Programación Orientada a Objetos

Práctica 0: Clases, objetos y excepciones

Implementación de clases de utilidad para la gestión de la librería

Versión 3.3

1. Se trata de hacer una clase para trabajar con fechas. Esta clase se llamará *Fecha* y sus atributos serán 3 enteros que representarán, por este orden, día, mes y año.

Una *Fecha* se podrá construir:

a) Con 3 parámetros, que serán, por este orden: el día, el mes y el año.

b) Con 2 parámetros; que serán, por este orden: el día y el mes, siendo el año el de la fecha del sistema.

c) Con un parámetro, el día, tomándose el mes y el año de la fecha del sistema. Pero no se permitirá la conversión implícita de un entero a una *Fecha*.

d) Sin parámetros, tomando los valores de la fecha del sistema.

e) A partir de otra *Fecha*.

f) A partir de una cadena de caracteres de bajo nivel en el formato `"dd/mm/aaaa"`, siendo *dd* el día expresado con 1 o 2 dígitos, *mm* el mes expresado con 1 o 2 dígitos y *aaaa* el año expresado con 4 dígitos (todos los dígitos en base 10). Se permiten conversiones de una cadena de caracteres en este formato a una *Fecha*.

Un valor 0 para día, mes o año no será considerado incorrecto, sino que en ese caso se tomará el valor correspondiente de la fecha del sistema. Ejemplos:

```
Fecha f(3, 0, 2012);     // 3 de marzo (mes en curso) de 2012.
Fecha hoy(0, 0, 0);      // Como 'Fecha hoy;': la fecha del sistema de hoy.
Fecha g("0/0/2015");     // Día y mes del sistema del año 2015.
```

Los constructores deben comprobar que las fechas que se van a construir sean correctas, es decir:

a) Que el día esté comprendido entre 1 y el número de días del mes, y

b) que el mes esté comprendido entre 1 y 12, y

c) que el año esté comprendido entre dos constantes que se definirán con los nombres *Fecha::AnnoMinimo* y *Fecha::AnnoMaximo* con valores respectivos 1903 y 2037. Estas dos constantes serán públicas.

En caso de que esto no suceda, el constructor correspondiente elevará una excepción del tipo *Fecha::Invalida*, que llevará información de porqué ha ocurrido el fallo en forma de una cadena de caracteres de bajo nivel que se le pasará como parámetro al construirla, y que devolverá con un método público llamado *por_que*.

Una *Fecha* podrá incrementarse o decrementarse en 1 día mediante los operadores de incremento o decremento prefijos y sufijos, con la semántica habitual de dichos operadores.

A una *Fecha* podrá sumársele o restársele un número cualquiera de días mediante los operadores de suma y resta de *Fecha* y entero. Estos operadores devolverán otra *Fecha* que será el resultado de la original más o menos el número de días especificado por el operando entero.

Una *Fecha* podrá incrementarse o decrementarse un número cualquiera de días mediante los operadores suma o resta de *Fecha* y entero con asignación.

En los operadores antedichos, habrá que comprobar que la fecha resultante de la operación sea válida, ya que podría sobrepasarse el rango de años. Por ejemplo:

```
Fecha fin_del_mundo(31, 12, Fecha::AnnoMaximo),
      big_bang(1, 1, Fecha::AnnoMinimo);
fin_del_mundo++;     // Error, desbordamiento superior en fin_del_mundo
big_bang - 3;        // Error, desbordamiento inferior en la Fecha devuelta
```

Dos objetos *Fecha* podrán restarse uno de otro, devolviendo el número de días entre las dos fechas como un entero largo con signo.

Una *Fecha* podrá asignarse a otra.

Una *Fecha* poseerá métodos observadores que devolverán los atributos. Estos métodos se llamarán *dia*, *mes* y *anno*.

