Práctica 4: Diseño modular de programas C++ que trabajan con vectores

4.1. Objetivos de la práctica

Los diferentes lenguajes de programación presentan uno o más tipos de datos predefinidos para trabajar con información numérica entera. En C++ hay predefinidos, entre otros, los tipos **`short`**, **`int`**, **`long`** o **`long long`**. En cualquiera de ellos la magnitud de los datos enteros que son capaces de representar está limitada. Si en la implementación del lenguaje se opta por representar los enteros con 32 bits, lo cual sucede en muchas ocasiones, solo se podrán representar los enteros del intervalo [−2 147 483 648, 2 147 483 647] o los naturales del intervalo [0, 4 294 967 295], es decir, todos los enteros o naturales con un máximo de 9 dígitos y solo algunos enteros o naturales de 10 dígitos.

En esta práctica se va a romper la barrera de los 10 dígitos y se va a trabajar con números naturales que puedan tener decenas, centenas o incluso un número mayor de dígitos.

Cada estudiante debe desarrollar, en primer lugar, un módulo de biblioteca denominado `naturales-grandes` que facilite una colección de funciones para trabajar con números naturales cuya magnitud, definida en tiempo de compilación, pueda ser tan grande como se desee.

Posteriormente deberá desarrollar y probar tres programas de aplicación que trabajan con naturales cuyo valor puede desbordar muy ampliamente la magnitud de los tipos de datos enteros predefinidos en C++.

Un primer objetivo de esta práctica es profundizar en el diseño modular de programas.

Un segundo objetivo es el ejercicio en el diseño de algoritmos que trabajan con estructuras indexadas de datos. La forma de representar grandes números naturales propuesta exige el trabajo con cadenas de caracteres, es decir, con estructuras indexadas (vectores) de datos de tipo **`char`**.

4.2. Tecnología y herramientas

4.2.1. Representación de números naturales mediante cadenas de caracteres

En C++ una cadena de caracteres se representa mediante un vector cuyos elementos son datos de tipo **`char`**, que representan los caracteres de la cadena, a los que se añade como último elemento el carácter cuyo código numérico es el 0 (representado en C++ como `'\0'`), que indica que se ha alcanzado el final de la cadena.

Se propone la representación de cualquier número natural mediante una cadena de caracteres. Consideremos la representación, por ejemplo, del número 27 043 573, que consta de un número de dígitos $n = 8$. Puede observarse como tras las componentes que almacenan los dígitos correspondientes a las decenas (`'7'`) y a las unidades (`'3'`), es decir, tras sus dígitos menos significativos, la cadena almacena un carácter cuyo código numérico es el 0 (`'\0'`), cuyo fin es representar el final de la cadena.

`'2'`	`'7'`	`'0'`	`'4'`	`'3'`	`'5'`	`'7'`	`'3'`	`'\0'`	`...`
0	1	2	3	...	...	$n-2$	$n-1$	n	...

Esta representación tiene el inconveniente de que el dígito más significativo está en la componente indexada por 0 del vector y el menos significativo en la componente indexada por $n - 1$, siendo n el número de cifras del número representado. Las posiciones de estos índices son las contrarias a las que dictaría el sentido común matemático, pero esta representación tiene la gran ventaja de que los números así representados se pueden manejar como meras cadenas de caracteres en C++: las operaciones de lectura del teclado o de escritura en la pantalla ya están disponibles a través de los operadores de extracción (`>>`) e inserción (`<<`) y en la biblioteca predefinida `cstring` se encuentran definidas algunas funciones que pueden resultar muy útiles al programar alguna de las funciones a desarrollar en esta práctica.

4.3. Trabajo a desarrollar en esta práctica

Cada estudiante debe completar las tareas que se describen a continuación, realizando un trabajo suficiente con anterioridad a la sesión de prácticas que le corresponda, con el objeto de sacar el máximo rendimiento de dicha sesión.

