Programación Orientada a Objetos

Grado en Ingeniería Informática

Examen de evaluación continua

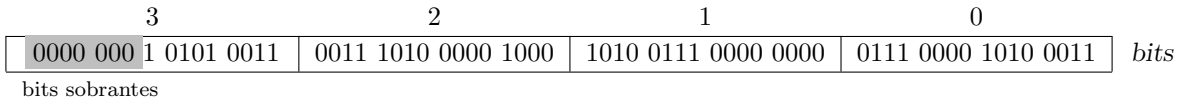
Curso 2019-20 **Viernes 15 de mayo de 2020**

Sea una clase llamada *Binario* que almacena un número binario de longitud arbitraria, representado por la secuencia de bits que lo componen (dígitos 0 y 1). La longitud de un objeto *Binario* se establecerá en el momento de la construcción y permanecerá fija hasta que el objeto sea destruido.

La secuencia de bits de un *Binario* se almacena de menos a más significativo en el atributo llamado *bits*, un vector de bajo nivel de **unsigned int**. Cada elemento del vector *bits* contiene hasta **sizeof**(**unsigned int**) bytes de la secuencia. Los bits más significativos del último elemento del vector que sobren valdrán 0 y se ignorarán. El tamaño del vector (m) depende de la longitud en bits (n) del número binario y viene dado por la expresión $m = (n + bits\_elto - 1) \;/\; bits\_elto$, donde la constante *bits_elto* corresponde al número de bits de **unsigned int**.

Ejemplo: Representación de un *Binario* de $n = 57$ bits, suponiendo que **sizeof**(**unsigned int**) = 2 bytes = 16 bits = *bits_elto*, almacenado en el atributo *bits*, un vector de bajo nivel de $m = (n + bits\_elto - 1) \;/\; bits\_elto = (57 + 16 - 1)/16 = 4$ **unsigned int** en el que sobran los 7 bits más significativos.

Binario = 1 0101 0011 0011 1010 0000 1000 1010 0111 0000 0000 0111 0000 1010 0011




```
class Binario {
public:
   refBit operator[](size_t i);
   bool bit(size_t i) const;
   // ...
private:
   static const size_t bits_elto = CHAR_BIT * sizeof(unsigned); // CHAR_BIT = bits por byte
 const size_t n;    // longitud en bits
 const size_t m;    // longitud del vector de enteros
 unsigned* bits;    // vector de enteros
};
```

Asuma que la clase *Binario* ya dispone de dos métodos de acceso a los bits individuales y de un tipo auxiliar, *refBit*, que representa una referencia a un bit. El método *Binario*::**operator** [ ] devuelve por referencia el bit i-ésimo de un *Binario*, representada por un objeto *refBit*. Y el método *Binario*::*bit*() devuelve un **bool** que representa el valor del bit i-ésimo, 0 (**false**) y 1 (**true**).

1. Complete la clase *Binario* con la declaración de los mínimos métodos imprescindibles para que las siguientes instrucciones proporcionen los resultados descritos en los comentarios. **1 p**

```
Binario b1(4);              // b1 = 0000
Binario b2("10100");        // b2 = 10100
Binario b3;                 // b3 = 0
size_t  n1 = b1.n_bits();   // n1 = 4
size_t  n2 = b2.unos();     // n2 = 2 (10100 tiene 2 dígitos 1)
```

2. Implemente los métodos declarados en el apartado anterior, teniendo en cuenta que habrá que lanzar una excepción de tipo *std*::*invalid_argument* en el caso de que se trate de construir un número binario con algún dígito distinto de 0 o 1. **3 p**

Suponga que el **operator** = está sobrecargado para el tipo *refBit*, de forma que usándolo junto a *Binario*::**operator** [ ] se puede modificar el valor de un bit individual. Por ejemplo:

```
Binario bin{8};
bin[7] = bin[2] = bin[1] = 1; bin[0] = 0;    // bin = 10000110
```

3. ¿Es válido el comportamiento por defecto del constructor de copia de *Binario*? En caso afirmativo explique por qué, de lo contrario implemente el método. **1 p**

4. Escriba un fragmento de código en el que se cree un objeto *Binario* a partir de una cadena de caracteres y seguidamente consulte un bit individual del mismo. Capture todas las posibles excepciones que podrían lanzarse e imprima un mensaje explicativo asociado a cada excepción por el flujo de salida estándar de error, teniendo cuenta que el método *Binario*:: *bit* lanza una excepción estándar del tipo *std*:: *out_of_range*. **1 p**

5. Un número Binario se podrá insertar en un flujo de salida, de forma que, por ejemplo, el siguiente código: **1 p**

Binario b2(`"10100"`);
std:: *cout* << *b2* << *std*::*endl*;

produzca la salida

```
El número tiene 2 bits con valor 1 y 3 con valor 0
```

Complemente la clase *Binario* con lo necesario para obtener el comportamiento anterior; para ello básese en los métodos *Binario*:: *n_bits* y *Binario*:: *unos*.

6. Sobrecargue el **operator** & (operador *AND* bit a bit) para la clase *Binario*. La longitud del resultado coincidirá con la del operando más largo (piense que el más corto es como si tuviera ceros a la izquierda). **1 p**