Una *Fecha* podrá convertirse implícitamente a una cadena de caracteres en el formato `"DIASEM DD de MES de AAAA"`, donde *DIASEM* es el nombre del día de la semana en español, *DD* es el día del mes con 1 o 2 dígitos, *MES* es el nombre del mes en español y *AAAA* es el año expresado con 4 dígitos. Ejemplo: `miércoles 12 de septiembre de 2001`. Obsérvese que en español los nombres de los días de la semana y de los meses se escriben en minúscula, debe colocarse la tilde donde corresponda y los números de los años no llevan separador de miles, de forma que el ejemplo anterior estaría mal expresado como `Miercoles 12 de Setiembre de 2.001`. En el caso de septiembre, debe usarse esta forma y no «setiembre». Esto es importante para que se pasen las pruebas automáticas.

Dos fechas podrán compararse mediante los operadores habituales de igualdad, desigualdad, mayor, menor, mayor o igual, y menor o igual, con el significado lógico de «menor = antes» y «mayor = después».

Deberá hacer el *Makefile* para la compilación. Se recomienda usar las opciones del compilador GNU C++ o clang++ *-Wall* y *-std=c++11* al menos, o sus equivalentes en otro compilador si se emplease otro. El primer objetivo del *Makefile* construirá dos programas de prueba (véase el párrafo siguiente). Habrá un objetivo llamado *clean* que limpiará el directorio de ficheros sobrantes (módulos objeto, respaldos del editor, ejecutables).

Se suministra el código de un programa de prueba semiautomático (*test-consola.cpp*) y otro de pruebas automáticas (*test-auto.cpp*). La clase *Fecha* debe compilarse y enlazarse

contra ellos para producir los dos ejecutables. Conviene estudiar el código del primero de ellos para comprender mejor lo que se pide en este enunciado.

2. Se trata de hacer una clase general para trabajar con cadenas de caracteres (`char`), como una muy pobre imitación de *string* de la biblioteca estándar. Esta clase se llamará *Cadena* y sus atributos serán un puntero a caracteres de tipo *char* y un entero sin signo que representará el tamaño de la cadena o número de caracteres en cada momento.

Una *Cadena* se construirá:

a) Con 2 parámetros, que serán por este orden: un tamaño inicial y un carácter de relleno (V. los ejemplos más adelante).

b) Con 1 parámetro, que será un tamaño inicial; en este caso la cadena se rellenará con espacios. No se permitirá la conversión implícita de un entero en una *Cadena*.

c) Sin parámetros: se crea una *Cadena* vacía, de tamaño 0.

d) Por copia de otra *Cadena*.

e) Por copia de otra sub-*Cadena*; el primer parámetro será la *Cadena* a copiar parcialmente, el segundo será la posición inicial a partir de la cual se empieza a copiar (0 es el primer carácter) y el tercero será la longitud a copiar en número de caracteres. Si se omite, se tomará la constante *Cadena::npos*, de tipo *size_t*, que tendrá que definir adecuadamente, y que valdrá el entero sin signo más grande (truco: -1). Este valor significará aquí: «hasta el final de la cadena».

f) A partir de una cadena de caracteres de bajo nivel, permitiéndose las conversiones desde `const char*` a *Cadena*.

g) Con los n primeros caracteres de una cadena de bajo nivel.

Ejemplos de construcción:

```
Cadena a(3, 'X');          // tamaño y relleno: "XXX"
Cadena b(5);               // tamaño y espacios: "     " (5 espacios)
Cadena c;                  // Cadena vacía, tamaño 0: ""
Cadena d(a);               // copia de Cadena: "XXX"
Cadena f("OLA K ASE?");    // copia de cadena de C: "OLA K ASE?"
Cadena g("OLA K ASE?", 3); // 3 primeros caracteres de cadena de C: "OLA"
Cadena e(f, 1, 2);         // copia de sub-Cadena: "LA"
Cadena h(f, 4);            // ídem, como h(f, 4, Cadena::npos): "K ASE?"
```

Una *Cadena* podrá asignarse a otra. Una cadena de bajo nivel también podrá asignarse directamente a una *Cadena*. La original se destruye.

