WUOLAH

Indalecia
www.wuolah.com/student/Indalecia

78044

la_solucion_comentada.pdf

JUNIO 2018 SOLUCIONADO

1° Metodología de la Programación

Grado en Ingeniería Informática

Escuela Técnica Superior de Ingenierías Informática y de Telecomunicación
Universidad de Granada

WUOLAH + #QuédateEnCasa

#KeepCalm #EstudiaUnPoquito

Enhorabuena, por ponerte a estudiar te **regalamos un cartel** incluído entre estos apuntes para estos días.

Reservados todos los derechos. No se permite la explotación económica ni la transformación de esta obra. Queda permitida la impresión en su totalidad.

EJERCICIO 1. Métodos básicos de la clase

El **constructor sin argumentos** debe crear una red vacía: una red sin líneas.

```
/*****************************************************************************/
// Constructor sin argumentos--> crea una red vacía

RedMetro :: RedMetro (void) : num_lineas(0), lineas(0) { }

/*****************************************************************************/
```

A partir del estado en el que queda el objeto creado por el constructor anterior, el método que nos indica si una **RedMetro** está vacía puede, simplemente, consultar el valor del campo **num_lineas**.
En cualquier caso, aseguramos, y nos preocupamos de implementarlo así, que cuando **num_lineas** tiene el valor 0, el puntero **lineas** también estará a 0. Es particularmente importante asegurarse del valor del puntero **lineas** cuando se intente liberar la memoria dinámica accesible a partir de él.

```
/*****************************************************************************/
bool RedMetro :: EstaVacia (void)
{
        return (num_lineas==0);

        // También  return (lineas==0);
        // También  return (num_lineas==0 && lineas==0);
}
/*****************************************************************************/
```

Por otro lado, los que hemos denominados ***métodos básicos*** se escriben en función de los **métodos privados** que se encargan de reservar, liberar memoria y copiar datos. Estos métodos han sido empleados en los diferentes problemas sobre los que hemos trabajado durante el curso.

```
/*****************************************************************************/
// PRE: el_num_lineas >0

void RedMetro :: ReservarMemoria (int el_num_lineas)
{
        lineas = new Linea[el_num_lineas]; // Nota 1
}

/*****************************************************************************/

void RedMetro :: LiberarMemoria ()
{
        if (lineas) // Muy importante
                delete [] lineas; // Nota 2

        lineas = 0;
        num_lineas = 0;
}

/*****************************************************************************/
```

Reservados todos los derechos. No se permite la explotación económica ni la transformación de esta obra. Queda permitida la impresión en su totalidad.