4.3.1. Tarea 1. Definición del módulo de biblioteca `naturales-grandes`

Crea y configura en el área de trabajo «Práctica 4» y en la carpeta «`Practica4`» un proyecto denominado «`NaturalesGrandesTest`», que nos va a servir para facilitar el desarrollo y la realización de pruebas de un módulo denominado `naturales-grandes`.

Define el módulo de biblioteca `naturales-grandes` en la carpeta «`Programacion1/Biblioteca`», creando o copiando en ella los ficheros «`naturales-grandes.h`» (que puedes descargar de la web de la asignatura) y «`naturales-grandes.cpp`».

Añade el fichero de interfaz y el de implementación del módulo al proyecto «`NaturalesGrandesTest`» directamente desde el directorio «`Biblioteca`».

Descarga de la página web de la asignatura los ficheros «`naturales-grandes-test.h`», «`naturales-grandes-test.cpp`» y «`naturales-grandes-test-main.cpp`», guardándolos en tu directorio «`Practica4/NaturalesGrandesTest`» y añadiéndolos al proyecto «`NaturalesGrandesTest`».

Si lo has hecho correctamente, el proyecto «`NaturalesGrandesTest`» compilará con errores indicando que las funciones del módulo `naturales-grandes` no están definidas. Impleméntalas en el fichero «`naturales-grandes.cpp`» después de leer la descripción del módulo que figura a continuación.

El módulo `naturales-grandes` facilita a otros módulos las siguientes funciones para trabajar con números naturales de una gran magnitud:

- Función `void convertir(const int n, char secuencia[])`, que permite transformar un natural (`n`) en una cadena de caracteres que almacena la secuencia de sus dígitos (`secuencia`).

- Función **int** `valor(`**const char** `secuencia[])`, que devuelve el valor numérico de un natural representado por una cadena de caracteres (`secuencia`) que almacena la secuencia de sus dígitos.
- Función **void** `sumar(`**const char** `a[],` **const char** `b[],` **char** `suma[])`, que permite sumar dos números naturales representados mediante dos cadenas (`a` y `b`) que almacenan las secuencias de dígitos de cada uno de ellos, asignando el resultado al tercer parámetro (`suma`), también representado como una secuencia de dígitos.
- Función **void** `calcularImagen(`**const char** `numero[],` **char** `imagen[])`, que permite calcular la imagen especular de un número natural representad mediante una cadena de caracteres (`numero`) que almacena su secuencia de dígitos, asignando el resultado al parámetro `imagen`, también representado como una secuencia de dígitos.
- Función **bool** `esCapicua(`**const char** `numero[])`, que permite determinar si un número natural representado mediante una cadena de caracteres (`numero`) que almacena su secuencia de dígitos, es o no capicúa.

Existen en C++ funciones de bibliotecas predefinidas predefinidas que permiten resolver los problemas planteados por algunas de las funciones con una mera invocación a la función adecuada. El objetivo de solicitar la implementación de dichas funciones en la presente tarea no es el de localizar y utilizar dichas funciones, sino el de ejercitarse en la manipulación de cadenas de caracteres para terminar de comprenderlas, por lo que se recomienda implementar las mencionadas funciones utilizando únicamente los recursos explicados en clase.

En cualquier caso, en las clases de teoría se ha trabajado con funciones que resuelven problemas análogos (aunque algo más complejos) a los correspondientes a las funciones `convertir` y `valor`. El código de las funciones referidas ha sido publicado en la web de la asignatura.

