nominilla\n"
        "3) Contar los registros\n"
        "4) Verificar un registro\n"
        "5) Editar un registro\n"
        "6) Crear la n\242mina\n"
        "7) Borrar un registro\n"
        "8) Vaciar la n\242mina\n"
        "0) Terminar\n"
        "--------------------------\n"
        "Entre una opci\242n: ";
        std::getline( std::cin, opcion );
        idx = conjunto.find( opcion );

        if ( Str::npos == idx ) {
            std::cerr << "\nError: opci\242n ilegal.\n\n";
            system( "pause" );
        } else {
            return idx;
        } // if-else
    } // while
}
```

- CPA: el problema es extenso y complejo y eso lo refleja el código. Sin embargo, todo ya ha sido usado antes. Aquí el método **`escribirNomina(ostream &escribe)`** recibe como parámetro un flujo de datos y ahora puede ser llamado a mostrarse en cualquier periférico. El programa lo llama dos veces: primero pasándole un flujo de escritura a consola y luego, pasándole un flujo de escritura a disco.

```
#include "cpa.h"

//  los obreros

//  entra las variables a la tupla
```

```
void Obrero::entrarObrero( Str nombre, Str apellido, int horas ) {
    dato = std::make_tuple( nombre, apellido, horas );
}

//  devuelve el dato
Datos Obrero::sacarDatos() { return dato; }

//  verifica si un dato ya está en la nómina;
//  sólo toma las variables necesarias
bool Obrero::yaEsta( Datos unDato ) {
    Str nombre     = std::get<0>( dato );
    Str apellido   = std::get<1>( dato );
    Str unNombre   = std::get<0>( unDato );
    Str unApellido = std::get<1>( unDato );
    return ( nombre == unNombre and apellido == unApellido );
}

//  la CPA

//  checa si el dato ya está
bool CPA::datoEsta( Par dat ) const {
    Obrero tmp;
    tmp.entrarObrero( dat.first, dat.second, 0 );

    for ( auto x : nomina )
        if ( x.yaEsta( tmp.dato ) )
            return true;

    return false;
}

#include "extras.h"
#include <ctime>
#include <cstdlib>
#include <iostream>
#include <algorithm>
#include <iomanip>
using std::setw;
using std::ostream;

//  calcula el salario de un obrero y pone la línea en la nómina
double CPA::verDatos( Datos & d, ostream & Escribiendo ) const {
    Str nom, ape;
    int hs;
    std::tie( nom, ape, hs ) = d;
    double sal = calcSalario( hs );
    Escribiendo << std::setfill( '.' ) << std::fixed  << std::left  <<
                   setw ( 20 ) << ( ape + ", " + nom ) << std::right <<
                   setw ( 5 )  << hs  << std::setprecision( 2 )      <<
                   setw ( 15 ) << sal << "  _______________\n";
    return sal;
}

//  escribe la nómina a un flujo de salida;
//  llama a cada obrero de la nómina
void CPA::escribirNomina( ostream & Escribiendo ) {
    if ( Escribiendo.fail() ) throw ERR_E;

    sort( nomina.begin(), nomina.end(), menor );
    time_v laFecha = time( 0 );    // obteniendo la fecha completa
    Str fecha = ctime( &laFecha ); // la fecha y la hora
    //  encabezado
```

```
    system( "cls" );
    Escribiendo <<
    "CPA S. Garc\241a - N\242mina" << '\n' << "Fecha: " << fecha << '\n' << std::left <<
    setw( 18 ) << "Obrero" << std::right << setw( 7 ) << "Horas" << setw( 16 ) << "Salario\n";
    //  visitar la nómina
    Obrero tmp;
    Dek::const_iterator i;
    Dek::const_iterator end( nomina.end() );
    double total = 0; // total nómina

    for ( i = nomina.begin(); i != end; ++i ) {
        tmp = *i;
        total += verDatos( tmp.dato, Escribiendo ); // a consola o a disco
    } // for

    Escribiendo
    << std::left  << setw( 30 ) << "Total"
    << std::right << setw( 10 ) << total << "$ Firma\n\n";
}

using std::cerr;
using std::cout;

void CPA::borrarObrero( Par dat ) {
    if ( datoEsta( dat ) ) {
        //  visitar la nómina
        Obrero tmp, proxy;
        tmp.entrarObrero( dat.first, dat.second, 0 );
        Dek::iterator i;
        Dek::iterator end( nomina.end() );

        for ( i = nomina.begin(); i != end; ++i ) {
            proxy = *i; // bajar un nivel de indirección

            if ( proxy.yaEsta( tmp.dato ) ) {
                nomina.erase( i );
                cout << HECHO;
                return;
            } // if
        } // for
    } else {
        cerr << MSG_N;
    } // if-else
}

using std::cin;

//  añade un obrero a la nómina
void CPA::adicionarObrero( Par dat ) {
    if ( datoEsta( dat ) ) {
        cerr << MSG_Y;
    } else {
        Obrero tmp;
        int hs;
        cout << "\250Horas? ";
        ( cin >> hs ).get();
        tmp.dato = std::make_tuple( dat.first, dat.second, hs );
        nomina.push_back( tmp );
        cout << HECHO;
    } // if-else
}
```

```
void CPA::editarObrero( Par dat ) {
    if ( datoEsta( dat ) ) {
        //  visitar la nómina
        Obrero tmp, proxy;
        tmp.entrarObrero( dat.first, dat.second, 0 );
        Dek::iterator i;
        Dek::iterator end( nomina.end() );

        for ( i = nomina.begin(); i != end; ++i ) {
            proxy = *i; // bajar un nivel de indirección

            if ( proxy.yaEsta( tmp.dato ) ) {
                int hs;
                cout << "\250Nuevas horas? ";
                ( cin >> hs ).get();
                nomina.erase( i );
                tmp.dato = std::make_tuple( dat.first, dat.second, hs );
                nomina.push_back( tmp );
                cout << HECHO;
                return;
            } // if
        } // for
    } else {
        cerr << MSG_N;
    } // if-else
}

#include <cstring>
#include <fstream>
using std::getline;

void CPA::leerPlantilla( std::istream & Leyendo ) {
    if ( Leyendo.fail() ) throw ERR_L;

    Obrero tmp;
    Str nom, ape, hs;

    while ( Leyendo.good() ) {
        //  construyendo un registro
        getline( Leyendo, nom, ',' );
        getline( Leyendo, ape, ',' );
        getline( Leyendo, hs, '\n' );
        tmp.entrarObrero( nom, ape, atoi( hs.d_str() ) );
        nomina.push_back( tmp );
    } // while
}
```

- Funciones auxiliares que van fuera de las estructuras

```
#include "extras.h"

