Abril-2008.pdf

fluxneon

Programación Orientada a Objetos

2º Grado en Ingeniería Informática

Escuela Superior de Ingeniería
Universidad de Cádiz

Programación Orientada a Objetos

Examen de evaluación continua

Curso 2007-08 **Jueves 24 de arbil de 2008**

1. Se desea implementar en C++ una clase para almacenar fragmentos de ADN. Esta clase se denomina *ADN* y su representación interna está basada en dos atributos: un puntero, c, y un tamaño, n. El atributo c apuntará a una zona de memoria dinámica creada con **new** que será destruida con **delete** tan pronto como deje de ser necesaria. Dicha zona contendrá la secuencia de nucleótidos de una de las hélices. El atributo n representará la longitud de la secuencia y se permitirá la existencia de secuencias vacías, que tendrán longitud 0. **4,5 p**

Los nucleótidos se definen mediante un tipo enumerado, *Nucleotido*, formado por las constantes enumeradas A (adenina), C (citosina), G (guanina) y T (timina), en dicho orden.

a) Defina externamente el constructor predeterminado para crear objetos *ADN* vacíos. No emplee asignaciones. **0,5 p**

```
ADN::ADN(): c(0), n(0)
{}
```

b) Defina externamente un constructor que cree un objeto *ADN* a partir de una secuencia de nucleótidos, ya creada con **new**, representada por dos parámetros: un puntero y un tamaño. Utilice para los parámetros idénticos nombres a los indicados para los atributos. No emplee asignaciones. Este constructor será privado. **0,5 p**

```
ADN::ADN(Nucleotido* c, size_t n): c(c), n(n)
{}
```

c) Defina externamente un constructor que cree un objeto *ADN* con la secuencia de nucleótidos correspondiente a una cadena de bajo nivel de caracteres constantes, que recibirá como único parámetro y que sólo contendrá caracteres `'A'`, `'C'`, `'G'` y `'T'`. **1,0 p**

```
ADN::ADN(const char* s)
{
    n = strlen(s);
    c = new Nucleotido[n];
    for (size_t i = 0; i < n; ++i)
        switch (s[i]) {
            case 'A': c[i] = A; break;
            case 'C': c[i] = C; break;
            case 'G': c[i] = G; break;
            case 'T': c[i] = T; break;
        }
}
```

d) ¿Es necesario definir un constructor de copia si queremos copiar objetos *ADN* con la semántica habitual que poseen los tipos primitivos? ¿Por qué? **0,5 p**

Sí. Porque, de lo contrario, el constructor de copia por omisión copiaría un objeto, pero atributo a atributo, lo que no es conveniente en la mayoría de las aplicaciones. Los punteros internos del objeto original y de su copia acabarían apuntando a la misma zona de memoria, que compartirían a todos los efectos. Si uno de ellos fuera modificado, el cambio se reflejaría automáticamente en el otro, que perdería así su identidad. Más grave aún, al morir uno de los objetos se destruiría la zona común, dejándola inservible para el otro, lo que puede provocar graves problemas durante la ejecución.

e) Sobrecargue externamente el operador de suma para devolver la concatenación de dos objetos *ADN*. Suponga que ha sido declarado como amigo. Utilice el constructor más apropiado para crear el objeto temporal que devolverá como resultado, así como la función *memcpy*() de la biblioteca estándar (recuerde que posee tres parámetros: destino, origen y número de *bytes*). No empleee bucles. **1,0 p**

```
ADN operator +(const ADN& a, const ADN& b)
{
    const size_t n = a.n + b.n;
    ADN t(new Nucleotido[n], n);
    memcpy(t.c, a.c, a.n * sizeof(Nucleotido));
    memcpy(t.c + a.n, b.c, b.n * sizeof(Nucleotido));
    return t;
}
```

f) Reescriba el siguiente fragmento de código colocando explícitamente las llamadas a los constructores implicados: **1,0 p**

```
const char* const s = "GA";
ADN a(s), b = a;
b[0] = C;
(a + "TTA" + b).mostrar();
```

```
const char* const s = "GA";
ADN a = ADN(s), b = ADN(a);
b[0] = C;
ADN(ADN(a + ADN("TTA")) + b).mostrar();
```

