Programación Orientada a Objetos

Práctica 1: Biblioteca de E/S

Implementación de los operadores de inserción y extracción de las clases de utilidad

Curso 2023–2024

1. Clase Fecha

Añadir a la clase *Fecha* realizada en la práctica P0 los operadores de inserción y extracción.

Operador de extracción. Para la extracción (lectura), y con el fin de simplificar el ejercicio, se seguirá el formato `DD/MM/AAAA`, donde se utilizarán 1 o 2 dígitos decimales para el día (`DD`) y el mes (`MM`), y 4 para el año (`AAAA`). Al aplicar el operador de extracción se reemplazarán los antiguos contenidos de la *Fecha*. Ejemplo de lectura de dos objetos de tipo *Fecha*:

```
1 void fu(Fecha& f1, Fecha& f2)
2 {
3     std::cin >> f1 >> f2;
4     std::cout << f1 << '\n' << f2 << std::endl;
5 }
```

Si la entrada estándar es:

```
17/07/1982
04/9/2019
```

la salida de la línea 4 será como en los ejemplos de la inserción (siguiente página).

Si la entrada no es correcta, el flujo de entrada debe quedar marcado con el estado de fallo (*fail*) y el operador debe lanzar la excepción *Fecha*::*Invalida*. Ejemplo:

```
1   Fecha f;   // fecha de hoy
2   try {
3     std::cin >> f;
4     std::cout << "OK, la fecha es: " << f << std::endl;
5   } catch(const Fecha::Invalida& e) {
6       if (!std::cin.fail())
7         std::cerr << "Mal hecho, cin debe estar en fallo" << std::endl;
```

```
8         std::cerr << e.por_que() << std::endl;
9         std::cin.clear();
10        if (f != Fecha())
11          std::cerr << "Mal hecho, la fecha no se ha leído bien y debe ser "
12                       "la de hoy." << std::endl;
13    }
```

PISTA: Como el formato pedido para la entrada coincide con el pedido para el constructor de conversión de *Fecha* desde una cadena de bajo nivel, la forma más fácil de hacer este ejercicio es leer una cadena de bajo nivel con el operador de extracción normal y convertirla a *Fecha*.

Operador de inserción. Para la inserción de una *Fecha* (salida), se utilizará el formato `NDS DD de NM de AAAA`, siendo `NDS` el nombre del día de la semana, `DD` el día del mes, `NM` el nombre del mes y `AAAA` el año expresado con 4 dígitos decimales. Los nombres estarán en español y, como es preceptivo, se respetarán las tildes donde correspondan y no se pondrá un punto como separador de miles en el año. Se evitarán las formas vulgares o desusadas *setiembre* y *otubre*. Ejemplos:

```
sábado 17 de julio de 1982
miércoles  4 de septiembre de 2019
```

En el último caso estaría 3 veces mal:

```
miercoles  4 de setiembre de 2.019
```

Para evitar problemas de ambigüedad, se retirará el antiguo operador de conversión implícita a **const char***, que pasará a ser un método explícito de conversión a **const char*** llamado *Fecha*::*cadena*().

2. Clase Cadena

Añadir a la clase *Cadena* realizada en la práctica anterior los operadores de inserción y extracción, más iteradores y semántica de movimiento (constructor y operador de asignación).

Operador de inserción. La *Cadena* se insertará en un flujo sin cambios. Gracias a este operador, deja de ser necesaria la conversión a **const char*** para insertar una *Cadena* en un flujo.

Operador de extracción. Una *Cadena* se podrá extraer (leer) de un flujo. Para ello, se leerá una palabra y se reemplazará con ella el contenido que tuviera la *Cadena* hasta ese momento.

Se entenderá por «palabra» una sucesión de caracteres que no sean espacios en blanco (es decir, que *isspace*(*c*) devuelva un valor distinto de cero). A la hora de manipular una palabra, hay que seguir estos pasos:

1. Buscar el inicio de la palabra, leyendo caracteres hasta encontrar uno que no sea un espacio en blanco. Si se llega al final de la entrada antes de que esto ocurra, la cadena quedará vacía.

2. Introducir caracteres en la cadena hasta que se encuentre un espacio en blanco o se llegue al final de la entrada o de la *Cadena*. Para simplificar el ejercicio, la *Cadena* **tendrá un tamaño máximo de 32 caracteres**, a lo cual habrá que sumar el terminador nulo (`'\0'`).

Es importante que el espacio en blanco que marca el final de la palabra **no** se extraiga del flujo de entrada, para que la próxima lectura realizada desde otro punto del programa obtenga el resultado esperado. Por ejemplo, si en la entrada se tiene el contenido «hola␣adiós» y ejecutamos este código:

```
1  Cadena c;
2  std::cin >> c;
3  std::cout << "Palabra 1: " << c << "\nCter. actual: '"
4            << static_cast<char>(std::cin.peek()) << "'" << std::endl;
5  std::cin >> c;
6  std::cout << "Palabra 2: " << c << std::endl;
```

Debería obtenerse esta salida:

```
Palabra 1: hola
Cter. actual: ' '
Palabra 2: adiós
```

PISTA: Todo lo dicho anteriormente coincide con el comportamiento normal del operador de extracción. Por lo tanto, la forma más fácil de hacer este ejercicio es usarlo para leer una cadena de bajo nivel y convertirla luego a *Cadena*.

Iteradores. Defina iteradores para esta clase, como los de los contenedores de la STL, y las funciones relacionadas *begin*() y *end*():

- Tipos miembro de *Cadena*:
  - *iterator*
  - *const_iterator*
  - *reverse_iterator*
  - *const_reverse_iterator*
- Funciones miembro de *Cadena*:
  - *begin*()
  - *end*()
  - *cbegin*()

- *cend*()
- *rbegin*()
- *rend*()
- *crbegin*()
- *crend*()

Tenga en cuenta que las funciones (*r*)*begin*/(*r*)*end* deben estar sobrecargadas para trabajar con objetos de *Cadena* tanto constantes como modificables.

Para ejemplos de uso, vea el código del programa de pruebas no automáticas.

PISTA: Las definiciones de los iteradores directos son triviales en este caso. Para hacer los iteradores inversos hay que incluir la cabecera estándar `iterator` y definirlos así (dentro de la clase, evidentemente):

```
typedef std:: reverse_iterator <iterator>       reverse_iterator ;
typedef std:: reverse_iterator <const_iterator> const_reverse_iterator ;
```

Las funciones (*c*)*rbegin*/(*c*)*rend* se definen construyendo objetos a partir de los iteradores directos, pero con los del extremo opuesto:

```
inline Cadena:: reverse_iterator  Cadena::rbegin() noexcept
{
    return  reverse_iterator (end());
}

inline Cadena:: reverse_iterator  Cadena::rend() noexcept
{
    return reverse_iterator(begin());
}
// ...
```

Análogamente las versiones **const** sobrecargadas y las explícitas con la *c* delante.

Semántica de movimiento. Respecto a la semántica de movimiento, debe tener en cuenta que la *Cadena* inicial debe quedar en un estado válido tras la operación. Por ello, la *Cadena* original pasará a ser la *Cadena* vacía una vez realizado el movimiento.