//  función de ordenamiento fuerte: trabaja sobre los nodos
//  del contenedor, añade un nivel extra de indirección
bool menor( const Obrero & n1, const Obrero & n2 ) {
    // hay que desempacar los valores a comparar
    // primer nodo
    Str nom1 = std::get<0>( n1.dato );
    Str ape1 = std::get<1>( n1.dato );
    // segundo nodo
    Str nom2 = std::get<0>( n2.dato );
    Str ape2 = std::get<1>( n2.dato );

    if ( ape1 < ape2 ) return true;
```

```
        else if ( ape1 == ape2 and nom1 < nom2 ) return true;
        else return false;
    }

    const int HORAS    =  40;  // a la semana
    const int SEMANAS  =   4;  // 4 semanas por mes
    const int PAGO     = 500;  // pesos
    const float BASICO = static_cast<float>( PAGO ) / HORAS;
    const float EXTRA  =   1.5 * BASICO;

    //  función auxiliar: devuelve el salario devengado
    float calcSalario( int hs ) {
        float salario;

        if ( hs > HORAS * SEMANAS ) {
            salario = ( hs - HORAS * SEMANAS ) * EXTRA;
            salario += HORAS * SEMANAS * BASICO;
        } else {
            salario = hs * BASICO;
        } // if-else

        return salario;
    }

    #include <iostream>

    //  función auxiliar para cargar nombre y apellido
    Par cargarDato() {
        Str nom, ape;
        std::cout << "\250Nombre? ";
        std::getline( std::cin, nom );
        std::cout << "\250Apellido? ";
        std::getline( std::cin, ape );
        return std::make_pair( nom, ape );
    }
```

Programa

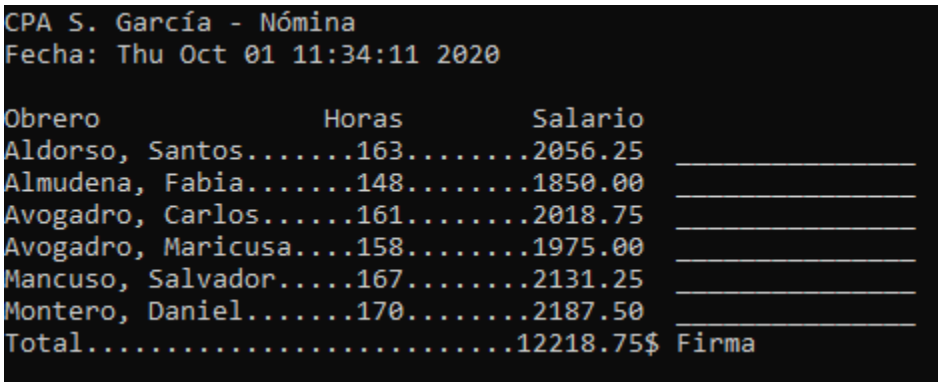
```
#include "menu.h"
#include "cpa.h"
#include "extras.h"
#include <iostream>
#include <cstdlib>

int main() {
    Str info( "Programa 20: n\242mina de la CPA Santos Garc\241a.\n\n" );
    std::ofstream aDisco( "C:/temp/BD/nomina.txt" );
    std::ifstream deDisco( "C:/temp/BD/plantilla.txt" );
    CPA * pLista = new CPA;
    int cant, opcion;
    char sn;

    while ( true ) {
        opcion = menu( info );

        if ( Terminar == opcion ) {
            delete pLista;
            return EXIT_SUCCESS;
        } else
            cant = pLista->nomina.size();
```

```
switch ( opcion ) {
case Contar:
    if ( 0 == cant )
        std::cerr << MSG_V;
    else
        1 == cant
           ? std::cout << "\nUn registro.\n\n"
           : std::cout << '\n' << cant << " registros.\n\n";
    break;
case Insertar:
    pLista->adicionarObrero( cargarDato() );
    break;
case Crear:
    try {
        pLista->escribirNomina( aDisco );
        pLista->escribirNomina( std::cout );
    } catch ( Str ) {
        std::cerr << ERR_E;
    } // try-catch
    break;
case Checar:
    0 == cant
       ? std::cout << MSG_V
       : pLista->datoEsta( cargarDato() )
           ? std::cout << MSG_Y
           : std::cout << MSG_N;
    break;
case Editar:
    if ( 0 == cant )
        std::cout << MSG_V;
    else
        pLista->editarObrero( cargarDato() );
    break;
case Vaciar:
    if ( 0 == cant )
        std::cout << MSG_V;
    else {
        std::cout << "\nPara vaciar la n\242mina.\n\250Prosigo? (s/n) ";
        ( std::cin >> sn ).get();

        if ( 'S' == toupper( sn ) ) {
            pLista->nomina.clear();
            std::cout << HECHO;
        } else std::cerr << "\nAcci\242n abortada.\n\n";
    } // if-else
    break;
case Borrar:
    if ( 0 == cant )
        std::cout << MSG_V;
    else
        pLista->borrarObrero( cargarDato() );
    break;
case Cargar:
    try {
        pLista->leerPlantilla( deDisco );
        std::cout << HECHO;
    } catch ( Str ) {
        std::cerr << ERR_L;
    } // try-catch
    break;
} // switch
```

```
        system( "pause" );
    } // while
}
```

Una posible salida

- Una vez escogida la opción número dos, se carga la plantilla del disco. Entonces, al escoger la opción seis:



PROBLEMAS PROPUESTOS

Buscando en un viejo manual de (Introducción a la programación en Turbo C) que el autor preparó y usó para sus clases de computación en el antiguo ISCAB (hoy UDG) hace ya…años, encontró y tomó algunos de los ejercicios, los replanteó y ahora se los propone.

Además, hay ejercicios de STL puros y ejercicios que resuelven ejemplos dados en el Tomo 1 o en este mismo tomo. Y recuerde, para cada uno de estos problemas —algunos triviales— hay más de una solución.

La restricción general es que todos ellos deben ser resueltos utilizando lo aprendido con la STL, Si se trabaja con estructuras o plantillas, documentarlas con Doxygen o si no lo tiene, con una aplicación similar.

1. El único objetivo de este ejercicio es que se codifique una plantilla para intercambiar valores de tipos nativos, la cual debería llamarse **`INTERCAMBIAR`**. La salida debería verse así:

```
Ejercicio 1: intercambio de valores.

Dos enteros:
x = 4
y = 9
Después:
x = 9  y = 4

Dos valores reales:
x = 3.21
y = 7.89
Después:
x = 7.89  y = 3.21

Dos palabras:
x = yo
y = soy
Después:
x = soy  y = yo
```

2. La biblioteca **`<cmath>`** no ofrece la función circular secante, pero se conoce que la secante es la inversa del coseno. Calcular la secante de un ángulo dado en grados sexagesimales en el periodo entre 0 y 360 grados. Como hay una división implicada, que puede ser por cero, proteger el código mediante límites prohibidos, usando $\varepsilon \pm ½$ grado. La salida debería verse así:

```
Ejercicio 2: cálculo de la secante
con un error de ± 0.5 grados alrededor de 90

Valor entre 0 y 360 grados sexagesimales: 60
Secante = 2

¿Una vez más? (s/n)> n
```

3. Cuando en Cuba son las siete de la mañana, en Moscú son aproximadamente las tres y media de la tarde. Dada la hora de Cuba, calcular la de Moscú y decir si es del mismo día o del día siguiente. Crear una estructura apropiada que maneje los tres datos (hora, minuto y meridiano) como un todo. La salida debería verse así:

```
Ejercicio 3: convierte la hora de Cuba a la de Moscú.

