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

4.1. Introducción

Los diferentes lenguajes de programación disponen de uno o más tipos de datos predefinidos para trabajar con información numérica entera. En el caso de C++ hay predefinidos, entre otros, los tipos `short`, `int`, `long`, `long long`, `unsigned short`, `unsigned`, `unsigned long` y `unsigned long long`. Independientemente del tipo de dato que se use, la magnitud de los datos enteros que se pueden representar está limitada. Por ejemplo, cuando un compilador en concreto opta por representar los enteros con 32 bits (lo cual sucede en el caso de MinGW y GCC, los compiladores que estamos utilizando en este curso en Windows y en GNU/Linux/macOS, respectivamente), solo es posible representar los enteros del intervalo [−2 147 483 648, 2 147 483 647] (*enteros con signo*) o los naturales del intervalo [0, 4 294 967 295] (*enteros sin signo*), 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 esa 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.

En primer lugar, vas a desarrollar un módulo denominado «`naturales-grandes`» que facilite una colección de procedimientos y funciones para trabajar con números naturales cuya magnitud, definida en tiempo de compilación, pueda ser tan grande como se desee. Posteriormente, desarrollarás y probarás al menos tres de los cuatro programas de aplicación que se proponen. Estos problemas trabajan con naturales cuyo valor puede desbordar muy ampliamente la magnitud de los tipos de datos enteros predefinidos en C++.

4.2. Representación de naturales grandes mediante vectores

En esta práctica, **se propone la representación de números naturales mediante vectores de enteros sin signo**. Cada vector tendrá `NUM_DIGITOS` componentes (constante definida con el valor 1000 en el código de partida suministrado) y en él se almacenarán los dígitos en base 10 del número natural a representar. En la componente indexada por 0 del vector se almacenará el dígito correspondiente a las unidades de ese número natural, en la componente indexada por 1, las decenas, en la indexada por 2, las centenas, y así sucesivamente.

Las componentes correspondientes a órdenes de magnitud superior a la de la cifra más significativa del número natural a representar tendrán valor cero.

Considera, a modo de ejemplo, la representación del número natural 1 742 863 427 043 573, que consta de 16 cifras. Con el esquema propuesto, su representación sería la siguiente:

999	998	...	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	0	...	0	0	0	1	7	4	2	8	6	3	4	2	7	0	4	3	5	7	3

El vector que representa al número natural 1 742 863 427 043 573 tiene `NUM_DIGITOS` (1000) componentes. Puede observarse que en la componente indexada por 0 del vector se almacena el dígito menos significativo (el valor 3); en la componente indexada por 1, las decenas (el valor 7); y así sucesivamente. La componente más significativa de este número natural de 16 cifras se almacena en la componente de índice 15 (el valor 1). A partir de esa componente, las restantes componentes (índices 16 a 999) almacenan el valor 0.

4.3. Trabajo a desarrollar en esta práctica

Es conveniente realizar un trabajo suficiente con anterioridad a las sesiones de prácticas que te correspondan, con el objeto de sacar el máximo rendimiento a dicha sesión.

El código fuente de los programas a desarrollar en esta práctica se localizará en un directorio denominado «`practica4`», ubicado dentro de tu carpeta «`Programacion1`». Para que el programa de pruebas descrito en la Sección 4.3.1 compile correctamente, el directorio «`practica4`» tiene que estar al mismo nivel que el directorio «`practica3`», es decir, ambos directorios tienen que estar en el mismo directorio (la carpeta «`programacion1`»). En la Sección 4.3.1 se muestra un esquema en el que se indica cómo tiene que quedar la organización de estos directorios.

En el repositorio `https://github.com/prog1-eina/practica4` tienes el código de partida para esta práctica, que contiene el módulo de biblioteca «`naturales-grandes`» con el que se va a trabajar, así como los esqueletos de los programas a desarrollar y un programa que hace pruebas de unidad. Todos ellos ya están configurados para ser compilados, ejecutados y depurados a través de los ficheros «`Makefile`» necesarios y a través de la definición de tareas de Visual Studio Code.

Puedes descargarte el área de trabajo completa (botón Code › Download ZIP de la web del repositorio), y descomprimirla en tu directorio «`programacion1`» como «`practica4`» (recuerda borrar el sufijo «`-main`» que añade GitHub al preparar el fichero comprimido).

