Examen de Programación Avanzada (IG17/II17)
Ingenierías Informáticas - Universitat Jaume I

25 de Enero de 2005

Normativa

Lee atentamente las siguientes instrucciones antes de empezar a resolver el examen:

— **Duración del examen: 3,5 horas.**

— No está permitido consultar libros, apuntes, resúmenes, etc.

— Utiliza hojas diferentes para la solución de las cuestiones y de cada uno de los dos 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.

— 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.

— La organización del examen y la distribución de puntos en los diferentes ejercicios propuestos se resume en la siguiente tabla:

Ejercicio	Puntos
Cuestión 1	1,0
Cuestión 2	1,0
Cuestión 3	0,5
Problema 1a	3,5
Problema 1b	1,0
Problema 2	3,0
Total	10,0

Cuestiones

1. **(1,0 puntos)** Indica cuál será la salida por la pantalla tras la ejecución de este programa. **Justifica tu respuesta**.

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

class Madre
{
  protected:
    int dato;
  public:
    Madre(int d = 0) : dato(d) {}
    void Ver() const { cout << "Madre: dato = " << dato << endl; }
    virtual void Inicializar() { dato = 55; }
    virtual void Modificar() { dato++; }
};

class Hija : public Madre
{
  public:
    Hija(int d = 0) : Madre(d) {}
    void Ver() const { cout << "Hija: dato = " << dato << endl; }
    void Inicializar() { dato = 99; }
    void Modificar() { dato--; }
};

int main()
{
  Madre a(10);
  Hija b(20);
  Madre *c = new Hija;

  a.Inicializar();
  a.Modificar();
  a.Ver();

  b.Inicializar();
  b.Modificar();
  b.Ver();

  c->Inicializar();
  c->Modificar();
  c->Ver();

  delete c;

  return 0;
}
```

2. **(1,0 puntos)** Supongamos que la siguiente declaración:

```
class Cola
{
   struct ElementoCola
   {
      int info;
      ElementoCola *sig;
      ElementoCola (int i, ElementoCola *s) : info(i), sig(s) {}
      ...
   } *cap,*cua;
public:
   ...
};
```

permite trabajar con colas en las que cada elemento tiene un campo de información de tipo entero y un puntero al siguiente elemento de la cola, los elementos se guardan en el orden en que son insertados, y la cola se maneja mediante un puntero `cap` al primer elemento que fue insertado y un puntero `cua` al último elemento insertado.

Encuentra los errores que hay en los siguientes fragmentos de código, e indica de qué tipo(s) son (conceptuales, de compilación, o de ejecución) y **cómo corregirlos**:

a)
```
Cola::Ver()
{
  ElementoCola *aux = new ElementoCola;
  aux = cap;
  while (aux!=NULL)
  {
    aux->info.Ver();
    aux = aux->sig;
  }
}
```

b)
```
Cola::Cola(const &Cola c)
{
  Insertar(c);
}

void Cola::Insertar(const &Cola c)
{
  for (ElementoCola *aux = cap; aux->sig != NULL; aux = aux->sig)
    Insertar(aux->info);
}

void Cola::Insertar(int info)
{
  ElementoCola *nuevo = new ElementoCola(info,NULL);
  if (cap==NULL)
    cap = nuevo;
  else
    cua->sig = nuevo;

  cua = nuevo;
}
```

3. **(0,5 puntos)** Encuentra los errores que hay en el siguiente fragmento de código, y explica de qué tipo(s) son (conceptuales, de compilación, o de ejecución) y en qué consisten.

```
class Fichero
{
  protected:
   string nombre;
   int nBytes;
  public:
    void Fichero(string, int = 0);
    virtual void Copiar(const Fichero &f);
    virtual void Borrar() = 0;
};