Entrar la hora de Cuba, luego los minutos
y por último el meridiano:

Hora: 7
Minutos: 50
¿(A)m o (P)m? a

Hora de Moscú: 4:20 pm del mismo día.
```

4. El chequeo de un dato se usa en las rutinas de transmisión: a cada uno se le aplica un checksum (o suma de verificación, en español) y se envían él y su chequeo juntos. Al llegar a destino se prueba su chequeo real contra el que mandaron y si casan, el dato pasó sin error. Si hay una discrepancia se pueden rechazar los datos o pedir una retransmisión. Uno de los tantos métodos de su cálculo consiste en sumar los códigos ASCII de cada carácter en el dato y calcular el módulo 256 de esa suma. Sólo se usan 7 bits para los símbolos y el 8vo es un bit de paridad, haciendo que una letra del español posea un código ASCII negativo. Codificar un programa que pida un dato (palabra o número) y muestre este Checksum en acción. La salida debería verse así:

```
Ejercicio 4: cálculo del Checksum

Entre un dato (un número o una frase)
-> Cañón

      Cód. Suma
C:     67    67
a:     97   164
ñ:    -92    72
ó:    -94   -22
n:    110    88

Checksum (88 % 256) = 88

El Checksum de "Cañón" es = 0X58
que en octal es 0130 y en decimal, 88

¿Una vez más? (s/n)> n
```

5. El ejercicio anterior sugiere una alternativa para resolver el programa 4. Aplicarla a una nueva solución. La salida no se alteraría. Para comodidad, he aquí los valores para este equipo, pero si los del suyo difieren, búsquelos.

Tabla 31. Códigos del ASCII de 7 bits para el español

Letra	Código	Letra	Código	Letra	Código	Letra	Código
á	-96	í	-95	ú	-93	ñ	-92
Á	-75	Í	-42	Ú	-23	Ñ	-91
é	-126	ó	-94	ü	-127	¿	-88
É	-112	Ó	-32	Ü	-102	¡	-83

¿Cuál es la apreciación del lector? ¿Así es mejor, peor o no hay diferencia alguna de una versión sobre la otra? ¿Pueden estos códigos sustituir a los octales en la cadena de petición de **`cout`**? Explicar la respuesta.

```
Ejercicio 5: cambio de tamaño

(M)ayúsculas, mi(N)úsculas, (O)ración, (T)erminar.
Entre una opción: m

   Entre una frase: una puñetera tarde de mayo
Frase a Mayúsculas: UNA PUÑETERA TARDE DE MAYO
```

6. El método de Montecarlo consiste en utilizar las leyes del azar para modelar fenómenos estadísticos, sustituyendo los valores extraídos de experimentos reales por valores generados aleatoriamente. Gran consumidor de tiempo, solamente vino a tomar la importancia que merece en la era de los grandes superordenadores; por ejemplo, en 2011 se le conocían ¡10 billones de cifras! Para calcular aproximadamente el valor de π se realiza un experimento llamado *La aguja de Bufón* —por Georges Leclerc, conde de Bufón, matemático francés del s XVIII. Ver en (Wikipedia en español)— que trata del lanzamiento de una aguja sobre un papel cuadriculado. Como el experimentador puede definir de antemano la longitud de la aguja y el lado de las cuadrículas del papel, puede ajustarlos para que coincidan con la teoría. El planteamiento es:

Sean N la cantidad de lanzamientos de la aguja sobre el papel y A la cantidad de éxitos, medidos como la generación de **`true`** de entre los valores **`(false:true)`** Si el lado de la cuadrícula es mayor que la longitud de la aguja, se puede demostrar matemáticamente que la probabilidad del éxito disminuye proporcionalmente al cociente entre ambas medidas. Tomando L como la longitud de la aguja, C como el lado de la cuadrícula, ajustando $\left(\frac{L}{C}\right) \approx \pi$ y tomando la probabilidad de éxitos $p = \frac{A}{N}$ entonces $\pi \approx 2\frac{AL}{NC}$. Si se toma un arreglo booleano de tamaño N, se le generan valores **`true/false`** al azar y se cuenta el número de veces que salió **`true`**, al dividirlo entre N da la probabilidad para ese pase.

Un pase constará de 5000 lanzamientos y el experimento se repetirá hasta tener el valor de Pi dentro del error permisible del ± 0.0001%. La salida debería verse así:

```
Ejercicio 6: generando el valor de Pi

Cada pase son 5000 réplicas y el cálculo
se hace hasta tener un error de ± 0.0001%.

Pase 2

En 2 pases:
  Valor base de Pi  = 3.141596
  Valor generado Pi = 3.141353
  Error cometido    = -0.000077 %
```

7. El programa 13 pedía organizar una búsqueda visual sobre un arreglo de 80 valores generados al azar, y una vez que el valor estaba, mostrar sus posiciones. Para ello se visitaba el vector y se encolaba la posición de la clave. Pero, una vez hallada y registrada, ¿para qué seguir con la visita? Eso es ineficiente e ineficaz. Sólo hay que pensar en el caso de una búsqueda lineal en un millón o más de registros, con un valor cerca de los inicios…Un refinamiento imprescindible aprovecha las posiciones contiguas. Codificarlo. La salida no debería sufrir alteraciones.

```
Ejercicio 7: prueba de búsqueda; si el valor está, dice cuántas veces y dónde.
Arreglo:

     2       2       3       8       9      11      14      15      17      18
    21      24      26      27      27      28      28      29      30      31
    31      31      32      32      32      32      33      33      34      35
    35      36      38      42      43      43      43      44      44      45
    46      47      48      49      53      59      59      59      59      61
    61      61      63      64      65      66      66      67      68      68
    68      72      73      74      76      79      79      79      83      86
    86      87      89      91      95      98      99     100     100     100

T para terminar ¿Clave? 100
100 está 3 veces a partir de la posición 78

T para terminar ¿Clave? 45
45 está una vez en la posición 40

T para terminar ¿Clave? 2
2 está 2 veces a partir de la posición 1

T para terminar ¿Clave? t
```

8. Codificar un programa con un mapa y aplicarlo a las 19 preposiciones del español de modo que el usuario seleccione todas o una de ellas y el programa devuelva su significado. Ayudarse con un diccionario. La salida debiera verse así:

```
Ejercicio 8: las 19 preposiciones del español.

