Programación Orientada a Objetos

Práctica 0: Clases, objetos y excepciones

Implementación de clases de utilidad para la gestión de la librería

Versión 3.0

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 construirá:

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 en 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 ó 2 dígitos, *mm* el mes expresado con 1 ó 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 en 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. Ejemplo:

```
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.
```

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 *AnnoMinimo* y *AnnoMaximo* con valores razonables, como por ejemplo 1600 y 2200.

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 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 (razonable) 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 (razonable) de días mediante los operadores suma o resta de *Fecha* y entero con asignación.

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, *DD* es el día del mes con 1 ó 2 dígitos, *MES* es el nombre del mes y *AAAA* es el año expresado con 4 dígitos.

Dos fechas podrán compararse mediante los operadores habituales de igualdad, desigualdad, mayor, menor, mayor o igual, y menor o igual.

Deberá hacer el *Makefile* para la compilación. Se recomienda usar las opciones del compilador GNU C++ *-Wall*, *-ansi* y *-pedantic* al menos, o sus equivalentes en otro compilador si Vd. emplea otro. El primer objetivo construirá un programa de prueba. 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. La clase *Fecha* debe compilarse y enlazarse contra él para producir el ejecutable.

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, como en:

```
Cadena a(5, '*');   // "*****"
```

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) A partir de otra *Cadena*.

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

Una *Cadena* podrá asignarse a otra. Una cadena de bajo nivel también podrá asignarse directamente a una *Cadena*.

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

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

A una *Cadena* podrá concatenársele otra, añadiéndose ésta 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..*longitud() - 1*, lanzará la excepción estándar *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 *subcadena* recibirá dos parámetros enteros: 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 novum sub solem");
Cadena nuevo = grande.subcadena(6, 5);  // nuevo <- "novum"
```

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

```
Cadena s("hola hola");
s.subcadena(6,10); // lanza std::out_of_range
```

Deberá hacer el *Makefile* para la compilación. Se recomienda usar las opciones del compilador GNU C++ *-Wall*, *-ansi* y *-pedantic* al menos, o sus equivalentes en otro compilador si Vd. emplea otro. El primer objetivo construirá un programa de prueba. 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. La clase *Cadena* debe compilarse y enlazarse contra él para producir el ejecutable.