WUOLAH

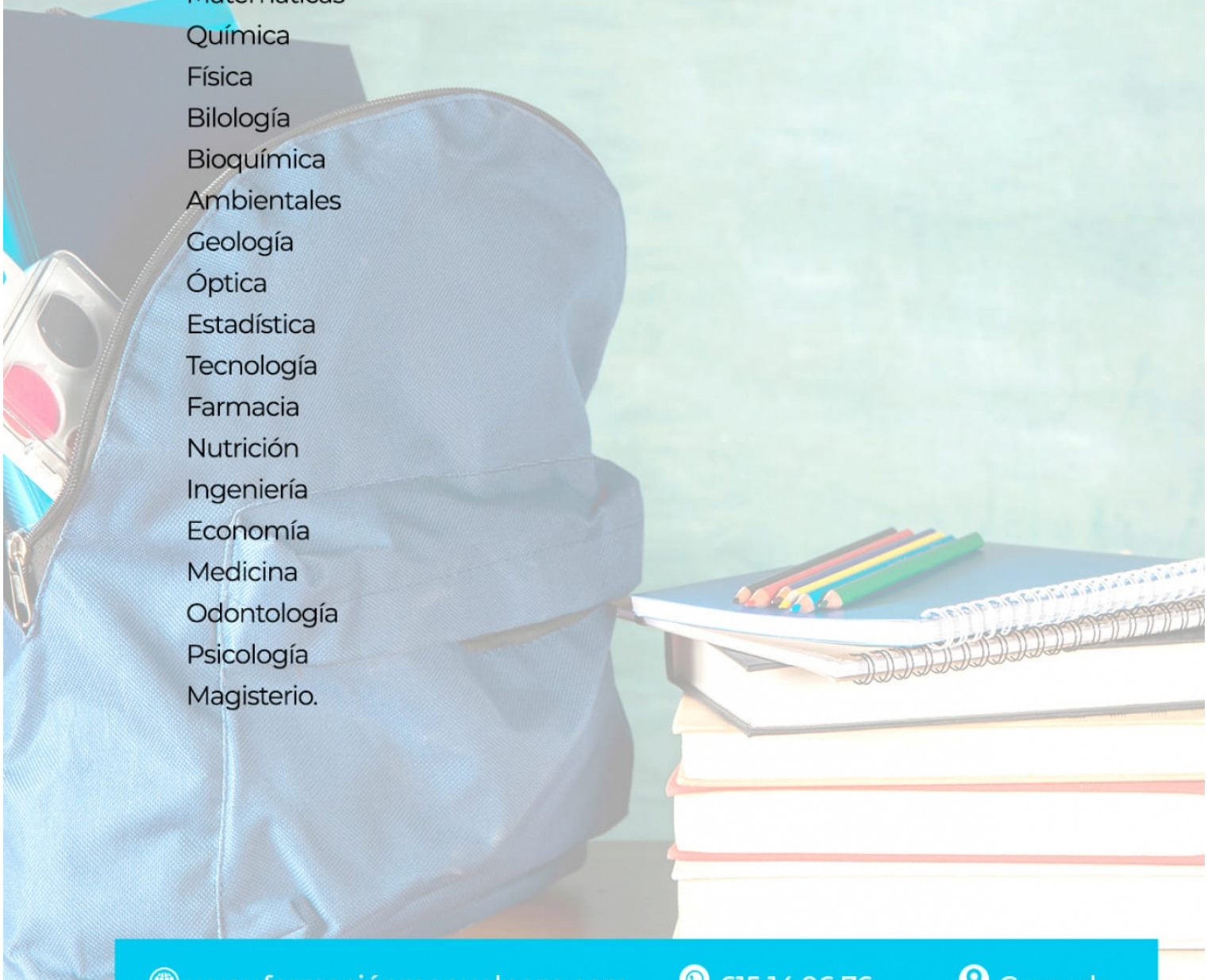
Formación Manuel Pozo

PRUEBA NUESTRA FORMACIÓN ONLINE CON CLASES EN DIRECTO.

PRIMERA CLASE DE PRUEBA EN GRUPO, INTERACTÚA CON NUESTROS PROFESORES DESDE TU PC, CONSÚLTA TUS DUDAS Y PREGUNTAS DIRECTAMENTE DE FORMA ONLINE.

PROFESORES ESPECIALIZADOS EN MÁS DE 150 ASIGNATURAS

- Matemáticas
- Química
- Física
- Bilología
- Bioquímica
- Ambientales
- Geología
- Óptica
- Estadística
- Tecnología
- Farmacia
- Nutrición
- Ingeniería
- Economía
- Medicina
- Odontología
- Psicología
- Magisterio.



www.formaciónmanuelpozo.com 615 14 96 76 Granada

```
/*****************************************************************************/

void RedMetro :: CopiarDatos (const RedMetro & otra)
{
        num_lineas = otra.num_lineas;

        for (int l=0; l<num_lineas; l++)
                lineas[l] = otra.lineas[l];  // Nota 3
}
 /*****************************************************************************/
```

Nota 1: **new** provoca que actúe el constructor sin argumentos de la clase **Linea** para cada objeto del array. Ese constructor se supone que está implementado.

Nota 2: **delete** provoca la liberación de la memoria que ocupa el array de objetos **Linea** , por lo que actúa el destructor de la clase **Linea** para cada objeto del array. Ese destructor se supone que está implementado.

Nota 3: Actúa el operador de asignación entre objetos de la clase **Linea**, que se supone implementado. El operador de asignación es NECESARIO sobrecargarlo para esa clase ya que contiene datos alojados en la memoria dinámica y la asignación requiere implementar la **copia profunda**. A raíz de este comentario, debo indicar que en la corrección he encontrado algunas respuestas que han empleado **memcpy()** en lugar del ciclo for.