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

Descripción y organización del código de partida

Abre en Visual Studio Code el directorio «`practica4`», con la opción de menú File › Open Folder... y elige la carpeta «`practica4`». Recuerda que si no la abres de esta manera, no verás las tareas configuradas que te proporcionamos y que te facilitará el trabajo de compilación, ejecución y depuración. Una vez abierta, si seleccionas el botón Explorer (el primero de los del panel de la izquierda) tienes que ver el nombre del directorio «`practica4`» en mayúsculas y, colgando de él, los directorios «`.vscode`», «`example`», «`src`», «`test`» y el fichero «`Makefile`». Estos elementos son, en concreto:

Directorio «`src`»: Contiene el código fuente con el que tendrás que trabajar, que consta de un módulo de biblioteca (el módulo «`naturales-grandes`», que se describe más adelante) y los esqueletos de cuatro módulos principales («`factorial-main.cpp`», «`fibonacci-main.cpp`», «`lychrel-main.cpp`» y «`primera-potencia-main.cpp`») en cuyas implementaciones tendrás que utilizar el módulo «`naturales-grandes`». El comportamiento de estos programas se especifica a continuación en las tareas 2 a 5.

Directorio «`test`»: Contiene el código fuente de un programa de pruebas de unidad de las funciones del módulo «`naturales-grandes`». Tienes más detalles en la sección 4.3.1.

El fichero «`Makefile`»: Contiene las reglas necesarias para que la herramienta «`make`» invoque al compilador de C++ de forma que se generen los programas «`factorial`», «`primera-potencia`», «`fibonacci`», «`lychrel`» y el programa de prueba del módulo «`naturales-grandes`», cuyo fuente está en el directorio «`test`».

Directorio «`.vscode`»: Contiene los ficheros de configuración de las tareas de Visual Studio Code que permiten compilar, ejecutar y depurar cada uno de los programas con los que se trabajará en esta práctica. Los detalles sobre su contenido los puedes volver a revisar en el guion de la práctica 3.

Directorio «`example`»: Contiene dos programas muy simples para ilustrar las diferencias sintácticas entre operar con enteros predefinidos y los naturales grandes que vas a implementar utilizando vectores. Tienes más información al respecto en la sección 4.3.1.

Descripción del módulo «`naturales-grandes`»

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

- Procedimiento `void copiar(const unsigned original[], unsigned copia[])`, que permite copiar el número natural grande representado por el vector `original` al vector `copia`. Como los vectores no tienen disponible el operador de asignación, utilizaremos esta función como sustituta y así poder copiar el valor de un número natural grande representado en un vector a otro vector.

- Función `bool sonIguales(const unsigned a[], const unsigned b[])`, que permite comprobar si el número natural grande representado por el vector `a` es igual al número natural grande representado por el vector `b`. De forma análoga al caso del procedimiento `copiar`, como los vectores no tienen disponible el operador de comparación, utilizaremos esta función como sustituta y así poder comprobar si los números naturales grandes representados por los dos vectores de entrada son el mismo o no.

- Función `unsigned numCifras(const unsigned natural[])`, que devuelve el número de cifras del número natural grande representado por el vector `natural` (esto es, el número de cifras con las que se escribe ese número natural grande sin ceros a la izquierda).

- Procedimiento `void escribirEnPantalla(const unsigned natural[])`, que escribe en la pantalla el número natural grande representado por el vector `natural` (sin ceros a la izquierda).

  Nota: Por la naturaleza de este procedimiento, no se proporcionan pruebas para el mismo en el programa de pruebas facilitado en esta práctica. Tenlo en cuenta y plantéate alguna estrategia para hacer pruebas de este procedimiento antes de utilizarlo en cualquiera de los programas solicitados en las tareas 2 a 5.

- Procedimiento `void convertir(const unsigned numero, unsigned naturalGrande[])`, que permite transformar un natural representable como `unsigned` (el valor del parámetro `numero`) en un vector de enteros que almacena la secuencia de sus dígitos (`naturalGrande`).

- Función `unsigned valor(const unsigned naturalGrande[])`, que devuelve el valor numérico de un número natural grande representado por un vector que almacena la secuencia de sus dígitos (`naturalGrande`).

Nota: No todos los valores de un vector que representa un número natural grande son representables como datos de tipo `unsigned`. En concreto, esta función solo producirá un resultado correcto cuando el número natural grande representado por `naturalGrande` sea inferior o igual al mayor `unsigned` representable, que es $2^{32} - 1 = 4\,294\,967\,295$. En otros casos, el valor devuelto por la función no está definido (es decir, puede ser cualquier cosa).