(U)na, (T)odas, (F)inalizar. Entre una opción > u
Preposición: contra

Prep.   Significado
--------------------------------------------------------------------------------
contra   Enfrentamiento, apoyatura, entrega a cambio de algo.
--------------------------------------------------------------------------------
```

9. Codificar una lista de animales salvajes. Introducir un gato, un oso y un impala. Cambiar el gato por un puma. Insertar una cebra al inicio, un león delante del oso, un castor al final y ordenar la lista. Mostrar las operaciones. La salida debiera verse así:

```
Ejercicio 9: un listado de animales.

Lista original:  gato, oso, impala
Lista cambiada:  puma, oso, impala
Lista aumentada: cebra, puma, león, oso, impala, castor
Lista ordenada:  castor, cebra, impala, león, oso, puma
```

10. Tal y como la describe (Wikipedia en español), la siega de Eratóstenes es una manera sencilla de hallar todos los números primos menores o iguales que un número dado. Se basa en confeccionar una lista de todos los números naturales desde el 2 hasta ese número y tachar repetidamente los múltiplos de los números primos ya descubiertos. Una manera de codificar esta acción es hacer un vector de booleanos puesto a **`true`** y si el número no es primo, ponerlo a **`false`**. Hacerlo así y mostrarlos.

```
Ejercicio 10: la siega de Eratóstenes.

¿Cuántos números? 45

2 3 5 7 11 13 17 19 23 29 31 37 41 43

Son 14 números primos.
```

11. Codificar un programa que mapee los meses con sus días y responda para un mes dado la cantidad de días que tiene. Para no complicar las cosas, febrero trae 28 días. La salida debería verse así:

```
Ejercio 11: los meses.

Para salir entre un nombre cualquiera.
Entre un mes: diciembre
diciembre tiene 31 días.

Para salir entre un nombre cualquiera.
Entre un mes: enero
enero tiene 31 días.

Para salir entre un nombre cualquiera.
Entre un mes: m
```

12. Crear dos arreglos numéricos con 10 números al azar cada uno. Sume el segundo al primero, doble los valores del segundo y divida el segundo entre el primero. Liste los valores en cada operación. Use un **`valarray`**. La salida debería verse así:

```
Ejercicio 12: valarrays.

Valarray v1: 13, 35, 3, 17, 34, 8, 3, 31, 6, 47
Valarray v2: 67, 68, 77, 59, 73, 60, 57, 83, 72, 81

Sumando v2 a v1
Valarray v1: 80, 103, 80, 76, 107, 68, 60, 114, 78, 128

luego multiplicando a v2 por 2
Valarray v2: 134, 136, 154, 118, 146, 120, 114, 166, 144, 162

y por último, sacando v2 / v1
Valarray v3: 1.68, 1.32, 1.93, 1.55, 1.36, 1.76, 1.90, 1.46, 1.85, 1.27
```

13. Crear un vector con 10 números tomados al azar de entre 1 y 50, imprimirlos, borrar su segunda mitad e imprimirlos otra vez. La salida, aunque varía en cada corrida, debería verse así:

```
Ejercicio 13: un vector.

El vector: 3, 3, 6, 8, 13, 17, 31, 34, 35, 47
El vector: 3, 3, 6, 8, 13
```

14. La factorial puede calcularse con un modo del algoritmo **`accumulate`**, que acepta un cuarto parámetro, un functor binario. Cuando el número es 1 y el functor es **`multiplies<T>()`**, se hace el cálculo, pero no va más allá de 12 porque

se acumulan las multiplicaciones. La sintaxis es **`accumulate(K.begin(), K.end(), número, functor)`**. Crear un programa que haga ese cálculo. La salida debería verse así:

```
Ejercicio 14: la factorial

(no más de 12) ¿Número? 8

La factorial de 8 es 40320
```

15. Codificar un programa que muestre un vector al derecho y al revés. La salida debería verse así:

```
Ejercicio 15: la reversa.

Vector:
  1 2 3 4 5 6 7 8 9 10

Vector revertido:
  10 9 8 7 6 5 4 3 2 1
```

16. Un **`heap`** en el contexto de ordenamiento es utilizado como una forma particular de ordenar elementos y es usado por el algoritmo **`sort_heap`**. Un **`heap`** puede ser considerado un árbol binario que es implementado como una colección secuencial. El **`heap`** máximo tiene dos propiedades: el primer elemento es siempre el más grande y se puede agregar o quitar un elemento en tiempo logarítmico.

Cuando se *popea* un **`heap`**, se toma el primer elemento y se pone al final. Cuando se *pushea*, se toma el último y se pone adelante. Codificar un programa que haga las operaciones mencionadas con un **`heap`** sobre dos vectores. La salida debería verse así:

```
Problena 16: operaciones con el heap.

Crear 2 vectores y hacerlos sendos heaps.
4 2 3 1
4 3 2 1

Popear ambos vectores.
3 2 1 4
3 1 2 4

Pushear 9 y 7 respectivamente.
9 3 1 4 2
7 3 2 4 1

Y ahora, ordenarlos.
1 2 4 3 9
1 2 4 3 7
```

17. Hacer un programa que maneje una pila de enteros, en la línea del programa para listas, adaptando y simplificando el menú a la posibilidad real de esta especializada estructura de datos que es la **`<stack>`**. La interfaz brindada es todo lo que se necesita para que haga correctamente su trabajo. La salida debería verse así:

```
Ejercicio 17: manejo de una pila de enteros.

(A)pilar, (C)ontar nodos, (D)esapilar, (V)er tope
(T)erminar. Entre un comando > a

Entre un entero: 123
123 fue apilado.

Presione una tecla para continuar . . .
```

18. Hacer un programa que maneje una cola de enteros, en la línea del programa para listas, adaptando y simplificando el menú a la posibilidad real de esta especializada estructura de datos que es la **<queue>**. La interfaz brindada es todo lo que se necesita para que haga correctamente su trabajo. La salida debería verse así:

```
Ejercicio 18: manejo de una cola de enteros.

(E)ncolar, (C)contar nodos, (D)esencolar, (V)er puntas
(T)erminar. Entre un comando > v

Cabeza = 123; Cola = 132

Presione una tecla para continuar . . .
```

19. El problema 18 de (Introducción al C++, tomo 1, pág. 96), decía: el teorema de Pitágoras es la proposición más conocida entre las que tienen nombre propio en las matemáticas y establece que: "*en todo triángulo rectángulo el cuadrado del valor la hipotenusa es igual a la suma de los cuadrados de los valores de ambos catetos*", donde la hipotenusa es el lado mayor de todos. Codificar un programa que tome tres lados enteros y decida si con ellos se puede construir un triángulo, y si ese triángulo es rectángulo. La salida se vería así:

```
Ejercicio 19: clasifica un triángulo.