NO ES VÁLIDA LA COPIA CON memcpy():

```
        memcpy (lineas, otra.lineas, num_lineas*sizeof(Linea));
```

ya que se realiza una **copia superficial** y se copiarán las direcciones de memoria de los punteros **paradas** (copia superficial) de manera que dos redes diferentes compartirían la misma memoria para sus líneas, lo que resulta inaceptable.

Una vez implementados los métodos privados de utilidad, los métodos públicos son triviales y bien conocidos.

```
/*****************************************************************************/
// Constructor de copia

RedMetro :: RedMetro (const RedMetro & otra)
{
        ReservarMemoria (otra.num_lineas);
        CopiarDatos (otra);
}

/*****************************************************************************/
// Destructor

RedMetro :: ~RedMetro(void)
{
        LiberarMemoria ();
}
/*****************************************************************************/
```


WUOLAH
Reservados todos los derechos. No se permite la explotación económica ni la transformación de esta obra. Queda permitida la impresión en su totalidad.
Recordatorio de Wuolah: HOY SI QUE ERES UN CRACK #KeepCalm y #estudiaUnPoquito

```
/*****************************************************************************/
// Operador de asignación

RedMetro & RedMetro :: operator = (const RedMetro & otra)
{
        if (this != &otra) {

                LiberarMemoria ();
                ReservarMemoria (otra.num_lineas);
                CopiarDatos (otra);
        }
        return (*this);
}
/*****************************************************************************/
```

He encontrado varias respuestas que incorporan constructores adicionales que reciben el número de objetos:

```
RedMetro :: RedMetro (const int el_num_lineas);
Linea :: Linea (const int el_num_paradas);
```

que crean un array de objetos (**Linea** ó **InfoParada**) vacíos que después "sustituyen" (con el operador de asignación) por objetos con valor. Lo usan para la sobrecarga del operador += y para la sobrecarga del operador >>

Veremos que es **innecesario** ya que:

1) en el caso del operador += bastará con redimensionar el array de **Lineas/InfoParadas**. El propio método se encargará de todo accediendo directamente a la representación, sin necesidad de usar otros métodos de la clase (PREGUNTA 2.1)

2) en el caso del operador >> para la clase **RedMetro** veremos que un objeto **RedMetro** se construirá creando uno vacío inicialmente y añadiendo línea a línea con el operador +=. Una línea se creará de la misma manera: a partir de un objeto **Linea** vacío se van añadiendo objetos **InfoParada** con el operador += (PREGUNTA 2.2.b)

WUOLAH
Reservados todos los derechos. No se permite la explotación económica ni la transformación de esta obra. Queda permitida la impresión en su totalidad.
Recordatorio de Wuolah: HOY SI QUE ERES UN CRACK #KeepCalm y #estudiaUnPoquito

Gana dinerito extra.
Recomienda a tus negocios favoritos que se anuncien en Wuolah y llévate 50€.
Te daremos un código promocional para que puedan anunciarse desde 99€.
1 Ve a tu negocio favorito • 2 Dales tu código de promo • 3 Diles que nos llamen o nos escriban.

EJERCICIO 2. Sobrecarga de operadores

2.1 Operador += para la clase **Línea**

El prototipo del métodfo que sobrecarga este operador es:

```
Linea & operator += (const InfoParada & nueva_parada)
```

El objetivo es añadir una nueva **InfoParada** al objeto implícito (clase **Linea**). El array referenciado por el puntero paradas **NO** tiene espacio disponible, por lo que debe redimensionarse para poder añadir el nuevo objeto **nueva_parada**. Así, se debe 1) reservar "nueva" memoria (con espacio para una casilla más) , 2) copiar los datos **InfoParada** que ya existen a la nueva zona de memoria, 3) copiar la nueva **InfoParada** en la última casilla de la "nueva" memoria, 4) liberar la "antigua" memoria y 5) actualizar los campos privados del objeto implícito.