- Procedimiento `void sumar(const unsigned a[], const unsigned b[], unsigned suma[], bool &desbordamiento)` que permite sumar dos números naturales grandes representados mediante dos vectores (`a` y `b`), asignando el resultado al tercer parámetro (`suma`), también representado como un vector de enteros sin signo.

  Notas: Este procedimiento y el siguiente son, probablemente, los más complicados de implementar de toda la práctica y, al menos uno de los dos, tienen que funcionar perfectamente para poder hacer los programas solicitados posteriormente. Para plantearte un algoritmo capaz de sumar los números almacenados en los vectores `a` y `b`, plantéate cómo te enseñaron a sumar números dígito a dígito en primaria, «llevándote una» cuando hacía falta o cómo estáis haciendo sumadores en Introducción a los Computadores, utilizando acarreos.

  Pese a que estamos trabando con vectores de 1 000 dígitos, el problema del desbordamiento con una operación como la suma no lo vamos a poder evitar, simplemente lo estamos «retrasando» con respecto a lo que ocurriría al trabajar con enteros predefinidos (donde el desbordamiento ocurre a partir de los 10 o 20 dígitos, dependiendo del tipo predefinido con el que trabajáramos). Por eso en esta función `sumar` hay un parámetro extra (`desbordamiento`) para indicar si el resultado de una suma tiene más de 1 000 dígitos.

- Procedimiento `void multiplicar(const unsigned a[], const unsigned b, unsigned producto[], bool &desbordamiento)`, que permite multiplicar un número natural grande representado mediante un vector (`a`) por otro representado por un `unsigned` (`b`), asignando el resultado al tercer parámetro (`producto`), también representado como un vector de enteros sin signo.

  Nota: Fíjate en que el segundo parámetro (`b`) no es un vector de enteros, sino directamente un `unsigned`. Esto, lejos de ser una complicación, es una simplificación que te va a permitir utilizar un algoritmo bastante similar al de la suma para hacer la multiplicación.

- Procedimiento `void calcularImagen(const unsigned natural[], unsigned imagen[])`, que permite calcular la imagen especular de un número natural grande representado mediante el vector de enteros sin signo `natural`, asignando el resultado al parámetro `imagen`, también representado como un vector de enteros sin signo.

- Función `bool esCapicua(const unsigned natural[])`, que permite determinar si un número natural grande representado mediante el vector de enteros sin signo `natural` es o no es capicúa. Recuerda que un número es capicúa si se lee igual de izquierda a derecha que de derecha a izquierda (por ejemplo, los números 13 355 331, 123 454 321 y 44 555 544 son capicúas).

Diferencias entre trabajar con enteros predefinidos y con «naturales grandes» implementados a través de vectores

En esta práctica, tienes que trabajar con números naturales grandes representados a través de sus dígitos utilizando los procedimientos y funciones del módulo «`naturales-grandes`» con el objeto de evitar los problemas de desbordamiento que se producen cuando los tipos enteros predefinidos no son capaces de representar números grandes.

Por ejemplo, el siguiente programa presenta un ejemplo de desbordamiento:


```
/*
 * Programa que muestra un desbordamiento al utilizar datos de tipo unsigned de 9 dígitos.
 */
int main() {
    // 1. Declaración e inicialización de variables
    unsigned a = UINT_MAX;     // Máximo unsigned representable: 2^32 - 1
    unsigned b = 1;

    // 2. Cálculo de la suma
    unsigned suma = a + b;

    // 3. Escritura de resultados
    //    (aunque, por diseño del programa, sabemos que hemos provocado un desbordamiento)
    cout << "Suma de " << UINT_MAX << " y 1: " << suma << endl;

    // 4. Devolución de código de salida 1: sabemos que ha escrito 0 en lugar de 4294967296
    return 1;
}
```

