Examen de Programación Avanzada
Ingeniería Informática - Universitat Jaume I

3 de febrero de 2003

Normativa

Lee atentamente las siguientes instrucciones antes de empezar:

- **Duración del examen: 3 horas 45 minutos.**
- No está permitido consultar libros, apuntes, resúmenes, etc.
- Utiliza hojas diferentes para la solución de cada uno de los dos problemas y para las cuestiones.
- Puedes pedir hojas adicionales si completas las que se te han entregado.
- Empieza escribiendo tu nombre, apellidos y DNI con letra clara en la parte superior de cada hoja.
- Puedes utilizar lápiz y borrar lo que desees. Puedes entregar la solución final escrita con lápiz si lo deseas. Tacha o borra claramente aquello que entregues y no desees que se tenga en cuenta en la evaluación.
- No es necesario entregar el enunciado del examen.
- La organización del examen y la distribución de puntos en los diferentes ejercicios propuestos se resume en la siguiente tabla:

Apartado	Puntos
Cuestión 1	0,75
Cuestión 2	0,75
Problema 1(a)	2,0
Problema 1(b)	3,0
Problema 2(a)	2,0
Problema 2(b)	1,5
Total	10,0

Cuestiones (15 %)

1. **(0,75 puntos)** ¿Qué se obtiene en la salida estándar al llamar a estas dos funciones?

```
void funcion1(void){
  int uno=1, dos=2;
  int & ref = uno;

  ref = dos;
  ref = 3;

  cout << "uno=" << uno << ", ";
  cout << "dos=" << dos << ", ";
  cout << "ref=" << ref << ". " << endl;
}

void funcion2(void){

  int uno=1, dos=2;
  int *ptr = &uno;

  ptr = &dos;
  *ptr = 3;

  cout << "uno=" << uno << ", ";
  cout << "dos=" << dos << ", ";
  cout << "*ptr=" << *ptr <<  "." << endl;
}
```

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

```
class Vector {
   int tam;
   float *vec;
  public:
...
};
```

Cada objeto de esta clase representa un vector para el que se reserva memoria dinámicamente en tiempo de ejecución. El puntero `vec` almacena la dirección de dicha memoria y el campo `tam` almacena el número de componentes del vector.

Un programador ha implementado para esta clase un constructor sin argumentos y un operador de asignación como sigue:

```
Vector::Vector () : tam(0), vec(NULL) {}

void Vector::operator= (const Vector & v) {
  vec = new float[v.tam];
  for (int i = 0; i <= v.tam; i++)
    vec[i] = v.vec[i];
}
```

¿Es correcta su implementación? Si crees que no lo es, indica qué error o errores observas, explica brevemente por qué y realiza una implementación correcta.

Problemas (85 %)

1. *a*) **(2 puntos)** Implementa en C++ la clase `Punto2D` y la clase `Region`. Cada objeto de la clase `Punto2D` tiene dos coordenadas x e y de tipo entero. Cada objeto de la clase `Region` está formado por una lista de puntos de la clase `Punto2D`. *Para implementar la lista de puntos no puedes usar la biblioteca estándar de plantillas (*STL*).*

   En la clase `Region` debes definir las funciones `PertenecePunto(p)` (que devuelve cierto si un punto `p` está en la lista de puntos de la región) e `InsertarPunto(p)` (que inserta un punto `p` en la lista de puntos de la región). Esta última debe comprobar que el punto a insertar no está ya en la lista, evitando que la lista contenga puntos repetidos.

   En la clase `Punto2D` debes definir sólo las funciones (incluidos constructores, destructores y operadores) de las que hagas uso en tu implementación de las funciones anteriores de la clase `Region`.

   Implementa también el operador de entrada `>>` para leer objetos de la clase `Punto2D`. Haciendo uso de él, implementa el operador de entrada `>>` para leer objetos de la clase `Region` con el formato:

   $$\{(x_1, y_1), (x_2, y_2), \ldots, (x_n, y_n)\}$$

   Puedes suponer que el formato en la entrada de datos es siempre correcto y que al menos se lee siempre un punto.

   b) **(3 puntos)** Añade a la clase `Region` los operadores necesarios para que el siguiente fragmento de código pueda compilarse y ejecutarse correctamente teniendo en cuenta que:

   - la región suma de dos regiones contiene todos los puntos de las dos.
   - la región suma de una región y un punto `p` contiene todos los puntos de la región y el punto `p`.

   En ambos casos, la región resultante no debe contener puntos repetidos

```
#include "Punto.h"
#include "Region.h"

int main() {
  Region reg1, reg2, reg3;
  Punto2D p(4,5);
  //...
  reg3 = reg1 + p;
  reg3 = p + reg1;
  reg3 = reg1 + reg2;
  reg3 = reg1 += reg2;
}
```

   ¿Sería necesario definir el constructor copia, el operador de asignación y el destructor para la clase `Region`? Razona brevemente tu respuesta. En caso afirmativo, realiza su implementación.

2. *a*) **(2 puntos)** Consideremos la construcción de un programa que se ocupa de la gestión del personal de una empresa en la que hay dos tipos de empleados: comerciales y administrativos. Todos ellos cobran un sueldo base, que puede ser diferente para cada empleado. Además, el sueldo de un administrativo tiene un plus por horas extra trabajadas, a razón de 5 euros la hora. Los comerciales cobran un plus por ventas, calculado como el 10 % de sus ventas netas de cada mes.

Disponemos de la función `sumaSueldos()` para calcular el total que la empresa gasta con los sueldos de sus empleados. A la función se le pasa un *array* de punteros a empleados y el número de empleados, como se muestra en el siguiente fragmento de código:

```
float sumaSueldos (Empleado * personal[], int numEmpleados) {
  float suma = 0.0;
  for (int i = 0; i < numEmpleados; i++)
    suma += personal[i]->sueldo();
  return suma;
}

int main() {
  const int N=3;

  Empleado * personal[N];

  personal[0]=new Comercial("Garcia", 1500, 5000);  // base 1500 EUR y ventas 5000 EUR
  personal[1]=new Administrativo("Perez", 1000, 5); // base 1000 EUR y 5 horas extra
  personal[2]=new Administrativo("Chiva", 1100, 2); // base 1100 EUR y 2 horas extra

  cout << "Suma de sueldos = " << sumaSueldos(personal,3) << " euros" << endl;
}
```

La ejecución de la función `main()` presenta este resultado en la pantalla:

```
Suma de sueldos = 4135 euros
```

4135 euros es el resultado de $(1500 + 5000 * 0{,}10) + (1000 + 5 * 5) + (1100 + 2 * 5)$

Define las clases necesarias y sólo aquellas funciones miembro que se requieran para hacer funcionar el programa, de tal forma que la función `sumaSueldos()` haga uso de la **ligadura dinámica** (o *polimorfismo en tiempo de ejecución*).

b) **(1,5 puntos)** Escribe de nuevo el código de las funciones `sumaSueldos()` y `main()` del apartado anterior usando, en lugar del *array* `personal`, un objeto de la clase `list` de la biblioteca estándar de plantillas (STL).