No olvide devolver una referencia al objeto implícito (6).

```
/*************************************************************************/
Linea & Linea :: operator += (const InfoParada & nueva_parada)
{
        InfoParada * tmp = new InfoParada[num_paradas+1];  // 1 (ver explicación)

        for (int p=0; p<num_paradas; p++)  // 2

                tmp[p] = paradas[p]; // Asignación entre InfoParada (sin memoria dinámica)

        tmp[num_paradas] = nueva_parada;  // 3. Asignación entre InfoParada (ídem)

        delete [] paradas;  // 4

        paradas = tmp;      // 5
        num_paradas++;

        return (*this); // 6
}
/*************************************************************************/
```

Nota: En este caso **SÍ** Podría haberse sustituido el ciclo **for** (2) y la asignación la nueva parada (3) por **memcpy()** ya que en este caso no hay direcciones de memoria entre el contenido copiado (son objetos de tipo **InfoParada**) y la copia superficial es una solución correcta:

```
memcpy (tmp, paradas, num_paradas*sizeof(InfoParada)); // 2
memcpy (&tmp[num_paradas], &nueva_parada, sizeof(InfoParada)); // 3
```

653
811
910

WUOLAH

Reservados todos los derechos. No se permite la explotación económica ni la transformación de esta obra. Queda permitida la impresión en su totalidad.

Aunque se supone que el operador += está sobrecargado para **RedMetro** mostraré su implementación. Observe el paralelismo con el operador += de la clase **Linea**

```
/*************************************************************************/
RedMetro & RedMetro :: operator += (const Linea & nueva_linea)
{
        Linea * tmp = new Linea[num_lineas+1];

        for (int l=0; l<num_lineas; l++)
            tmp[l] = lineas[l];  // 1. Operator = de la clase "Linea"

        tmp[num_lineas] = nueva_linea;    // 2. Operator = de la clase "Linea"

        delete [] lineas;

        lineas = tmp;
        num_lineas++;

        return (*this);
}
/*************************************************************************/
```

Nota: En este operador las asignaciones entre datos de tipo **Linea** (1 y 2) se realizan con el operador sobrecargado = para la clase **Linea** ya que hay memoria dinámica involucrada. Se supone implementado. No es posible la copia superficial con **memcpy()**

Reservados todos los derechos. No se permite la explotación económica ni la transformación de esta obra. Queda permitida la impresión en su totalidad.

WUOLAH
Recordatorio de Wuolah: HOY SI QUE ERES UN CRACK #KeepCalm y #estudiaUnPoquito

2.2.a Operador << para la clase **RedMetro**

Se supone que está implementado el operador para las otras clases:

```
friend ostream & operator << (ostream & out, const Linea & la_linea);
friend ostream & operator << (ostream & out, const InfoParada & la_parada);
```

Tomando como referencia la figura C es fácil darse cuenta que el operador << para una **InfoParada** debería dar como resultado, p.e.: **1 S** mientras que para una **Linea**, p.e.:

```
4
1 S
2 N
3 S
4 N
```

El operador << para la clase **RedMetro** tendrá como prototipo:

```
friend ostream & operator << (ostream & out, const RedMetro & red);
```

y se implementa empleando los operadores disponibles:

```
/*****************************************************************************/
ostream & operator << (ostream & out, const RedMetro & red)
{
        out.setf (ios::fixed);
        out.setf (ios::showpoint);

        out << setw(3) << red.num_lineas << endl;

        for (int l=1; l<=red.num_lineas; l++) {

            Linea la_linea = red[l]; // Operator [] de la clase RedMetro
            out << la_linea;  // Operator << de la clase Linea
            // Resumido: out << red[l];
        }
        return (out);
}
/*****************************************************************************/
```

Aunque no se pide, muestro la implementación para las otras clases:

```
/*****************************************************************************/
ostream & operator << (ostream & out, const Linea & la_linea)
{
  out << setw(3) << la_linea.num_paradas << endl;

  for (int p=1; p<=la_linea.num_paradas; p++) {

    InfoParada ip = la_linea[p]; // Operator [] de la clase Linea
    out << ip << endl;  // Operador << de la clase Infoparada
    // Resumido: out << la_linea[p];
  }
  return (out);
}
/*****************************************************************************/
```