El siguiente ejemplo, que utiliza el módulo «`naturales-grandes`», evita el problema de desbordamiento, puesto que está configurado para trabajar con números naturales de hasta 1 000 dígitos:

```
/*
 * Programa que muestra un desbordamiento al utilizar datos de tipo unsigned de 9 dígitos.
 */
int main() {
    // 1. Declaración e inicialización de variables
    unsigned a[NUM_DIGITOS], b[NUM_DIGITOS];
    convertir(UINT_MAX, a);
    convertir(1, b);

    // 2. Cálculo de la suma
    unsigned suma[NUM_DIGITOS];
    bool desborda;
    sumar(a, b, suma, desborda);

    // 3. Determinación de la corrección del resultado
    if (desborda) {
        // 4. «desborda» es «true»: se ha producido un desbordamiento
        cout << "Error: ha desbordado (y no debería)" << endl;
        return 1;
    } else {
        // 5. «desborda» es «false»: se ha calculado el resultado correcto
        // Escritura de resultados
        cout << "Suma de " << UINT_MAX << " y 1: ";
        escribirEnPantalla(suma);
        cout << endl;

        // 6. Devolución de código de salida 0 si ha escrito correctamente 4294967296
        return 0;
    }
}
```

Observa que, pese a que las acciones algorítmicas de los dos programas mostrados son casi las mismas, las construcciones sintácticas utilizadas en ambas son muy distintas. Las operaciones disponibles para datos de tipo primitivo como el **`unsigned`** son sustituidas en el caso del programa que utiliza el módulo «`naturales-grandes`» por invocaciones a procedimientos.

Esto mismo ocurrirá cuando implementes algunos de los procedimientos y funciones del módulo «`naturales-grandes`». Algunos son similares a funciones realizadas en la práctica 3, pero tendrás que adaptar los algoritmos a la sintaxis exigida por los procedimientos y funciones del módulo «`naturales-grandes`».

Además, el esquema basado en convertir el número natural grande a un dato de tipo **`unsigned`** a través de la función `valor`, realizar la operación deseada (cálculo de la suma, de la imagen o determinación de si es capicúa) con los operadores y funciones definidos para datos de tipo **`unsigned`** y volver a convertir el resultado a un vector con la función `convertir` no va a funcionar con carácter general. Como se ha comentado anteriormente, esto es debido a que la función `valor` solo puede devolver valores correctos cuando el número natural grande tiene 9 dígitos o menos. En el resto de los casos, se producirán comportamientos no definidos (normalmente, desbordamientos).

Tienes disponibles los programas correspondientes a los dos ejemplos anteriores en el directorio `example` del repositorio correspondiente a la práctica. El segundo ejemplo no te funcionará correctamente hasta que no hayas implementado los procedimientos `convertir`, `sumar` y `escribirEnPantalla` del módulo «`naturales-grandes`».

Descripción de un programa de pruebas para el módulo «`naturales-grandes`»

En el directorio «`test`» se encuentra la mayor parte del código de un programa que va a servir para facilitar el desarrollo y la realización de pruebas del módulo denominado «`naturales-grandes`».

El programa consta de cuatro módulos organizados en siete ficheros:

1. El módulo objeto de las pruebas de este programa, «`naturales-grandes`», que se encuentra en el directorio «`practica4/src`» y cuyo fichero de implementación tienes que completar.
2. El módulo «`naturales-grandes-test`», formado también por un fichero de interfaz y otro de implementación. Ambos ficheros se encuentran en el directorio «`practica4/test`». Este módulo contiene funciones que permitirán comprobar el correcto funcionamiento de las funciones del módulo «`naturales-grandes`» que implementarás.
3. El módulo principal correspondiente al fichero «`naturales-grandes-test-main.cpp`», que se ubica también en el directorio «`practica4/test`». Este módulo especifica las pruebas concretas que se hacen de las funciones del módulo «`naturales-grandes`».
4. El módulo «`testing-prog1`», ya utilizado en la práctica 3. Está compuesto por los ficheros «`testing-prog1.hpp`» y «`testing-prog1.cpp`», ubicados en el directorio «`practica3/test/testing-prog1`».

Para que este programa se compile correctamente, las ubicaciones y nombres de los directorios correspondientes a las prácticas 3 y 4 y sus subdirectorios tienen que ser exactamente los indicados en los enunciados de las prácticas:


```
programacion1/
├── ...
├── practica3/
│   ├── ...
│   └── test/
│       ├── ...
│       └── testing-prog1/
│           ├── testing-prog1.hpp
│           └── testing-prog1.cpp
└── practica4/
    ├── ...
    ├── src/
    │   ├── naturales-grandes.hpp
    │   ├── naturales-grandes.cpp
    │   └── ...
    ├── test/
    │   ├── naturales-grandes-test.hpp
    │   ├── naturales-grandes-test.cpp
    │   └── naturales-grandes-test-main.cpp
    └── ...
```

Si no fuese así, ajusta las ubicaciones y nombres de los directorios hasta que el programa de pruebas se pueda compilar y ejecutar.