El fichero de interfaz del módulo `naturales-grandes` es el siguiente:

```
/*****************************************************************************
 * Programación 1. Práctica 4
 * Autores: Javier Martínez y Miguel Ángel Latre
 * Ultima revisión: 10 de noviembre de 2018
 * Resumen: Fichero de interfaz «naturales-grandes.h» de un módulo denominado
 *          «naturales-grandes» para trabajar con números naturales de gran
 *          magnitud el la 4.ª práctica.
 * Codificación de caracteres original de este fichero: UTF-8 con BOM
\*****************************************************************************/

#ifndef NATURALES_GRANDES_H_INCLUDED
#define NATURALES_GRANDES_H_INCLUDED

/*
 * Pre:  n >= 0
 * Post: «secuencia» almacena una cadena de caracteres, la secuencia
 *       de dígitos de «n». Sea «nD» el número de dígitos de «n».
 *       El carácter secuencia[«nD»-1] representa las unidades de «n»,
 *       el carácter secuencia[«nD»-2] las decenas de «n» y así, sucesivamente, el
 *       carácter secuencia[0] representa el dígito más significativo. El carácter
 *       secuencia[«nD»] es igual al carácter '\0' (cuyo código ASCII es 0).
 */
void convertir(const int n, char secuencia[]);
```

```
/*
 * Pre:  «secuencia» almacena una secuencia de caracteres que corresponden a los
 *       dígitos de un número natural comprendido en el intervalo de
 *       valores representables por el tipo «int», seguida por el carácter '\0'
 *       (cuyo código ASCII es 0).
 * Post: Ha devuelto el valor numérico del natural almacenado en «secuencia».
 */
int valor(const char secuencia[]);

/*
 * Pre:  «a» y «b» almacenan sendas cadenas de caracteres con la secuencias
 *       de dígitos de dos números naturales. Sea «nA» el número de dígitos de «a»
 *       y «nB» el número de dígitos de «b». Los caracteres a[«nA»-1] y b[«nB»-1]
 *       representan las unidades, los caracteres a[«nA»-2] y b[«nB»-2] las
 *       decenas y así sucesivamente.
 * Post: «suma» almacena la secuencia de dígitos de la suma de los naturales
 *       representados por «a» y «b». Sea «nS» el número de dígitos de suma.
 *       El carácter suma[«nS»-1] representa las unidades de la suma, el carácter
 *       suma[«nS»-2] representa las decenas y así sucesivamente.
 */
void sumar(const char a[], const char b[], char suma[]);

/*
 * Pre:  «numero» almacena una cadena de caracteres con la secuencia de dígitos de
 *       un número natural. Sea «nD» el número de dígitos de «numero».
 *       El carácter numero[«nD»-1] representa las unidades de «numero»,
 *       el carácter numero[«nD»-2] las decenas de «numero» y así,
 *       sucesivamente, el carácter numero[0] representa el dígito más
 *       significativo. El carácter numero[«nD»] es igual al carácter '\0' (cuyo
 *       código ASCII es 0).
 * Post: «imagen» almacena una cadena de caracteres con la secuencia de dígitos
 *       correspondiente a la imagen especular de «numero».
 */
void calcularImagen(const char numero[], char imagen[]);

/*
 * Pre:  «numero» almacena una cadena de caracteres con la secuencia de dígitos de
 *       un número natural. Sea «nD» el número de dígitos de «numero».
 *       El carácter numero[«nD»-1] representa las unidades de «numero»,
 *       el carácter numero[«nD»-2] las decenas de «numero» y así,
 *       sucesivamente, el carácter numero[0] representa el dígito más
 *       significativo. El carácter numero[«nD»] es igual al carácter '\0' (cuyo
 *       código ASCII es 0).
 * Post: Ha devuelto «true» si y solo si la secuencia de dígitos «numero» se
 *       corresponde con la de un número capicúa.
 */
bool esCapicua(const char numero[]);

#endif // NATURALES_GRANDES_H_INCLUDED
```

4.3.2. Tarea 2. Números de Fibonacci grandes

Diseña un programa interactivo que, al ser ejecutado, itere el siguiente diálogo con el usuario hasta que este responda con un 0 o un negativo. En cada iteración, pide al usuario que determine las posiciones del término inicial y final de la sucesión de Fibonacci que debe presentar a continuación. El programa ha de ser capaz de ser capaz de calcular términos de la sucesión de Fibonacci de hasta 300 dígitos.