WUOLAH
Reservados todos los derechos. No se permite la explotación económica ni la transformación de esta obra. Queda permitida la impresión en su totalidad.
Recordatorio de Wuolah: HOY SI QUE ERES UN CRACK #KeepCalm y #estudiaUnPoquito

```
/*****************************************************************************/
ostream & operator << (ostream & out, const InfoParada & la_parada)
{
    out << setw(3) << la_parada.indice_parada << "  ";
    out << ((la_parada.activa) ? "S" : "N");

    return (out);

}
/*****************************************************************************/
```

He encontrado soluciones que no emplean el operador << para las clases **Linea** e **InfoParada**. En estos casos, además de **repetir** código, si se resuelve sin errores el código es realmente farragoso y difícilmente modificable. Lo habitual, no obstante, ha sido encontrar soluciones incorrectas en las que se accede a los **datos privados** de las clases **Linea** e **InfoParada**.

Una solución sin errores que no emplea el operador << para las clases **Linea** e **InfoParada** sería:

```
/*****************************************************************************/
ostream & operator << (ostream & out, const RedMetro & red)
{
        out << setw(3) << red.num_lineas << endl;

        for (int l=1; l<=red.num_lineas; l++) {

          out << setw(3) << red[l].GetNumParadas() << endl;

          for (int p=1; p<=red[l].GetNumParadas(); p++) {

             out << setw(3) << red[l][p].GetIndice() << "  ";

             out << ((red[l][p].EstaActiva()) ? "S" : "N");

             out << endl;

          } // for p

      } // for l

      return (out);
}
/*****************************************************************************/
```

Ahora, en la misma línea de trabajo, si se escribiera el operador << para la clase **Linea**, sin emplear el operador << para la clase **InfoParada**:

Reservados todos los derechos. No se permite la explotación económica ni la transformación de esta obra. Queda permitida la impresión en su totalidad.

Recordatorio de Wuolah: HOY SI QUE ERES UN CRACK #KeepCalm y #estudiaUnPoquito

Gana dinerito extra.
Recomienda a tus negocios favoritos que se anuncien en Wuolah y llévate 50€.
Te daremos un código promocional para que puedan anunciarse desde 99€.
1 Ve a tu negocio favorito • 2 Dales tu código de promo • 3 Diles que nos llamen o nos escriban.


```
/*****************************************************************************/
ostream & operator << (ostream & out, const Linea & linea)
{
     out << setw(3) << linea.num_paradas << endl;

     for (int p=0; p< linea.num_paradas; p++) {

          out << setw(3) << linea[p].GetIndice() << "  ";
          out << ((linea[p].EstaActiva()) ? "S" : "N");
          out << endl;

     } // for p

   return (out);
}
/*****************************************************************************/
```

y finalmente, el operador << para la clase **InfoParada** sería el mostrado anteriormente.

Es evidente que no es una buena solución, ya que además de repetir código éste resulta difícilmente modificable: ¿cuántos cambios hay que realizar en estos métodos si quisiéramos separar el número de parada de cada línea de la S ó la N con un guión, por ejemplo? ¿y si ese mismo cambio se hiciera con la implementación sugerida -usando los operadores de las otras clases-?

653
811
910
WUOLAH

Reservados todos los derechos. No se permite la explotación económica ni la transformación de esta obra. Queda permitida la impresión en su totalidad.

2.2.b Operador >> para la clase **RedMetro**

En este caso **NO** se supone implementado el operador >> para las otras clases. Decir que no se supone su implementación no impide que se usen y se implementen, de hecho es la solución recomendada. Resolver esta pregunta sin usar las sobrecargas para las otras clases conduce inevitablemente a **repetir código** y lo que es peor, si se resuelve sin errores, a código farragoso. No obstante, lo habitual ha sido encontrar soluciones en las que se accede a los **datos privados** de las clases **Linea** e **InfoParada**.