Puedes compilar el programa de pruebas ejecutando el comando `make naturales-grandes-test` en la terminal de Visual Studio Code o ejecutando la tarea "Compilar tests del módulo «naturales-grandes»" del menú Terminal > Run Tasks... . Para ejecutar el programa, puedes ejecutar la orden `bin/naturales-grandes-test` en la terminal de Visual Studio Code una vez que el programa ha sido compilado sin errores, lanzar la tarea "Ejecutar tests del módulo «naturales-grandes»" desde el menú Terminal > Run Tasks... .

Inicialmente, el programa «`naturales-grandes-test`» compilará con varias advertencias debido a que el código del fichero de implementación «`naturales-grandes.cpp`» no está completo. Al ser ejecutado, también informará de errores en los resultados de todas las pruebas realizadas. El número de errores detectados en las pruebas se irá reduciendo hasta llegar a cero conforme implementes correctamente las funciones del módulo.

En el fichero «`naturales-grandes-test-main.cpp`» puedes ver los casos de prueba que ejecuta el programa de pruebas. Para facilitar la escritura de este programa, los números naturales grandes se han expresado como cadenas de caracteres[1]. Esto implica que, si lo necesitas, puedes añadir fácilmente nuevos casos de prueba que te ayuden a detectar o corregir errores que pueda haber en tus implementaciones de las funciones del módulo «`naturales-grandes`».

El programa de pruebas inicializa intencionadamente todas las componentes de los vectores que sirven de argumento a parámetros de salida de las funciones que tenéis que implementar con un valor determinado, de forma que el propio programa de pruebas puede detectar si tu código inicializa adecuadamente todas las componentes de los vectores.

Un mensaje indicando que el resultado calculado ha sido, por ejemplo, «`???...???902`» indica, precisamente, que no has inicializado adecuadamente todas las componentes del vector. Cada signo de interrogación representa, en ese caso, una componente que no has inicializado.

[1] En el módulo «`naturales-grandes-test`» se han definido funciones que convierten estas cadenas de caracteres a los vectores de dígitos que se utilizan en el módulo «naturales-grandes». No es necesario comprender el código de estas funciones para compilar y utilizar el programa de pruebas. No obstante, las cadenas de caracteres se explicarán próximamente en clase de teoría.

De forma similar, cuando el programa de pruebas muestra un resultado incorrecto en el que en una o varias componentes hay un dato entero que no está entre 0 y 9, lo indicará escribiendo el valor de esa componente entre corchetes.

Así, por ejemplo, un mensaje indicando que el resultado calculado ha sido «`18[10]5[32]`» indicaría que tu vector almacena el siguiente resultado **erróneo**, puesto que cada componente solo debe almacenar números entre 0 y 9:

999	998	...	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	0	...	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	8	**10**	5	**32**

Por último, ten en cuenta que el programa de pruebas **no** hace pruebas del procedimiento `escribirEnPantalla`.

4.3.2. Tarea 2. Factorial

Diseña un programa interactivo que, al ser ejecutado, solicite al usuario un número natural y escriba en la pantalla su factorial correspondiente, tal y como se muestra en el siguiente ejemplo de ejecución:

```
Escriba un número natural: 42
42! = 1405006117752879898543142606244511569936384000000000
```

El programa ha de ser capaz de calcular factoriales de hasta `NUM_DIGITOS` dígitos:

```
Escriba un número natural: 449
449! = 385193051802807257632158476912128755483958058935344670126490557678962889262985
944578816686286764157914351361878187202174635968529028925560185495470696703723781598 1
933427173547163838273480784018675124958304298372030008135578113075160109993539942002 5
859541702588941624119769786447979635875887987628187121141743814227340405786877075540 7
001362279198186340074255791261365831560129333487474491021498159626478638347055767141 7
901506957598984400050949734076123012925464888066424970799677282484257435855853348645 6
993617018144080838058452833163022395716238804463454122374136551392458402546135467775 9
729187297731663124277878618877498334677521800812698843489928636349843038102559471536 6
326609578439988831269885578825815480900532753911744090820890535333089139442867815892 1
052069748074205537868136717094006764031023426591318276097353833638375226039787340475 6
849743287469148412627488289694186769477953126400000000000000000000000000000000000000 0
000000000000000000000000000000000000000000000000000000000000000000000000000000000000 0
```