Una *Cadena* podrá convertirse automáticamente en una cadena de bajo nivel (`const char*`).

La función observadora *length* devolverá el número de caracteres de una *Cadena*.

A una *Cadena* podrá concatenársele otra, añadiéndose esta al final, mediante el operador de suma con asignación.

Dos *Cadena* podrán concatenarse mediante el operador de suma, resultando una nueva *Cadena* que será la concatenación de ambas.

Dos *Cadena* podrán compararse con los operadores lógicos habituales: igualdad, desigualdad, mayor que, menor que, mayor o igual y menor o igual. El resultado será un valor lógico (*booleano*). Que una *Cadena* sea menor que otra significa que está antes en el sistema de ordenación alfabético según los códigos de caracteres. Si son iguales, es que tienen los mismos caracteres en el mismo orden y son de igual longitud.

Podrá obtenerse un carácter determinado de una *Cadena* mediante su índice en ella, para lo que se redefinirá o sobrecargará el operador de índice (corchetes) y una función *at*. La diferencia es que el operador índice no comprobará si el número que se le pasa como operando está dentro del rango de tamaño de la *Cadena*, y la función *at* sí lo hará. En este caso, si el parámetro de *at* no está dentro del rango 0..*length() - 1*, lanzará la excepción estándar *std::out_of_range*. Estas funciones de índice podrán funcionar para *Cadena* definidas `const`, y tanto para asignación como para observación. Es decir, por ejemplo:

```
const Cadena cc("hola");
Cadena c("ola");
cc[0] = ' ';    // ERROR, cc es const
char h = cc[0]; // OK, h <- 'h'
c[0] = 'a';     // OK, c <- "ala"
h = c[1];       // OK, h <- 'l'
```

Cuando una *Cadena* salga fuera de ámbito o se destruya, deberá liberarse la memoria dinámica que pudiera tener reservada.

La función miembro *substr* recibirá dos parámetros enteros sin signo: un índice y un tamaño, y devolverá una *Cadena* formada por tantos caracteres como indique el tamaño a partir del índice. Por ejemplo:

```
Cadena grande("NIHIL NOVVM SVB SOLEM"); // Latín: «Nada nuevo bajo el Sol»
Cadena nuevo = grande.substr(6, 5);  // nuevo <- "NOVVM"
```

La función *substr* deberá lanzar una excepción *std::out_of_range* cuando se proporcione una posición inicial después del último carácter, o cuando la subcadena pedida se salga de los límites de la cadena. Por ejemplo:

```
Cadena s("hola hola");
s.substr(9,  1); // lanza std::out_of_range (9 está fuera del rango 0..8)
s.substr(6, 10); // ídem: después del índice 6 no hay 10 caracteres
```

Obviamente, ya que se está haciendo una imitación de *string*, está prohibido usar dicha clase de la biblioteca estándar, aunque pueden usarse otras funciones de la biblioteca estándar de C++ (que incluye la de C) que se estimen necesarias o convenientes. Tampoco debe usarse *string* en *Fecha*, sino cadenas de caracteres de bajo nivel, como en C. En las prácticas sucesivas, donde hubiera que emplear *string* se usará *Cadena* en su lugar.

Deberá hacer el *Makefile* para la compilación. Se recomienda usar las opciones del compilador GNU C++ o clang++ *-Wall* y *-std=c++11* al menos, o sus equivalentes en otro compilador si se emplease otro. El primer objetivo del *Makefile* construirá dos programas

de prueba (véase el párrafo siguiente). Habrá un objetivo llamado *clean* que limpiará el directorio de ficheros sobrantes (módulos objeto, respaldos del editor, ejecutables).

Se suministra el código de dos programas de prueba: *test-consola.cpp* y *test-auto.cpp*. Conviene estudiar el código del primero para comprender mejor lo que se pide en este enunciado. La clase *Cadena* debe compilarse y enlazarse contra ellos para producir los ejecutables.