2. Determina los errores que produce el siguiente código, indicando el tipo (de compilación o ejecución), la línea en la que está, la causa por la que se produce y una posible solución. **4 p**

```
#include <iostream>
#include <cmath>
using namespace std;

class Complejo {
    double real, imag;
public:
    explicit Complejo (double r = 0., double i = 0.): real(r), i(imag) {}
    Complejo (const Complejo& c) {this = c;}
    static void print() { cout << "(" << real << "," << imag << ")"; }
    operator double() const { return sqrt(real * real + imag * imag); }
};

Complejo operator +(const Complejo& c1, const Complejo& c2) {
    c1.real += c2.real, c1.imag += c2.imag;
    return Complejo(c1.real, c1.imag);
}
Complejo& operator -(Complejo c1, Complejo c2) {
    c1.real -= c2.real, c1.imag -= c2.imag;
    return Complejo(c1.real, c1.imag);
}
const Complejo operator *(Complejo& c1, Complejo& c2) {
    return Complejo(c1.real * c2.real -  c1.imag * c2.imag,
```

PARA TI ESTE PORTÁTIL ES GENIAL

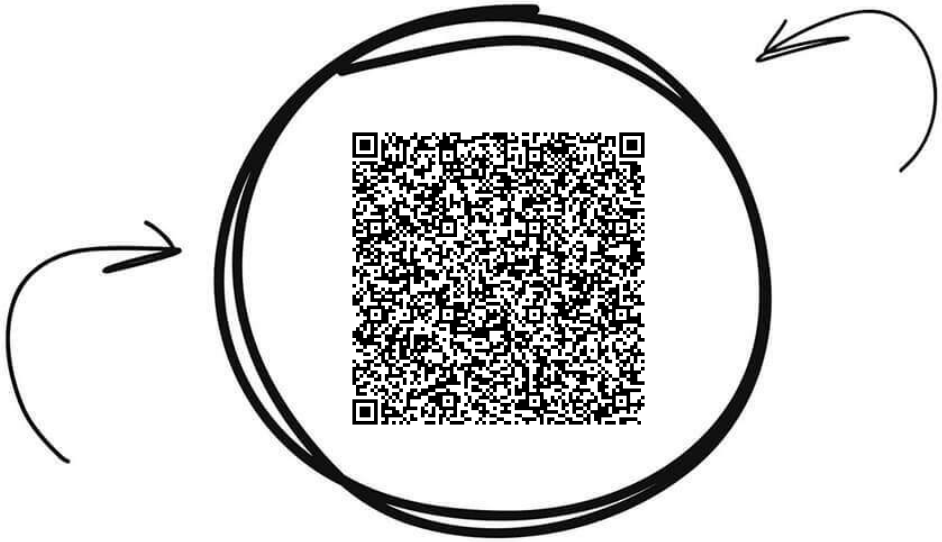
Para tu padre... está en oferta

Pásale este QR
a los Reyes magos...
o a tu padre

GAMING - MSI Stealth 16 Studio

Queridos Reyes Magos, este año he sido excepcionalmente bueno, y tengo en mente un regalo que podría hacer que mis tareas escolares se conviertan en épicas batallas de conocimiento. Me encantaría recibir un portátil ultraligero que, aunque suene a ciencia ficción, esté equipado con todo lo que un amante de los videojuegos como yo podría desear: pantalla UHD+, tarjeta gráfica hasta RTX4070, procesador i9, ¡y un peso menor a 2kg para llevarlo a donde sea! el color no me importa, podéis elegeri en blanco o azul marino.