Si el factorial del número solicitado por el usuario tiene más de `NUM_DIGITOS` dígitos, debe informar de que ha ocurrido un desbordamiento:

```
Escriba un número natural: 450
No se ha podido calcular el factorial.
```

Si el usuario introduce un entero negativo, el programa debe informar de que ha cometido un error:


```
Escriba un número natural: -1
El número tiene que ser mayor o igual que 0.
```

Este programa debe ser desarrollado completando el fichero «`factorial-main.cpp`» del directorio «`src`». Tendrás que hacer uso del módulo «`naturales-grandes`», por lo que has de utilizar una cláusula de inclusión `#include "naturales-grandes.hpp"` en el fichero «`factorial-main.cpp`».

Podrás compilar el programa ejecutando el comando `make factorial` en la terminal de Visual Studio Code o ejecutando la tarea "Compilar proyecto «factorial»" del menú Terminal › Run Tasks... . Para ejecutar el programa, puedes ejecutar la orden `bin/factorial` en la terminal de Visual Studio Code una vez que el programa ha sido compilado sin errores, o lanzar la tarea "Ejecutar proyecto «factorial»" desde el menú Terminal › Run Tasks... .

4.3.3. Tarea 3. Potencias de 2

Diseña un programa interactivo que, al ser ejecutado, itere el siguiente tipo de diálogo con el usuario hasta que este responda con un 0 o un número negativo. En cada iteración, el programa ha de pedir 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 trabajar con valores de potencias de 2 de hasta `NUM_DIGITOS` dígitos.

```
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 dígitos.

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 dígitos.

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 dígitos.

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 dígitos.

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 dígitos.

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 dígitos.

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

Este programa debe ser desarrollado completando el fichero «`primera-potencia-main.cpp`» del di-

rectorio «`src`». Tendrás que hacer uso del módulo «`naturales-grandes`», por lo que has de utilizar una cláusula de inclusión `#include "naturales-grandes.hpp"` en el fichero «`primera-potencia-main.cpp`».

Podrás compilar el programa ejecutando el comando `make primera-potencia` en la terminal de Visual Studio Code o ejecutando la tarea "Compilar proyecto «primera-potencia»" del menú Terminal > Run Tasks... . Para ejecutar el programa, puedes ejecutar la orden `bin/primera-potencia` en la terminal de Visual Studio Code una vez que el programa ha sido compilado sin errores, o lanzar la tarea "Ejecutar proyecto «primera-potencia»" desde el menú Terminal > Run Tasks... .

Para resolver este problema, puedes utilizar tanto el procedimiento `multiplicar` como el procedimiento `sumar` ya que para generar potencias de dos, la operación matemática más básica que hace falta es la suma:

$$
\begin{aligned}
1 + 1 = 2 &= 2^1 \\
2 + 2 = 4 &= 2^2 \\
4 + 4 = 8 &= 2^3 \\
8 + 8 = 16 &= 2^4 \\
16 + 16 = 32 &= 2^5 \\
&\dots
\end{aligned}
$$

4.3.4. Tarea 4. Números de Fibonacci

La sucesión de Fibonacci es una sucesión infinita de números naturales cuyo primer término se define como 0 y cuyo segundo término como 1. Los restantes términos son iguales a la suma de los dos que le preceden. Estos son los primeros términos de esta sucesión infinita:

$$0, 1, 1, 2, 3, 5, 8, 13, 21, 34, 55, 89, 144, 233, 377, \dots$$

Diseña un programa interactivo que, al ser ejecutado, itere el siguiente diálogo con el usuario hasta que este responda con un 0 o un número 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 `NUM_DIGITOS` dígitos.