Lado 1 = 17
Lado 2 = 8
Lado 3 = 15

Se puede hacer un triángulo rectángulo.
  Hipotenusa = 17
Cateto mayor = 15
Cateto menor = 8
```

20. El problema 37 de (Introducción al C++, tomo 1, pág. 177) decía: para formar una bibliografía, cree una lista ordenada de cadenas C++ y manéjela por menú…Pero una buena elección del tipo de dato, según sea el problema, simplifica y flexibiliza la solución. La STL brinda la bolsa como una secuencia de pares formada por un árbol ordenado por diseño, con claves repetidas: ahora no hay que identificar a nadie. Aprovechando sus bondades, se creará un listado de bibliografías manejado por menú que permita al usuario manejar más a fondo su bibliografía.

En aras de conseguir el objetivo didáctico de mostrar cómo se simplifica e incluso, se flexibiliza una solución con un tipo de dato complejo, pero adecuado —y como se muestra en la bibliografía— ahora sólo se entrará el primer apellido del autor junto con la inicial de su nombre, separados por una coma. Los datos de prueba entrados en este mismo orden son: Halterman, R. *C++ programming*; Jaeschke, R. *Void Pointers*; Eckel, B. *Pensar en C++*; Jensen, T. *Tutorial en C*; Bellas Permuy, F. *El Lenguaje C++*; Gottfried, B. *Programación Pascal*; Jaeschke, R. *Header Design*; Deitel & Deitel. *Cómo programar en C/C++*; Galliano S. *C++ Primer*.

```
Ejercicio 20: manejo de una biblioteca.
 BIBLIOTECA
 1) Insertar una o varias obras.
 2) Mostrar la biblioteca.
 3) Contar las obras.
 4) Buscar un nombre.
 5) Buscar una obra.
 6) Eliminar una obra.
 7) Eliminar un nombre.
 8) Vaciar la biblioteca.
 9) Ayuda.
 0) Terminar.
Entre una opción: 2
Autor                Título
Deitel & Deitel.....Cómo programar en C++
Eckel, B............Pensar en C++
Halterman, R........C++ programming
Jaeschke, R.........Void pointers
Jaeschke, R.........Header design
```

21. El problema 24 de (Introducción al C++, tomo 1, pág. 125) decía: en el pañol de la universidad pedagógica Blas Roca, donde se guardan las herramientas de la carrera Educación Laboral, existen las siguientes herramientas automáticas y su precio por unidad: Sierra de calar a $6.80, Sierra de trocear a $8.99, Atornillador a $3.55 y Lijadora plana a $11.22 Mediante el llenado de un modelo, un docente pide al pañol la cantidad de estas herramientas que necesita para una clase práctica y el pañolero las registra y además el costo del préstamo, quién hizo el pedido y en qué fecha lo hizo. Diseñar una estructura adecuada al problema y escribir un pequeño programa para probarla:

```
Problema 21: Pedido del pañol del Blas Roca.

Stock del pañol:
Herramienta        Precio
Sierra de calar.......6.80
Sierra de trocear.....8.99
Atornillador..........3.55
Lijadora plana.......11.22

Nombre completo del docente: Salvador Aldorso
Fecha del pedido (día/mes/año): 13/9/2020
   Sierra de calar, cantidad? 6
 Sierra de trocear, cantidad? 6
      Atornillador, cantidad? 15
    Lijadora plana, cantidad? 3

Hoja de pedidos.
Salvador Aldorso - Fecha: 13/9/2020
Herramienta         Precio    Cant      Costo
Sierra de calar       6.80       6       40.80
Sierra de trocear     8.99       6       53.94
Atornillador          3.55      15       53.25
Lijadora plana       11.22       3       33.66
Costo total:.............................181.65$
```

22. El programa 9 (en la página 82) aportó una solución a la codificación de la calculadora HEXelon MAX 6. Añadir una función que cubra la conversión de lecturas de los glucómetros para milimoles por litro (mmol/l)/miligramos por decilitros(mg/dl). La fórmula de conversión (Wikipedia en español) es mg/dl→mmol/l: $X = (Y - 3.75)/17.5$ Sin embargo, una base de datos con los valores y un vector que los cargue al inicio sería un diseño más eficiente. Rehacerlo con este esquema. Tener presente que hay cuatro tipos de cálculos:

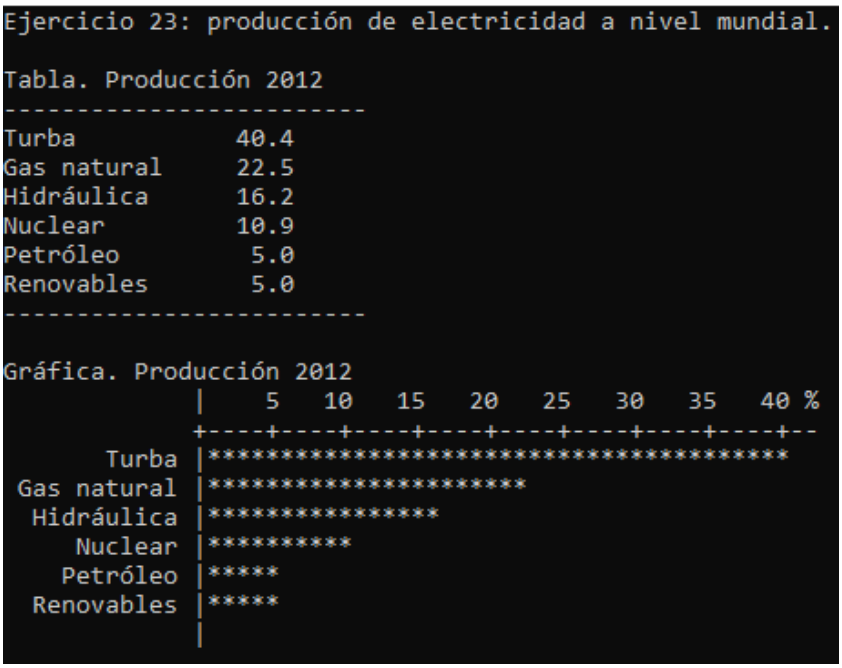
a) El de temperaturas que usa un factor de conversión (32) y uno de desplazamiento (1.8)
b) El de ángulos, que al calcular los grados debe eliminar las vueltas superfluas.
c) El de diabetes, que usa un factor de conversión (17.5), uno de desplazamiento (3.75) y además tiene salida en una sola dirección: de mmol/l a mg/dl
d) El resto que hace todas las conversiones con un solo factor.

```
Ejercicio 22: calculadora de conversiones.

   MENÚ
1) Galón USA y Litro
2) Pie cúbico y metro cúbico
3) Libra y Kilogramo
4) Acre y Hectárea
5) Milla y Kilómetro
6) Grados Fahrenheit y Celsius
7) Radián y grado sexagesimal
8) Miligramos por decilitro a milimoles por litro
0) Terminar

Entre una opción: 8

¿Valor? 105

105 mmol/l equivalen a 5.8 mg/dl

Presione una tecla para continuar . . .
```

23. El programa 17 (en la página 118) presenta un medidor de glucosa. Codificar un programa que presenta la gráfica de su historial. Emplear el historial de dicho programa. La salida debiera verse así:

```
Ejercicio 24: gráfica de un historial.

