Examen de las asignaturas IG17/II17/F30/E28
Ingenierías Informáticas - Universitat Jaume I

20 de Enero de 2004

Normativa

- **Duración del examen: 3.5 horas.**
- No está permitido consultar libros, apuntes, resúmenes, etc.
- Utiliza hojas diferentes para la solución de cada uno de los problemas.
- Puedes pedir hojas adicionales si completas las que se te han entregado.
- Empieza poniendo tu nombre, apellidos y DNI con letra clara en la parte superior de cada hoja.
- Tacha claramente aquello que entregues y no desees que se tenga en cuenta en la evaluación.

Cuestiones

1. **(0,75 puntos)** Señala qué está mal, y por qué, en la siguiente definición de los operadores << y >>:

```
ostream & operator << (ostream &canal, const Complejo &z) const
{
   cout << '(' << z.Real() << ", " << z.Imaginaria() << ')';
}

istream & operator >> (istream &canal, const Complejo &z) const
{
   cin >> '(' >> z.real >> ", " >> z.imaginaria >> >> ')';
}
```

   suponiendo que la implementación de la clase `Complejo` es la siguiente:

```
class Complejo {
   float real, imaginaria;
 public:
   Complejo(float r, float i) : real(r), imaginaria(i) {}
   float Real() const {return real;}
   float Imaginaria() const  {return imaginaria;}
};
```

2. **(0,75 puntos)** Considera la siguiente declaración de una clase `Cola`:

```
class Cola {
   struct Nodo {
      int info;
      Nodo *sig;
      ...
   };
   Nodo *cap, *cua;
 public:
   ...
};
```

Cada nodo de la cola contiene un campo de información de tipo entero y un puntero al siguiente nodo de la cola. La cola se maneja mediante un puntero `cap` que apunta al primer nodo y un puntero `cua` que apunta al último nodo.

Un programador experto ha implementado el operador de asignación para esta clase como sigue:

```
Cola & Cola::operator= (const Cola & c) {
   if (this != &c) {
      LiberarCola();
      CopiarCola(c);
   }
   return *this;
}
```

¿Qué peligro potencial existiría si no se comprobase la condición `(this != &c)`? ¿Qué diferencia existiría entre comprobar `(this != &c)` y comprobar `(*this != c)`?

3. **(2 puntos)** Considera la siguiente implementación (incompleta) de una clase `Matriz`:

```
class Matriz {
   int filas;
   int columnas;
   int **valores;
 public:
   Matriz(int, int);
   Matriz operator+ (const Matriz &m) const;
};
Matriz::Matriz(int m, int n) : filas(m), columnas(n) {
   valores = new int* [filas];
   for (int i=0; i<filas; ++i)
        valores[i] = new int [columnas];
}
```

Esta implementación permite representar matrices de números enteros de modo tal que, internamente, `valores[i][j]` es el elemento de la fila i-ésima y la columna j-ésima de la matriz.

a) Implementa el destructor de esta clase.

b) Modifica la declaración de la clase `Matriz` y la implementación del constructor y del destructor para convertirla en una clase genérica, en la que los elementos de la matriz puedan ser de un tipo T cualquiera.

c) Supongamos que disponemos también de la clase `Complejo` para representar números complejos y de la clase `Polinomio` para representar polinomios, y que en un programa queremos hacer uso de tu matriz genérica para trabajar con una matriz cuyos elementos son números complejos y con una matriz cuyos elementos son polinomios. Indica cómo se tendrían que declarar esas matrices y qué ficheros necesitaríamos incluir para ello.

d) Implementa en tu matriz genérica la función `operator+`, suponiendo que las dos matrices a sumar son del mismo tamaño, que no están vacías, y que la suma de dos matrices se realiza siempre componente a componente, igual que la suma de matrices de números enteros, sea cual sea el tipo de los elementos de la matriz. Indica qué operaciones asumes que soporta el tipo `T`.

e) Otro programador que necesita hacer sumas de matrices cuyos elementos son complejos y sumas de matrices cuyos elementos son polinomios ha decidido utilizar la herencia en vez de la genericidad para obtener las clases derivadas `MatrizComplejos` y `MatrizPolinomios` a partir de la clase base `Matriz`. ¿Qué te parece este planteamiento alternativo? Razona tu respuesta.

Problemas

Si lo deseas puedes hacer uso de la *Standard Template Library (STL)* para resolver el problema 2, pero no puedes utilizarla para resolver el problema 1.