```
Términos inicial y final (0 o negativo para acabar): 1 10
 1.  0
 2.  1
 3.  1
 4.  2
 5.  3
 6.  5
 7.  8
 8.  13
 9.  21
10.  34

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): 200 205
200.  173402521172797813159685037284371942044301
201.  280571172992510140037611932413038677189525
202.  453973694165307953197296969697410619233826
203.  734544867157818093234908902110449296423351
204.  1188518561323126046432205871807859915657177
205.  192306342848094413966711477391830921208052 8


Términos inicial y final (0 o negativo para acabar): 4787 5000
4787.  733434330043152674541324920847148094129588603938388568959179901266062131677939
582370530693595444037449098331644051093821866150303086163055154817629322246843047582 6
259959800604212901428982055681829150950655199287783857015061224226795482474623367665 2
222822568703161170938307328128950976674731050491142708174521074182798839916884500061 7
077050256606303256407531063247473442549068919063525954976794009990508750132092929663 0
972279603414930997444279094039374065272740853520039785786698227588316593293922583225 0
680837133335912300832578909036158039046757151414755357060431422988667920452983711605 6
906566393638335274662765345115546268972543061081580625736713365604040132271786328407 6
339742756245203855814663113490689862210696900867531084265725317796456979714878328669 9
468074437288877978394354634137959129404973167586111684473112137349259886358121516550 6
372413775535583586784737950810233967523823269517389682323406142523826085250814485671 4
319522307480482321929193614716737343149471726333522305539123082795222023
Desbordamiento al calcular el término 4788.


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

Este programa debe ser desarrollado completando el fichero «`fibonacci-main.cpp`» del directorio «`src`».

También tendrás que hacer uso del módulo «`naturales-grandes`», por lo que has de utilizar una cláusula de inclusión `#include "naturales-grandes.hpp"` en el fichero «`fibonacci-main.cpp`».

Puedes compilar el programa de pruebas ejecutando el comando `make fibonacci` en la terminal de Visual Studio Code o ejecutando la tarea "Compilar proyecto «fibonacci»" del menú Terminal » Run Tasks... . Para ejecutar el programa, puedes ejecutar la orden `bin/fibonacci` en la terminal de Visual Studio Code una vez que el programa ha sido compilado sin errores, o lanzar la tarea "Ejecutar proyecto «fibonacci»" desde el menú Terminal » Run Tasks... .

4.3.5. Tarea 5. Números de Lychrel

Dado un número natural *n*, vamos a considerar un proceso aritmético consistente en calcular la imagen especular de *n* y sumársela al propio *n*, proceso que repetiremos con las sumas resultantes 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 iteraciones. Por ejemplo, el número 89 necesita 24 iteraciones del proceso hasta que se convierte en un capicúa[2].

No se ha demostrado todavía que existan los números de Lychrel en base decimal, aunque el número 196 es el menor natural que podría serlo.

Escribe un programa que solicite al usuario un número natural y que muestre el proceso de obtener un número capicúa a través del proceso iterativo ilustrado previamente de sumarlo con su imagen especular y repetir el proceso. El programa debe terminar cuando se obtenga un número capicúa o un resultado que alcance los `NUM_DIGITOS` dígitos.

Se muestran a continuación resultados de la ejecución del programa con varias entradas de usuario:

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

22 no es un número de Lychrel.
```

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

56 no es un número de Lychrel.
```

[2] Wikipedia. «Lychrel number» *Wikipedia, The Free Encyclopedia.* `https://en.wikipedia.org/w/index.php?title=Lychrel_number`. Consultado el 24 de octubre de 2023.