La solución deseada requiera la implementación del operador >> para las clases **Linea** e **InfoParada**. Los prototipos e implementaciones son:

```
        friend istream & operator >> (istream & in, RedMetro & red);
        friend istream & operator >> (istream & in, Linea & la_linea);
        friend istream & operator >> (istream & in, InfoParada & la_parada);

/*****************************************************************************/

istream & operator >>  (istream & in, RedMetro & red)
{
      red.LiberarMemoria (); // Limpiar el objeto: Liberar memoria reservada previamente

      int lineas_a_leer;
      in >> lineas_a_leer;

      for (int l=1; l<=lineas_a_leer; l++) { // Solo interesa el núm. de vueltas: no se usa "l"

        Linea linea_nueva;
        in >> linea_nueva;  // Operador >> de Linea

        red += linea_nueva;
      }
      return (in);
}
/*****************************************************************************/

istream & operator >> (istream & in, Linea & la_linea)
{
      la_linea.LiberarMemoria (); // Limpiar el objeto: Liberar memoria reservada previamente

      int el_num_paradas;
      in >> el_num_paradas;

      for (int p=1; p<=el_num_paradas; p++) { // Solo interesa el núm. de vueltas: no se usa "p"
        InfoParada parada_nueva;
        in >> parada_nueva;     // Operador >> de InfoParada

        la_linea += parada_nueva;
      }

      return (in);
}

/*****************************************************************************/
```


WUOLAH
Reservados todos los derechos. No se permite la explotación económica ni la transformación de esta obra. Queda permitida la impresión en su totalidad.
Recordatorio de Wuolah: HOY SI QUE ERES UN CRACK #KeepCalm y #estudiaUnPoquito

```
/*****************************************************************************/

istream & operator >> (istream & in, InfoParada & la_parada)
{
        int el_num_parada;
        char esta_activa_sn;

        in >> el_num_parada >> esta_activa_sn;

        bool esta_activa = ((esta_activa_sn == 'S') ? true : false);

        InfoParada parada_leida (esta_activa, el_num_parada);
        la_parada = parada_leida;

        return (in);
}
/*****************************************************************************/
```

Como notas finales:

1) He encontrado bastantes soluciones que no se han preocupado de liberar la memoria del objeto que se está "leyendo" lo que no es correcto.

2) También he encontrado soluciones que usan un constructor específico para crear los arrays de datos **Linea** e **InfoParada** y después emplean en operador de asignación para cambiar los objetos. Esta solución es más propensa a errores y menos intuitiva que la que propongo.

WUOLAH
Reservados todos los derechos. No se permite la explotación económica ni la transformación de esta obra. Queda permitida la impresión en su totalidad.
Recordatorio de Wuolah: HOY SI QUE ERES UN CRACK #KeepCalm y #estudiaUnPoquito

EJERCICIO 3. Métodos y funciones para E/S

Siguiendo la línea de los ejercicios con los que hemos trabajado, proponemos implementar un método privado de la clase **RedMetro**:

```
void RecuperaDatosDeFichero (const char * nombre);
```

que podría emplearse también, p.e., en un método como

```
void LeeRedMetro (const char *nombre);
```

En nuestro caso, el constructor pedido es trivial (no requiere liberar memoria) mientras que el método **LeerRedMetro** sí requiere liberarla antes de leer del fichero.

```
/*****************************************************************************/
// PRE: El fichero "nombre" existe y puede abrirse para lectura

RedMetro :: RedMetro (const char *nombre) : num_lineas(0), lineas(0)
{
        // Leer los datos del fichero
        RecuperaDatosDeFichero (nombre);
}
/*****************************************************************************/
```

El método privado se encargará de abrir el fichero (se supone que existe y se puede abrir), comprobar si aparece la palabra mágica y (¡¡¡fundamental!!!) ---> **leer los datos de la red usando el operador >>.** Observe que si no se encuentra la palabra mágica el objeto estará **vacío.**

```
/*****************************************************************************/
// PRE: El fichero "nombre" existe y puede abrirse para lectura

void RedMetro :: RecuperaDatosDeFichero (const char * nombre)
{
        ifstream fi (nombre);

        const int MAX_CADENA = 100;
        char cadena[MAX_CADENA];

        fi.getline(cadena, MAX_CADENA);// Lectura palabra mágica

        if (!strcmp(cadena, "METRO")) {

           fi >> (*this);  // Operador >> de la clase RedMetro

        } // if (!strcmp(cadena, "METRO"))

        fi.close();
}
/*****************************************************************************/
```


Reservados todos los derechos. No se permite la explotación económica ni la transformación de esta obra. Queda permitida la impresión en su totalidad.