Valores de glucosa en sangre.
       |    5   10   15 mmol/ml
       +----+----+----+--
Aug  8 |     *  6.6
Aug 17 |      *  7.8
Aug 23 |    *  5.7
Sep  1 |       *  9.7
Sep  6 |        *  10
Sep 12 |       *  9.4
Sep 23 |      *  8.7
Oct 10 |      *  8.4
Oct 21 |      *  8
       |
```

24. El problema 30 de (Introducción al C++, tomo 1, pág. 162) decía: ¿Se puede graficar en modo consola? La producción de electricidad a nivel mundial se calcula mediante seis acápites que, según (Wikipedia en español), para el año 2012 fueron: Turba (40.4%), Gas natural (22.5%), Hidráulica (16.2%), Nuclear (10.9%), Petróleo (5%) y Renovable (5%). Codificar un programa que brinde la tabla y su gráfico de barras. La salida sería algo así:



25. El problema 33 de (Introducción al C++, tomo 1, pág. 164) decía: para manejar información acerca de las barritas de dulce, la gerencia de las TRD en Manzanillo pidió a su personal de informática crear una base de datos para operar sus productos. Como piloto se pidió una demostrativa (‘demo’) que maneje las “tente-en-pie” (barritas de dulce o *snacks*, en inglés.) Usar un vector complejo cuyos nodos sean una tripleta. El gerente manejará la comercialización mediante un menú apropiado. Los datos iniciales son:

Marca	Stock	Precio
Mocha Munch	208	0.90
Krispy Krunch	411	1.05
Honey Beez	789	1.25

```
Ejercicio 25. Una base de datos.

1) Imprimir inventario
2) Adicionar un producto
3) Eliminar un producto
4) Editar un precio
5) Editar un stock
6) Vaciar el inventario
7) Pedir ayuda
0) Terminar
Seleccione una opción: 1

En almacén:
=======================================
Marca           Precio  Stock   Costo
---------------------------------------
Mocha Munch      0.90     208  187.20
Krispy Krunch    1.05     411  431.55
Honey Beez       1.25     789  986.25
=======================================

Hay 3 productos diferentes.
Hay 1408 unidades en inventario.
El inventario vale $1605.00
```

26. En el ejercicio 20 de este apartado se resolvió el problema de formar una bibliografía usando un contenedor asociativo multiset o bolsa como una secuencia formada de pares [clave, valor] ordenados por diseño, por lo que el tipo de ordenamiento no se puede controlar. Un contenedor no asociativo con una estructura tipo par [autor, obra] podría permitir un ordenamiento estricto. Rehacer el problema con esta última variante, mostrando un orden estricto. Utilizar un contenedor apropiado.

```
Ejercicio 26: manejo de una biblioteca.

 BIBLIOTECA
 1) Insertar una o varias obras.
 2) Mostrar la biblioteca.
 3) Contar las obras.
 4) Buscar un autor.
 5) Buscar una obra.
 6) Eliminar una obra.
 7) Eliminar un autor.
 8) Vaciar la biblioteca.
 9) Ayuda.
 0) Terminar.
Entre una opción: 2
Autor              Título
Deitel & Deitel.....Cómo programar en C++
Eckel, B............Pensar en C++
Halterman, R........C++ programming
Jaeschke, R.........Header Design
Jaeschke, R.........Void Pointers
```

27. Un vector en el plano real $(\mathbb{R}^2)$ se define como el par ordenado (a, b) que denota sus coordenadas cartesianas rectangulares en los ejes (x, y) respectivamente. Desde este punto de vista un tipo de dato adecuado sería un par de valores. Codificar esta solución.

```
Ejercicio 27: cálculo del producto-punto de dos vectores

Vector A en formato (x, y): (9.8, 6.5)
Vector B en formato (x, y): (5.4, 2.1)

El producto-punto de A y B vale 66.57

Norma de A = 11.76 u
Norma de B = 5.79 u

El coseno entre A y B vale 0.977 rad o 55.98 grados.
```

Aquí hay un contrapunto entre abstracción y facilidad de codificación. Al usar el vector, el cálculo principal —la codificación del producto interior— la hace un algoritmo de la STL, pero el par permite una mayor abstracción, simplificando el código, aunque delegando el cálculo principal al programador (en este caso puntual, ese cálculo es trivial.) ¿Recomendación? Ante casos parecidos, e incluso cuando las soluciones sean triviales, buscar las que aporten la mayoría de los cálculos pedidos mediante la STL, anteponiendo la integridad de la aplicación a todo lo demás.

28. Tomar los números de un archivo llamado muestra.txt y traer sus estadígrafos a consola, línea por línea. Lo correcto es llamarlo desde el DOS: **`prob29 <muestra.txt`**. La salida debiera verse así:

```
c:\temp>prob28 <muestra.txt
Ejercicio 29: algunos estadísticos.

No.   Valor    Mín.    Máx.   Rango    Suma    Moda    Med.   Prom.   Desv.
-------------------------------------------------------------------------
  1       8       8       8       0       8     n/d    8.00    8.00    n/d
  2       6       6       8       2      14     n/d    7.00    7.00    1.41
  3      33       6      33      27      47     n/d    8.00   15.67   15.04
  4      13       6      33      27      60     n/d   10.50   15.00   12.36
  5      59       6      59      53     119     n/d   13.00   23.80   22.40
  6      11       6      59      53     130     n/d   12.00   21.67   20.70
  7      99       6      99      93     229     n/d   13.00   32.71   34.81
  8      96       6      99      93     325     n/d   23.00   40.62   39.23
  9      91       6      99      93     416     n/d   33.00   46.22   40.36
 10      83       6      99      93     499     n/d   46.00   49.90   39.79
 11      72       6      99      93     571     n/d   59.00   51.91   38.33
 12      42       6      99      93     613     n/d   50.50   51.08   36.66
 13      24       6      99      93     637     n/d   42.00   49.00   35.89
 14      24       6      99      93     661      24   37.50   47.21   35.12
 15      66       6      99      93     727      24   42.00   48.47   34.19
 16      71       6      99      93     798      24   50.50   49.88   33.51
 17      93       6      99      93     891      24   59.00   52.41   34.09
 18       9       6      99      93     900      24   50.50   50.00   34.62
 19      73       6      99      93     973      24   59.00   51.21   34.05
 20      62       6      99      93    1035      24   60.50   51.75   33.23
-------------------------------------------------------------------------
```

29. La estadística inferencial es una parte de la estadística que comprende los métodos y procedimientos que… determinan propiedades de una población estadística, a partir de una pequeña parte de esta. Su objetivo es…hacer deducciones sobre una totalidad, basándose en la información numérica. En el muestreo aleatorio simple se puede calcular la probabilidad de extracción de cualquiera de las muestras posibles. Es el más aconsejable y si la muestra es verdaderamente grande, se puede suponer que es sin reposición, esto es, cada lectura es única. (Wikipedia en

español) Codificar un programa que genere un millón de valores gaussianos y les calcula los verdaderos valores. Entonces hacer que se tomen 5 muestras al azar de 2000 valores, se promedien y se constaten contra las respuestas reales. ¿Está la realidad en concordancia con la teoría?