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

58 no es un número de Lychrel.
```

```
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

89 no es un número de Lychrel.
```



```
Escriba un número natural: 196
Iteración 0: 196
Iteración 1: 196 + 691 = 887
Iteración 2: 887 + 788 = 1675
...
Iteración 2390: ...
Iteración 2391: ...
Iteración 2392: 175043768675264878065341640395131982621883398589554455358185726168317
570081228507136616212822234646022779089774443219792043792647337129922504052490757509 1
529361884146326420533770100499850326918609899122069954345931233387039197100433697828 9
355068894061544427198091812310167109530672850807969083108629040703045269759187718467 9
304179329282599842556488974269847600248514501354673699104068160818048835479173421859 8
281591182206195932350411522816911778321959506410189325170493538136991740799941095405 8
487894849988385946890500079382985318363851705328920037148591348760197271261051531306 9
159227218518290572243619746477408289627604010072675432044149519968380625788836661389 9
637202308139308538287818589516503069408268023710697980492670260017521022291907828134 4
625939995956289377063440007029307832222495334500691220098969086340579050009684349246 2
374148716293510157470843594042300218326473973392088133334878909861207553422283125165 4
171571229007462475261857295344544608588339813627912160403525247088737356587833957 + 7
593387856537378807425253040612197263189338858064454435927581625742647009221751714561 5
213822243557021689098784333318802933793746238120032404953480747510153926178414732642 9
434869000509750436809698900221960054335942222387039207000443607739826595999395264431 8
287091922201257100620762940897960173208628049603056159858187828358039318032027369983 1
666388875260838699159414402345762700104067269828047746479163422750928158127229519603 1
351501621727910678431958417300298235071583638135892839700050986495838899484987848504 5
901499970471996318353940715239810146059591238771196182251140532395916022811951828958 1
243719745388408180618604019963764531054158420067489624798846552489952829239714039764 8
177819579625403070409268013809697080582760359017610132181908917244451604988605539828 7
963340017919307833321395434599602219989068196230589940010773350246236414881639251905 7
570942504052299217337462973402979123444779809772206464322282126166317058221800757138 6
162758185355445598589338812628913159304614356087846257686734057 1 = 934382554329002758
807866944456351708940817284395999898950943888742582271003403678592768351044670216239 6
700776177640782138173010971833024655358729823261069219797256195907148824601055973546 9
501559890134167049770535345725743126710486977522720146688335879876100690110343227381 1
573830225978757081519490953673360686834106600204832359825320195981592203777268782345 9
184265852481230070014474085909852630027080764936910974039051581562458654277400960185 6
264154367941317167588632991949592013780045096036423778433487732364059054997841029485 0
190325885772702149774450373647215909377245685526607615084137011863846808172003614900 9
581364420971032074258573372064327687377730218628949201262795432850200660143857705337 7
350094816180766878632027385019362234411099700057897864388565102722566968401862223861 6
543644968049671341998965104864647845011741785170860156368802187007331802686345652923 9
170110370742187936778267700758327219664510438682858773934090831763368894391488990005 9
448172793991704375344966880796719993345517452 8

196 podría ser un número de Lychrel.
```

Como la escritura de los números naturales grandes en la pantalla es lenta (sobre todo, si se utiliza la terminal integrada en el Visual Studio Code), **el programa «`lychrel`» debe mostrar los números que se suman en cada iteración y el resultado de la suma solo cuando los sumandos tengan menos de 20 dígitos o cuando tengan `NUM_DIGITOS` dígitos. En el resto de los casos, debe escribir**

solo el número de la iteración seguido de unos puntos suspensivos (aunque, evidentemente, el programa siga calculando imágenes especulares, sumando y comprobando si el resultado es un número capicúa). Se muestra un ejemplo a continuación:

```
...
Iteración 39: 3603815405135183953 + 3593815315045183063 = 7197630720180367016
Iteración 40: 7197630720180367016 + 6107630810270367917 = 13305261530450734933
Iteración 41: ...
Iteración 42: ...
...
```

Este programa debe ser desarrollado en el fichero de nombre «`lychrel-main.cpp`» del directorio «`src`». Como en los dos problemas anteriores, tendrás que hacer uso del módulo «`naturales-grandes`».

Puedes compilar el programa de pruebas ejecutando el comando `make lychrel` en la terminal de Visual Studio Code o ejecutando la tarea "Compilar proyecto «lychrel»" del menú Terminal » Run Tasks... . Para ejecutar el programa, puedes ejecutar la orden `bin/lychrel` en la terminal de Visual Studio Code una vez que el programa ha sido compilado sin errores, o lanzando la tarea "Ejecutar proyecto «lychrel»" desde el menú Terminal » Run Tasks... .

4.4. Entrega de la práctica

Antes del **sábado 15 de noviembre a las 18:00**, se deberán haber subido a Moodle el fichero «`naturales-grandes.cpp`» y **al menos tres** de los ficheros correspondientes a los módulos principales de los programas solicitados en las tareas 2, 3, 4 y 5 («`factorial-main.cpp`», «`primera-potencia-main.cpp`», «`fibonacci-main.cpp`» y «`lychrel-main.cpp`»).

Si has realizado los programas de todas las tareas, puedes entregar también los ficheros correspondientes, ya que contribuirán a la calificación del apartado «Pruebas con carácter voluntario» establecido en la guía docente de la asignatura.

El fichero «`naturales-grandes.hpp`» no hay que subirlo a Moodle, puesto que no debe ser modificado (ni añadiendo funciones en el mismo, ni modificando las cabeceras de las ya existentes). Si lo modificas, es muy probable que el resultado de la corrección que realicemos los profesores concluya con un error de compilación en todos y cada uno de los programas correspondientes a esta práctica, lo que acarreará una calificación de 0.