1. **(3 puntos)** Implementa en C++ una clase `Polinomio` que permita representar y trabajar con polinomios de una variable con coeficientes reales:

$$Q(x) = a_0 + a_1 \cdot x + a_2 \cdot x^2 + \cdots + a_n \cdot x^n$$

cuya derivada es:

$$Q'(x) = a_1 + 2 \cdot a_2 \cdot x + 3 \cdot a_3 \cdot x^2 + \cdots + n \cdot a_n \cdot x^{n-1}$$

Los coeficientes del polinomio $a_0, a_1, \ldots, a_n$ se deben guardar en un vector unidimensional. La memoria necesaria para el vector se debe reservar y liberar dinámicamente. Tu implementación debe incluir todos los constructores, destructores, operadores y demás funciones que intervienen en el siguiente programa de prueba:

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

#include "Polinomio.h"

int main () {
   float a[] = {1.2,-7,0,9}; // Coeficientes a[0]=1.2, a[1]=-7, a[2]=0, a[3]=9
   Polinomio p(a, 4); // Inicializa p con los 4 coeficientes que contiene a

   Polinomio p2 = p.Derivada();

   if (p * 7.7 != 7.7 * p)
      cout << "Algo no funciona bien." << endl;

   if (p * 5 == p)
      cout << "Algo no funciona bien." << endl;

   return 0;
}
```

El producto de un polinomio por un valor real r se realiza multiplicando por r cada uno de los coeficientes del polinomio. Y dos polinomios se consideran iguales si y solo si todos sus coeficientes coinciden.

2. **(3,5 puntos)** Supongamos que, en un sistema operativo muy simple:

   - El descriptor de un fichero contiene (1) una cadena con el nombre del fichero, (2) su tamaño en bytes y (3) su fecha de creación.
   - El descriptor de un enlace simbólico contiene (1) una cadena con el nombre del enlace y (2) la dirección del descriptor del fichero al que apunta ese enlace. El tamaño en bytes de un enlace simbólico no se almacena, y se calcula como el número de caracteres del nombre del fichero al que apunta.
   - El descriptor de un directorio contiene (1) una cadena con el nombre del directorio y (2) una lista de descriptores en la que puede haber tanto descriptores de ficheros como descriptores de enlaces simbólicos (no puede contener subdirectorios).

Teniendo en cuenta eso, y **haciendo uso de la herencia** donde resulte apropiado, implementa en C++ las clases y funciones necesarias para que se pueda ejecutar el siguiente programa de prueba:

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

#include "Fecha.h"
#include "SistemaFicheros.h"

int main () {
   Descriptor *f1 = new DescriptorFichero("curriculum", 2500, Fecha(20,1,2004));
   Descriptor *f2 = new DescriptorFichero("mifoto.jpg", 45000, Fecha(27,12,2003));
   Descriptor *e  = new DescriptorEnlaceSimbolico("cvitae", f1);
   DescriptorDirectorio d("/home/luis");

   d.Insertar(f1);
   d.Insertar(f2);
   d.Insertar(e);

   d.MostrarDatos();                cout << endl;
   d.MostrarDatos("curriculum");    cout << endl;
   d.MostrarDatos("cvitae");        cout << endl;
   d.MostrarDatos("notas.pdf");     cout << endl;

   delete f1; delete f2; delete e;
   return 0;
}
```

Tu implementación debe satisfacer también los siguientes requisitos:

- La función `DescriptorDirectorio::Ocupacion()` debe calcular y devolver la suma de los tamaños de todos los ficheros y enlaces simbólicos del directorio.
- La función `DescriptorDirectorio::MostrarDatos()` sin argumentos debe mostrar en la pantalla el nombre del directorio y el resultado de `DescriptorDirectorio::Ocupacion()`.
- La función `DescriptorDirectorio::MostrarDatos(nombre)` debe buscar en el directorio un fichero o enlace con ese nombre, y mostrar sus datos en la pantalla, haciendo uso, gracias a la **ligadura dinámica**, de la función `MostrarDatos()` de la clase a la que pertenezca el descriptor encontrado. Si no lo encuentra, debe indicarlo también.
- La función `DescriptorFichero::MostrarDatos()` debe mostrar en la pantalla el nombre, el tamaño y la fecha de creación del fichero.
- La función `DescriptorEnlaceSimbolico::MostrarDatos()` debe mostrar en la pantalla el nombre del enlace, su tamaño y todos los datos del fichero al que apunta el enlace.

No se pide que implementes el destructor ni el constructor copia de ninguna de las clases, y puedes suponer que los usuarios de la clase no insertarán más de un descriptor con el mismo nombre en el directorio. La salida que debe obtenerse al ejecutar la función `main()` anterior es:

```
{Directorio: /home/luis, Ocupa: 47510}
{Fichero: curriculum, Ocupa: 2500, Fecha: 20/1/2004}
{Enlace: cvitae, Ocupa: 10, Apunta a: {Fichero: curriculum, Ocupa: 2500, Fecha: 20/1/2004}}
No se ha encontrado notas.pdf en el directorio /home/luis
```

La ocupación del directorio es el resultado de sumar 2500, 45000 y 10, siendo 10 la longitud de "`curriculum`", el nombre del fichero al que apunta el enlace simbólico `e`.