int main()
{
  Fichero f1("examen.dat", 100), f2("examen.dat.backup");
  f2.Copiar(f1);
  return 0;
}
```

Problemas

1. Resuelve los apartados siguientes.

   a) **(3,5 puntos)** Un multiconjunto es un conjunto que puede contener elementos repetidos. Estamos desarrollando una aplicación en la que necesitamos trabajar con multiconjuntos que contengan elementos de tipo entero, como por ejemplo {`12,35,7,23,12,12,35,7,9`}.

   Las operaciones que necesitamos realizar con dichos multiconjuntos son:

   1) Insertar elementos en el multiconjunto, como en:

      ```
      Multiconjunto datos1, datos2, datos3;
      ...
      datos1.Insertar(7).Insertar(25);
      ```

   2) Averiguar cuántas veces aparece un elemento en el multiconjunto, como en:

      ```
      if (datos1.Contar(7) > 5) ...
      ```

      Si el elemento `elem` no está en el multiconjunto, `Contar(elem)` devolverá `0`.

   3) Eliminar todas las ocurrencias de un elemento del multiconjunto, como en:

      ```
      datos1 -= 7;
      datos2 = datos1 - 7;
      ```

   4) Dados dos multiconjuntos, eliminar del primero todas las ocurrencias de los elementos que contiene el segundo, como en:

      ```
      datos1 -= datos2;
      datos3 = datos1 - datos2;
      ```

   5) Comparar dos multiconjuntos para averiguar si son iguales, como en:

      ```
      if (datos1 == datos2) ...
      ```

      Dos multiconjuntos se consideran iguales si, y sólo si, contienen los mismos elementos y en la misma cantidad, aunque aparezcan en diferente orden. Por ejemplo, los multiconjuntos {`7,9,5,7,7,5`} y {`9,5,7,7,7,5`} se consideran iguales.

   Implementa en C++ la clase `Multiconjunto` con dichas operaciones, teniendo en cuenta que no se conoce *a priori* el tamaño del multiconjunto, y **la memoria necesaria para guardar los elementos se debe gestionar dinámicamente**, reservándola y liberándola en tiempo de ejecución cuando corresponda. Implementa también el destructor, el operador de asignación, el constructor sin argumentos y el constructor de copia en caso de que los que proporciona el compilador por defecto no sean adecuados.

   En las operaciones de los apartados 3) y 4) ten en cuenta que cuando se intente eliminar del multiconjunto un elemento que no tiene, el resultado debe ser el mismo multiconjunto. Considera también en todas las funciones la posibilidad de que cualquiera de los argumentos sea un multiconjunto vacío.

   Sugerencias:

   — Puedes utilizar una lista en la que cada nodo contenga un elemento y un contador de ocurrencias de ese elemento.

   — No es necesario que busques la solución más eficiente; la eficiencia de la solución no se tendrá en cuenta en este ejercicio.

   — Al definir las funciones de la clase, puedes abreviar `Multiconjunto` por `MC`, para que te cueste menos escribir.

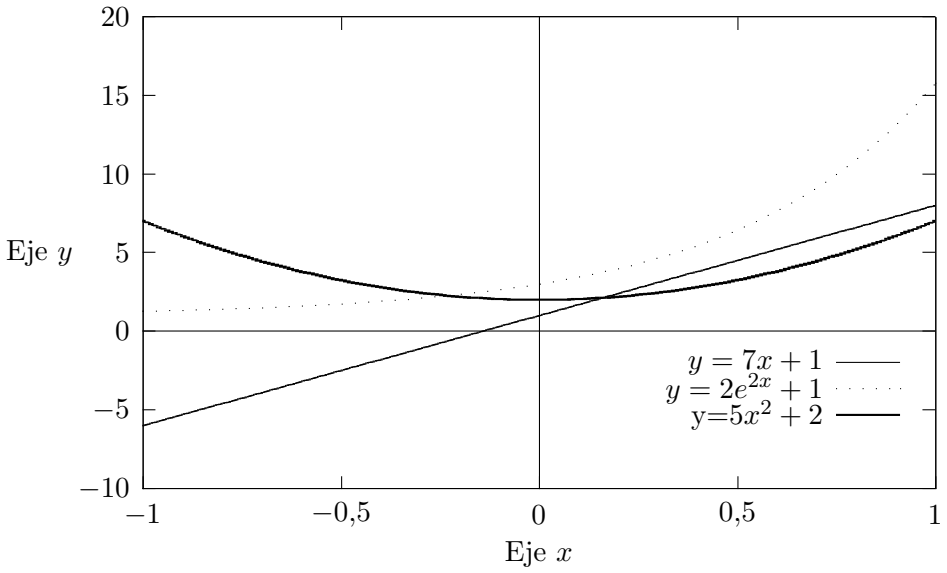
b) **(1,0 puntos)**

1) Reescribe la declaración de la clase `Multiconjunto` para que sea una clase genérica, en la que los elementos del conjunto puedan ser de cualquier tipo T, no sólo de tipo entero (suponiendo que todos los elementos del conjunto son del mismo tipo). Basta con que digas cómo sería la declaración de la clase, incluyendo las declaraciones de las funciones miembro; no hace falta que implementes las funciones.
2) Indica qué operaciones debería soportar el tipo T para que las funciones de la implementación de la clase multiconjunto genérica se pudieran instanciar y funcionar correctamente.
3) Indica cómo se crearían instancias de esa clase genérica, es decir, cómo se declararía, por ejemplo, un objeto de la clase multiconjunto cuyos elementos fuesen de tipo `Punto`.

2. **(3,0 puntos)** En una aplicación necesitamos trabajar con varios tipos de funciones matemáticas:

   — Rectas de la forma $y = ax + b$.
   — Exponenciales de la forma $y = ae^{bx} + c$.
   — Parábolas de la forma $y = ax^2 + b$

   Supongamos, por ejemplo, que necesitamos almacenar las funciones de la siguiente figura en una lista que nos permita realizar algunas operaciones sobre el conjunto de funciones almacenadas.



   Utilizamos para ello el código de la función `main()` siguiente, en el que, además de crear un objeto para cada una de las funciones matemáticas y almacenarlos en una lista, se muestra la representación en forma de texto de cada una de las funciones matemáticas (observa la salida del programa) y se calcula cuál es el valor $y = f(x)$ menor de todas ellas cuando $x = 0{,}0$ y cuando $x = 0{,}5$:

```
int main()
{
  Recta recta(7,1);                 // y = 7*x+1
  Exponencial exponencial(2,2,1);   // y = 2*e^(2*x)+1
  Parabola parabola(5,2);           // y = 5*x^2+2

  ListaFunciones funciones;
  funciones.InsertarRecta(recta);
  funciones.InsertarExponencial(exponencial);
  funciones.InsertarParabola(parabola);

  cout << "Lista de funciones:" << endl;
  funciones.Mostrar();

  cout << "La menor y=f(x) cuando x=0.0 es " << funciones.MenorY(0.0) << endl;
  cout << "La menor y=f(x) cuando x=0.5 es " << funciones.MenorY(0.5) << endl;

  return 0;
}
```

La salida del programa es la siguiente:

```
Lista de funciones:
y = 5*x^2 + 2
y = 2*e^(2*x) + 1
y = 7*x + 1
La menor y=f(x) cuando x=0.0 es 1
La menor y=f(x) cuando x=0.5 es 3.25
```

Observa que necesitamos que las clases `Recta`, `Exponencial` y `Parabola` almacenen internamente los datos necesarios para representar cada función (los valores a, b, ...). Cada una de estas clases debe ser capaz de responder a las funciones `Mostrar()`, que muestra en la salida estándar la representación en texto de la función, y `Evaluar(x)`, que calcula $y = f(x)$, es decir, evalúa la función para un valor de x concreto[1].

También necesitamos que la clase `ListaFunciones` permita almacenar una lista de funciones de cualquiera de los tipos mencionados. Como puedes comprobar en el ejemplo, esta clase debe permitir la inserción de funciones matemáticas en la lista (sin importar el orden de inserción). Además, la función `Mostrar()` debe mostrar cada una de las funciones de la lista y la función `MenorY(x)` debe devolver el menor de los valores $y = f(x)$, para todas las funciones de la lista.

Implementa las clases mencionadas arriba y cualquier otra que consideres necesaria para que al ejecutar la función `main()` del ejemplo se obtenga la salida de datos presentada. Para que tu solución sea considerada valida, debes establecer una relación de herencia razonable entre las clases y utilizar *ligadura dinámica* (polimorfismo en tiempo de ejecución).

Nota: Para calcular e^x puedes usar la función `double exp(double)` de la biblioteca matemática.

[1] Evaluar la función consiste en sustituir x por un valor concreto y realizar las operaciones matemáticas oportunas para calcular $y = f(x)$.