```
Problema 29: muestreo estadístico.
Promedio de 5 muestras de 2000 valores
c/u sobre 1000000 de valores.

Paciencia. Preparando el archivo...hecho.
Resp:
     Media = -0.0003
   Mediana = -0.0011
Desviacion = +0.9997
     Rango = +9.3030

Paciencia. Muestreando...hecho.
Resp:
     Media = -0.0028
   Mediana = -0.0074
Desviacion = +0.9806
     Rango = +6.1048
```

30. La búsqueda puede hacerse de varias formas. El programa 14 muestra una de ellas. Rehacer el programa de la búsqueda usando la programación por composición funcional. La salida no debe cambiar.

```
Ejercicio 30: búsqueda visual; si el valor está, dice cuántas veces y dónde.

Arreglo:

     1       2       5       8       9      10      10      12      13      14
    16      16      17      20      22      22      23      23      23      23
    25      26      28      29      32      34      36      37      37      38
    38      40      40      41      41      41      43      43      44      50
    50      51      51      56      57      57      57      58      58      59
    61      62      62      62      63      64      65      67      67      68
    72      74      75      76      78      80      82      83      85      85
    87      87      91      95      96      96      97      98      98     100

T para terminar. ¿Clave? 3
No está.

T para terminar. ¿Clave? 1
Está una vez a partir de la posición 1

T para terminar. ¿Clave? 57
Está 3 veces a partir de la posición 45

T para terminar. ¿Clave? t
```

31. La biblioteca estándar contiene la clase **`mt19937`** que permite instanciar objetos para la generación de números seudoaleatorios aplicando el algoritmo denominado Mersenne Twister (Wikipedia en español) Para garantizar que los números se distribuyan según cierta función estadística, se unen el motor sembrado con una semilla aleatoria y la función distribuidora de valores en un todo que se le pasa a una función generadora. El mecanismo completo se muestra en las plantillas de generación de números seudoaleatorios. Se pide codificar un programa que genere

10’000,000 de enteros azarosos distribuidos uniformemente en un intervalo [0:9999]. La teoría espera obtener para cualquier valor dado una frecuencia de 1,000 veces como promedio. Hacer que trabaje por 10 veces y promediar el resultado final. Trabajar sobre los valores 5 y 9995. Aunque el verdadero valor de cada uno de los dos números encuestados variará entre ejecuciones, debería estar muy cerca del teórico. La salida se vería más o menos así:

```
Ejercicio 30: generación de valores azarosos con alta calidad.
Los valores dados deben quedar cerca de 1000

Pase #  1 5: 974        9995: 1009
Pase #  2 5: 988        9995: 1035
Pase #  3 5: 1032       9995: 1042
Pase #  4 5: 1034       9995: 955
Pase #  5 5: 1017       9995: 1010
Pase #  6 5: 1023       9995: 1092
Pase #  7 5: 1027       9995: 982
Pase #  8 5: 1031       9995: 986
Pase #  9 5: 1019       9995: 1006
Pase # 10 5: 987        9995: 1042
----------------------------
Promedio para 10 pases: 5: 1013.2; 9995: 1015.9
Tardé 3.398 s
```

Una inspección a su desempeño muestra que en tres ficheros:

- Líneas en total = 199
- Líneas con código = 111
- Eficiencia = 55.8 %

32. Esto que sigue no es propiamente un problema: es la comparación con el anterior. El problema de comparación fue tomado del listado *13.11:highqualityrandom.cpp*, de (Halterman, págs. 403-404) y modificado para que trabajara la misma cantidad de valores y mostrara el consumo de tiempo.

```
//  Programa 30b: high quality random numbers
//  Halterman, ps. 403-404

#include <iostream>
#include <iomanip>
#include <random>

const int NUMBER_OF_TRIALS = 10;

int main() {
    std::cout << "Listing 13.11: high quality random numbers.\n\n";
    std::random_device rdev; // Used to establish a seed value

    // The timing
    clock_v totalBeg, totalEnd;
    double  totalTime = 0;
    totalBeg = clock();

    // Create a Mersenne Twister random number generator with a seed
    // value derived from rd
    std::mt19937 mt( rdev() );

    // Create a uniform distribution object. Given a random
    // number generator, dist will produce a value in the range
    // 0...9999.
```

```
    std::uniform_int_distribution<int> dist( 0, 9999 );

    // Total counts over all the runs.
    // Make these double-precision floating-point numbers
    // so the average computation at the end will use floating-point
    // arithmetic.
    double total5 = 0.0, total9995 = 0.0;

    // Accumulate the results of 10 trials, with each trial
    // generating 1,000,000,000 pseudorandom numbers
    int r, count5, count9995;

    for ( int trial = 1; trial <= NUMBER_OF_TRIALS; trial++ ) {
        // Initialize counts for this run of a billion trials
        count5 = 0;
        count9995 = 0;

        // Generate one billion pseudorandom numbers in the range
        // 0...9,999 and count the number of times 5 and 9,995 appear
        for ( int i = 0; i < 10000000; i++ ) {
            // generate a pseudorandom number in the range 0...9,999
            r = dist( mt );

            if ( r == 5 )
                count5++;    // Number 5 generated, so count it
            else if ( r == 9995 )
                count9995++; // Number 9,995 generated, so count it
        } // for

        // Display the number of times the program generated 5 and 9,995
        std::cout << "Trial #" << std::setw( 2 ) << trial
                  << " 5: " << std::setw( 6 ) << count5
                  << "\t9995: " << std::setw( 6 ) << count9995 << '\n';

        total5 += count5;       // Accumulate the counts to
        total9995 += count9995; // average them at the end
    } // for

    totalEnd = clock();
    totalTime = difftime( totalEnd, totalBeg ) * 0.001;

    std::cout << "-------------------" << '\n';
    std::cout << "Averages for " << NUMBER_OF_TRIALS << " trials: 5: "
              << std::setw( 6 ) << total5 / NUMBER_OF_TRIALS << "; 9995: "
              << std::setw( 6 ) << total9995 / NUMBER_OF_TRIALS << "\n"
              << "Time: " << totalTime << " s\n\n";

    system( "pause" );
    return EXIT_SUCCESS;
}
```

En esta solución la generación de los números fue manual y la salida, similar a la de su texto (pero en inglés), se vio así:

```
Listing 13.11: high quality random numbers.