Este programa debe ser desarrollado en el área de trabajo «Práctica 4», en un proyecto denominado «`Fibonacci`».

```
Términos inicial y final (0 o negativo para acabar): 10 20
10. 34
11. 55
...
19. 2584
20. 4181

Términos inicial y final (0 o negativo para acabar): 40 60
40. 63245986
41. 102334155
...
59. 591286729879
60. 956722026041

Términos inicial y final (0 o negativo para acabar): 0
```

4.3.3. Tarea 3. Potencias de 2 grandes

Diseña un programa interactivo que, al ser ejecutado, itera el siguiente tipo de diálogo con el usuario hasta que este responde con un 0 o un negativo. En cada iteración, pide al usuario que determine un número de dígitos y el programa escribe el exponente y valor de la primera potencia de 2 cuyo número de dígitos es mayor o igual que el número de dígitos introducidos por el usuario. El programa ha de ser capaz de ser capaz de trabajar con valores de potencias de 2 de hasta 500 dígitos.

Este programa debe ser desarrollado en el área de trabajo «Práctica 4», en un proyecto denominado «`PrimeraPotencia`».

```
Número de dígitos (0 o negativo para acabar): 1
1 es 2 elevado a la 0.ª potencia
y es la primera potencia de 2 de 1 digitos.

Número de dígitos (0 o negativo para acabar): 2
16 es 2 elevado a la 4.ª potencia
y es la primera potencia de 2 de 2 digitos.

Número de dígitos (0 o negativo para acabar): 3
128 es 2 elevado a la 7.ª potencia
y es la primera potencia de 2 de 3 digitos.

Número de dígitos (0 o negativo para acabar): 4
1024 es 2 elevado a la 10.ª potencia
y es la primera potencia de 2 de 4 digitos.

Número de dígitos (0 o negativo para acabar): 10
1073741824 es 2 elevado a la 30.ª potencia
y es la primera potencia de 2 de 10 digitos.

Número de dígitos (0 o negativo para acabar): 21
147573952589676412928 es 2 elevado a la 67.ª potencia
y es la primera potencia de 2 de 21 digitos.

Número de dígitos (0 o negativo para acabar): 0
```

4.3.4. Tarea 4. Números de Lychrel

Dado un número natural *n*, vamos a considerar un proceso aritmético consistente en calcular la imagen especular de *n* y sumársela el propio *n*, proceso que se repite con la suma resultante hasta obtener un número capicúa.

Se denomina *número de Lychrel* a un número natural que nunca produce un número capicúa cuando se le aplica reiteradamente el proceso descrito en el párrafo anterior.

Así, por ejemplo, 56 no es un número de Lychrel, puesto que 56 + 65 = 121, que es un número capicúa. El número 58 tampoco es número de Lychrel, puesto que 58 + 85 = 143, que, en este caso, no es capicúa; pero repitiendo el proceso con el número resultante (143), se obtiene el número 484 (= 143 + 341), que sí es capicúa.

Alrededor del 80 % de los números naturales por debajo de 10 000 producen un número capicúa en 4 iteraciones o menos, y en torno al 90 % lo producen en 7 o menos. El número 89 necesita 24 iteraciones del proceso hasta que se convierte en un capicúa[1].

No se ha demostrado todavía que, en base 10, exista ningún número de Lychrel, aunque el número 196 es el natural más pequeño que podría serlo.

