Junio-2005.pdf

fluxneon

Programación Orientada a Objetos

2º Grado en Ingeniería Informática

Escuela Superior de Ingeniería
Universidad de Cádiz

Programación Orientada a Objetos

Examen final

Curso 2004–05 **Miércoles 22 de junio de 2005**

1. Indique y dé una breve descripción de al menos cuatro ventajas del enfoque orientado a objetos respecto del enfoque estructurado. Además, describa también cuatro limitaciones propias del enfoque orientado a objetos. **1,0 p**

2. Se desea implementar en C++ una clase que almacene un diccionario. Esta clase se denominará `Diccionario` y su representación interna estará basada en un atributo, una aplicación o `map`. Cada elemento de esta aplicación será un par de palabras (`string`); el primer componente del par será una palabra en un idioma y el segundo será su correspondiente traducción en otro idioma. **4,0 p**

a) Defina un constructor que reciba un nombre de fichero en un `string`. Ese fichero contendrá una serie de líneas compuestas de dos palabras separadas por espacios. El constructor deberá leer cada una de esas líneas e introducir, por cada una de ellas, un par en la aplicación cuyo primer elemento será la primera palabra de la línea y el segundo la segunda palabra de la línea. **0,75 p**

Si se le pasa la cadena vacía como nombre de fichero no añadirá nada a la aplicación.

b) ¿Qué hay que hacer para que dicho constructor sirva además de constructor predeterminado (para la cadena vacía) y para que no se realicen conversiones implícitas con él? **0,25 p**

c) Defina un método `entradas` que devuelva el número de entradas del diccionario. **0,25 p**

d) Defina un método `introduce` que reciba dos `string`, una palabra y su traducción, e introduzca una nueva entrada en el diccionario. **0,25 p**

e) Defina un método `traduccion` que reciba una palabra y devuelve su traducción asociada. Si no se encuentra, devuelve la cadena vacía. **0,50 p**

f) Sobrecargue el operador de índice para poder acceder al par i-ésimo del diccionario. Si no se encuentra, devuelve un par de cadenas vacías. **0,50 p**

Para realizar este operador le será útil la función genérica de la STL `advance` que recibe un iterador y un entero, y hace avanzar el iterador tantas posiciones como indique el entero.

g) A continuación se quiere desarrollar una clase `DiccionarioBilingue` que implemente un diccionario bilingüe a partir de un diccionario simple. Este diccionario bilingüe debe permitir realizar consultas en los dos sentidos. Es decir, si el diccionario bilingüe es de español e inglés, debería dada una palabra inglesa devolver su correspondiente española, y viceversa. **1,5 p**

¿Qué relación podemos establecer entre las clases `Diccionario` y `DiccionarioBilingue` para poder implementar la segunda a partir de la primera? Razone la respuesta y escriba todo lo necesario para implementar dicha relación.

Tenga en cuenta que la clase `DiccionarioBilingue` tiene que tener un constructor, un método `entradas`, un método `introduce` y el operador índice que actúen de la misma forma que los correspondientes en la clase `Diccionario`.

Además, deberá tener dos métodos `traduccionDirecta` y `traduccionInversa` que reciban una palabra y devuelvan su traducción asociada, cada una en un sentido distinto. Si no se encuentra, devuelven la cadena vacía.

3. Observe el siguiente programa: **1,5 p**

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

class Libro {
  char* titulo_;
  int paginas_;
public:
  Libro() : titulo_(new char[1]), paginas_(0) { *titulo_ = 0; }
  Libro(const char* t, int p) : paginas_(p)
  {
    titulo_ = new char[strlen(t) + 1];
    strcpy(titulo_, t);
  }
  ~Libro() { delete [] titulo_; }
  void paginas(int p) { paginas_ = p; }
  int paginas() const { return paginas_; }
  char* titulo() const { return titulo_; }
};


void mostrar(Libro l)
{ cout << l.titulo() << " tiene " << l.paginas() << " páginas" << endl; }

int main()
{
  Libro l1("Fundamentos de C++", 474),
        l2("Por Fin: C ISO", 224),
        l3;
  mostrar(l3);
  l3 = l1;
  mostrar(l1), mostrar(l2), mostrar(l3);
}
```

- Diga si funciona correctamente. En caso afirmativo indique lo que imprime. En caso negativo haga las modificaciones necesarias para que funcione correctamente.

4. Se trata de implementar en C++ una función genérica, *ordenado*(), que decida si los elementos de un rango de iteradores aparecen en un cierto orden. Para lograr la máxima generalidad, los únicos requisitos que la función impondrá es que los iteradores sean de entrada y que la relación de orden tenga las propiedades de un orden estricto débil. También hay que tratar adecuadamente los rangos vacíos, que se suponen ordenados, ya que no contienen elementos. **1,5 p**

a) Escriba una versión que compruebe el orden respecto de una función de comparación arbitraria. **1,0 p**

b) Escriba un fragmento de código que compruebe si un vector de bajo nivel de enteros está en orden ascendente de valores absolutos. Emplee un objeto función. **0,5 p**

5. Observe atentamente el siguiente programa: **2,0 p**

```
#include <iostream>

using namespace std;
```

```
class Objeto
{
public:
    Objeto(char const *nombre): nombre_(nombre) {
        cout << "Constructor de Objeto para " << nombre_ << endl;
    }
    ~Objeto() {
        cout << "Destructor de Objeto para " << nombre_ << endl;
    }
    void lanzamiento() {
        Objeto o("'objeto local de lanzamiento()'");
        cout << "Método lanzamiento() para " << nombre_ << endl;
        throw &o;
    }
    void saludo() {
        cout << "Hola de parte de " << nombre_ << endl;
    }
private:
    char const *nombre_;
};

int main()
{
    Objeto o("'objeto de main()'");

    try {
        o.lanzamiento();
    }
    catch(Objeto *o) {
        cout << "Excepción capturada" << endl;
        o->saludo();
    }
}
```

a) Escriba las líneas que imprimiría el programa **0,5 p**

b) Diga qué anomalía hay **0,5 p**

c) Corríjala reescribiendo el método `Objeto::lanzamiento()` para que lance un objeto creado dinámicamente, o su dirección, y el capturador de la excepción en `main()`. **1,0 p**