Trial # 1 5:    994     9995:   1009
Trial # 2 5:   1051     9995:   1003
Trial # 3 5:   1021     9995:   1000
Trial # 4 5:   1028     9995:   1025
Trial # 5 5:   1020     9995:   1023
Trial # 6 5:    950     9995:    946
Trial # 7 5:    949     9995:    984
Trial # 8 5:   1059     9995:    988
Trial # 9 5:    975     9995:    999
Trial #10 5:   1008     9995:   1031
------------------
Averages for 10 trials: 5: 1005.5; 9995: 1000.8
Time: 1.498 s
```

Una inspección a su desempeño muestra que:

- Líneas en total = 75
- Líneas con código = 38
- Eficiencia = 50.7 %

Adaptado a estas condiciones y cronometrado, su desempeño tardó alrededor de 1½ s en hallar y mostrar los valores.

La diferencia en tiempo con el anterior (más o menos la mitad) es despreciable y la eficiencia prácticamente es la misma, PERO con la versión del autor, el programa genera valores distintos en cada ejecución, la aplicación de plantillas y algoritmos garantiza la integridad y precisión de los resultados, y se aporta una gran cantidad de abstracción, pues las plantillas tipifican y simplifican el trabajo.

Parafraseando a Tim Peters:

> *Usar la programación de plantillas junto con la STL —como se hizo en el ejercicio 31— es una gran idea. ¡Hagamos más de esas cosas!*

BIBLIOGRAFÍA

Black Dog Media: Linux Tricks and Tips. (2020). Retrieved mayo 2020, from www.dbmpublications.com.

Deitel, H. M., & Deitel, P. J. (2008). *Como programar en C++* (Sexta ed.). Ciudad México, México: Pearson Educación. Retrieved abril 2019

Eckel, B. (2000). *Pensar en C++* (Vol. I). Madrid, España: McGraw-Hill Interamericana de España SAU.

Galliano Vidal, S. A. (1996). *Introducción a la programación en Turbo C.* Folleto, ISCAB, Departamento de Matemáticas-Computación, Bayamo MN. Retrieved 2019

Galliano Vidal, S. A. (2020). *Introducción al C++, tomo 1* (Primera ed., Vol. I). (D. G. Orozco, Ed.) Manzanillo, Granma, Cuba. Retrieved from http://www.enciclopedia-manzanillo.cu

Halterman, R. L. (2018). *Fundamentals of C++ programming.* USA: Southern Adventist University, School of Computing.

Josuttis, N. M. (1999, august). *The C++ Standard Library: A Tutorial and Reference.* USA: Addison Wesley.

Kalb, J., & Ažman, G. (2015). *C++ Today. The Beast is Back* (Primera ed.). (R. Roumeliotis, & K. Schooling, Eds.) Sebastopol, CA, USA: O'Reilly Media, Inc.

Lipschutz, S. (1991). *Estructura de Datos.* Habana, Habana, Cuba: Ediciones R.

Lischner, R. (2003, may). *C++ in a Nutshell.* USA: O'Reilly.

Plauger, P. (1999, dec). Standard C/C++ FAQs: STL. *XVII*. Miller Freeman Inc. Retrieved 2019

Prata, S. (2012). *C++ Primer Plus* (Sexta ed.). Upper Saddle River, New Jersey, USA: Pearson Education, Inc.

Prata, S. (2014). *C Primer Plus* (Sexta ed.). Upper Saddle River, New Jersey, USA: Pearson Education, Inc. Retrieved 2018

s/a. (s/f). *Lenguaje C. Biblioteca de funciones.* La Habana, Cuba: Ediciones R. Retrieved mayo 2020

Schildt, H. (2003). *C/C++ Programmer's Reference* (Tercera ed.). USA: McGraw-Hill/Osborne.

Stroustrup, B. (1999, may). Learning Standard C++ as a New Language. *XVII*. Miller Freeman Inc. Retrieved 2018

Sutter, H. (2001). Exceptional C++: 47 Engineering Puzzles, Programming Problems, and Solutions. (4), 240. Indianapolis, IN, USA: Addison Wesley Longman, Inc. Retrieved 2018

Sutter, H. (2001). More Exceptional C++. (I. Addison Wesley Longman, Ed.) Indianápolis, IN, USA. Retrieved 2019

VOX: diccionario de uso del español de América y España. (n.d.). Retrieved marzo 2019

Wikipedia en español. (2016). *https://es.wikipedia.org/*, Español. Retrieved 2018

Williams, K. B. (1995, julio). Testing Sort Functions. *XIII*. Miller Freeman Inc. Retrieved 2019, from Software Tools.

CONTRAPORTADA

El autor, Ing. Simón Adolfo Galliano Vidal, jubilado de la Universidad Médica de Granma “Celia Sánchez Manduley” con el cargo de Jefe de Informática del Nodo Provincial de Infomed, pone a disposición del amable lector este libro, segundo de una serie de tres.

Con la precondición de que el lector sabe algo de programar en ANSI C, o mejor aún, como súper C (condición alcanzable mediante el estudio del primer tomo), pero siempre manteniendo un nivel introductorio, su objetivo pedagógico es doble:

1. Enseñar elementos de la programación genérica de plantillas de funciones.
2. Instruir como potenciar al C++ mediante el uso de la biblioteca estándar de plantillas (STL) desde el paradigma de la programación imperativa.

El texto despliega cinco temas: el primero se dedica a presentar el sistema de desarrollo, acercándose al estilo de programación típica de ANSI C, pero conservando la claridad de lectura humana del código. El segundo tema revisa brevemente la historia detrás de la STL y analiza su filosofía de aplicación. El tercer tema se consagra a los elementos de la programación genérica, y desarrolla un sistema de plantillas que luego será utilizado amplia y fuertemente en el resto del libro. El cuarto tema es el más largo y presenta el uso de la STL como sustituto de código hecho a la medida de un problema, aplicándola a ejemplos puntuales del tomo anterior, adaptados a los nuevos objetivos pedagógicos, sincronizados con sus contrapartes e identificados para una posible comparación. El quinto y último tema estudia la ampliación de los conceptos de contenedores cuyos elementos son otros contenedores, o nodos complejos, e indica cómo manejar los distintos niveles de referenciación.

El texto se complementa con 13 ilustraciones aclaratorias, 30 tablas para consultas, 5 programas auxiliares, 20 programas resueltos y concluye con 31 ejercicios propuestos y una solución de comparación

Se adjuntan los programas y las soluciones a los ejercicios planteados. Para cualquiera de ellos casi siempre hay más de una solución, mejor o al menos diferente. Ante la duda el lector debe buscar una personal, que es de las mejores formas de aprender alegre y eficazmente: ¡mediante la experimentación propia!

Se puede contactar al autor por e-mail en mailto: simongv@infomed.sld.cu?subject=texto2 de Cpp para cualquier queja, consulta o sugerencia. Muchas gracias.