Escribe un programa que solicite al usuario un número natural y que muestre el proceso de obtener un capicúa a través del proceso iterativo ilustrado previamente de sumarlo con su imagen especular y

[1] Wikipedia. «Lychrel number» *Wikipedia, The Free Encyclopedia.* `https://en.wikipedia.org/w/index.php?title=Lychrel_number`. Consultado el 10 de noviembre de 2018.

repetir. El programa de debe terminar cuando se obtenga un capicúa o un resultado alcance los 1000 dígitos. Se muestran a continuación los resultados de la ejecución del programa con varias entradas de usuario:

```
Escriba un número natural: 22
Iteración 0: 22
```

```
Escriba un número natural: 56
Iteración 0: 56
Iteración 1: 56 + 65 = 121
```

```
Escriba un número natural: 58
Iteración 0: 58
Iteración 1: 58 + 85 = 143
Iteración 2: 143 + 341 = 484
```

```
Escriba un número natural: 89
Iteración 0: 89
Iteración 1: 89 + 98 = 187
Iteración 2: 187 + 781 = 968
Iteración 3: 968 + 869 = 1837
Iteración 4: 1837 + 7381 = 9218
Iteración 5: 9218 + 8129 = 17347
Iteración 6: 17347 + 74371 = 91718
Iteración 7: 91718 + 81719 = 173437
Iteración 8: 173437 + 734371 = 907808
Iteración 9: 907808 + 808709 = 1716517
Iteración 10: 1716517 + 7156171 = 8872688
Iteración 11: 8872688 + 8862788 = 17735476
Iteración 12: 17735476 + 67453771 = 85189247
Iteración 13: 85189247 + 74298158 = 159487405
Iteración 14: 159487405 + 504784951 = 664272356
Iteración 15: 664272356 + 653272466 = 1317544822
Iteración 16: 1317544822 + 2284457131 = 3602001953
Iteración 17: 3602001953 + 3591002063 = 7193004016
Iteración 18: 7193004016 + 6104003917 = 13297007933
Iteración 19: 13297007933 + 33970079231 = 47267087164
Iteración 20: 47267087164 + 46178076274 = 93445163438
Iteración 21: 93445163438 + 83436154439 = 176881317877
Iteración 22: 176881317877 + 778713188671 = 955594506548
Iteración 23: 955594506548 + 845605495559 = 1801200002107
Iteración 24: 1801200002107 + 7012000021081 = 8813200023188
```

```
Escriba un número natural: 196
Iteración 0: 196
Iteración 1: 196 + 691 = 887
Iteración 2: 887 + 788 = 1675
...
Iteración 2391: 92966934...43867028 + 82076834...43966929 = 175043768...87833957
```

Este programa debe ser desarrollado en el área de trabajo «Práctica 4», en un proyecto denominado «`Lychrel`».

4.3.5. Resultados del trabajo desarrollado en las cuatro primeras prácticas

Como resultado de las cuatro primeras prácticas, cada estudiante dispondrá en su cuenta de `hendrix` de una carpeta denominada «`Programacion1`» dentro de la cual se localizarán los siguientes proyectos de programación en C++ organizados en los directorios y denominaciones que se detallan a continuación:

1. Directorio «`Biblioteca`» con los seis ficheros correspondientes a los siguientes módulos de biblioteca: `calculos`, `fechas` y `naturales-grandes`.
2. Área de trabajo «Práctica 1» con los siguientes proyectos ubicados en la carpeta «`Programacion1/Practica1`»:
   - Proyectos C++ «`Bienvenida`», «`Circunferencias`», «`Circulo`» y «`Formatear`».
3. Área de trabajo «Práctica 2» con los siguientes proyectos ubicados en la carpeta «`Programacion1/Practica2`»:
   - Proyectos C++ «`Fecha`», «`CambioMoneda`», «`Cajero`», «`Tiempo`», «`Romano`» y «`Trigonometria`».
4. Área de trabajo «Práctica 3» con los siguientes proyectos ubicados en la carpeta «`Programacion1/Practica3`»:
   - Proyectos C++ «`Calculadora`», «`CalculadoraTest`», «`FechasTest`», «`MenuFechas`».
5. Área de trabajo «Práctica 4» con los siguientes proyectos ubicados en la carpeta «`Programacion1/Practica4`»:
   - Proyecto C++ «`Fibonacci`».
   - Proyecto C++ «`PrimeraPotencia`».
   - Proyecto C++ «`Lychrel`».