Programación Orientada a Obj...



Banco de apuntes de la

WUOLAH

Comparte estos flyers en tu clase y consigue más dinero y recompensas

1. Imprime esta hoja
2. Recorta por la mitad
3. Coloca en un lugar visible para que tus compis puedan escanar y acceder a apuntes
4. Llévate dinero por cada descarga de los documentos descargados a través de tu QR

```
                    c1.real * c2.imag + c2.real * c1.imag);
}

int main() {
    Complejo a(3.), b(2., 2.), c = a + b, d = b + 3.;
    double e = 3. + b;
    cout << "a * b = ", (a * b).print();
}
```

```
#include <iostream>
#include <cmath>

using namespace std;

class Complejo {
    double real, imag;
public:
    explicit Complejo (double r = 0., double i = 0.):
// real(r), i(imag) {}
        real(r), imag(i) {}
    Complejo (const Complejo& c) // {this = c;}
        {}
    void print() {
// static void print() {
        cout << "(" << real << "," << imag << ")";
    }
    operator double() const { return sqrt(real * real + imag * imag); }

// Anadido

    friend Complejo operator +(const Complejo& c1, const Complejo& c2);
    friend Complejo operator -(Complejo c1, Complejo c2);
    friend const Complejo operator *(Complejo& c1, Complejo& c2);
};

Complejo operator +(const Complejo& c1, const Complejo& c2)
{
/* c1.real += c2.real;
    c1.imag += c2.imag;
    return Complejo(c1.real, c1.imag);
*/
    return Complejo(c1.real + c2.real, c1.imag + c2.imag);
}

//Complejo& operator -(Complejo c1, Complejo c2)
Complejo operator -(Complejo c1, Complejo c2)
{
    c1.real -= c2.real, c1.imag -= c2.imag;
    return Complejo(c1.real, c1.imag);
}

const Complejo operator *(Complejo& c1, Complejo& c2)
{
    return Complejo(c1.real * c2.real - c1.imag * c2.imag, c1.real * c2.imag + c2.real * c1.imag);
```

```
}

int main()
{
    Complejo a(3.), b(2., 2.), c = a + b;
    //Complejo d = b + 3.;
    double e = 3. + b;

    // cout << "a * b = ", (a * b).print();
    cout << "a * b = ", (Complejo(a * b)).print();
}
```

3. Contesta verdadero (V) o falso (F) a las siguientes afirmaciones. Un acierto suma 0,1 y un fallo resta un acierto. **1,5 p**

- Un método const puede modificar un atributo mutable solamente si el objeto al que pertenecen no es constante. (F)
- Un método observador que devuelva una referencia no constante a un atributo privado permite violar el principio de ocultación de información. (V)
- Una función amiga de una clase C sólo tiene acceso a los miembros privados de C. (F
- Una función amiga de una clase C no tiene acceso al puntero this. (V)
- El puntero this no es modificable. (V)
- Se pueden definir clases sin escribir ningún constructor. (V)
- Al inicializar un objeto se llama al constructor de copia, si existe, y si no, al operador de asignación. (F)
- El único parámetro de un constructor de copia puede ser una referencia **const** o no **const**. (V)
- El único parámetro de un constructor de copia se puede pasar por valor, aunque a veces es obligatorio pasarlo por referencia. (F)
- El único parámetro del operador de asignación se puede pasar siempre por valor. (V)
- La declaración de un constructor con dos o más parámetros, todos con un valor por defecto definido, en realidad declara dos constructores: el predeterminado y uno de conversión. (F)
- C++ permite sobrecargar un operador dándole un significado diferente al que por naturaleza le corresponde. (V)
- La sobrecarga externa de un operador para una clase requiere la definición de un operador de conversión. (F)
- No se pueden sobrecargar operadores en la parte privada de una clase. (F)
- Siempre que se defina un constructor para una clase hay que definir al menos un destructor. (F)