Recordatorio de Wuolah: HOY SI QUE ERES UN CRACK #KeepCalm y #estudiaUnPoquito

Gana dinerito extra.
Recomienda a tus negocios favoritos que se anuncien en Wuolah y llévate 50€.
Te daremos un código promocional para que puedan anunciarse desde 99€.
1 Ve a tu negocio favorito • 2 Dales tu código de promo • 3 Diles que nos llamen o nos escriban.

EJERCICIO 4. Sobrecarga de operadores relacionales

La comparación entre dos redes puede realizarse en base a asignar a cada red un valor numérico. Ese valor se calculará con un nuevo método público (**ValorCalidad**) como indica el enunciado usando dos valores:

1) **El número de líneas.** Se conoce por el método **GetNumLineas** (ó el campo **num_lineas**)
2) **El número de paradas activas.** No hay ningún método disponible para conocer este valor. Podría a) implementarse o b) calcularse sin más en el propio método **ValorCalidad**. Como no encontramos utilidad a un nuevo método decidimos calcular el número de paradas en el propio método. Implementar un nuevo método (público o privado) para este cálculo sería también válido, se trata de una decisión de diseño (ver pregunta 5: lo implemento).

```
/*****************************************************************************/

double RedMetro :: ValorCalidad (void) const
{
        int cont_paradas_total = 0;
        int cont_paradas_no_activas_total = 0;

        for (int l=0; l<num_lineas; l++) {

                Linea la_linea = lineas[l];  // Nota 1: Asignación entre objetos de clase Linea

                int paradas_linea = la_linea.GetNumParadas();

                // Contar paradas no activas de la linea

                int cont_paradas_no_activas = 0;

                for (int p=1; p<=paradas_linea; p++) {

                        if (!(la_linea[p].EstaActiva()))   // Nota 2: Operator [] de "Linea"
                                cont_paradas_no_activas++;
                }

                // Acumular contadores
                cont_paradas_total += paradas_linea;
                cont_paradas_no_activas_total+=cont_paradas_no_activas;
        }

        // Calcular valor de calidad
        int cont_paradas = cont_paradas_total-cont_paradas_no_activas_total;
        double valor = 0.7*cont_paradas + 0.3*num_lineas;

        return (valor);
}

/*****************************************************************************/
```

Nota 1: No se trata del operator [] de la clase **RedMetro** sino del acceso a una casilla del array referenciado por **lineas**.

Nota 2: Ahora sí se trata del **operator []** para obtener la parada **p** (**p** >= 1) de la línea **la_linea**

653
811
910
WUOLAH

Reservados todos los derechos. No se permite la explotación económica ni la transformación de esta obra. Queda permitida la impresión en su totalidad.

Los operadores relacionales (todos) tienen como prototipos:

```
        bool operator == (const RedMetro & otra);
        bool operator != (const RedMetro & otra);
        bool operator >  (const RedMetro & otra);
        bool operator >= (const RedMetro & otra);
        bool operator <  (const RedMetro & otra);
        bool operator <= (const RedMetro & otra);
```

e implementaciones:

```
/*****************************************************************************/
bool RedMetro :: operator == (const RedMetro & otra)
{
        double valor_this = ValorCalidad();
        double valor_otra = otra.ValorCalidad();

        return (SonIguales(valor_this, valor_otra));   // Nota 1
}
/*****************************************************************************/
bool RedMetro :: operator != (const RedMetro & otra)
{
        return (!(*this == otra));
}
/*****************************************************************************/
bool RedMetro :: operator > (const RedMetro & otra)
{
        return (ValorCalidad() > otra.ValorCalidad());
}
/*****************************************************************************/
bool RedMetro :: operator < (const RedMetro & otra)
{
        return (ValorCalidad() < otra.ValorCalidad());
}
/*****************************************************************************/
bool RedMetro :: operator >= (const RedMetro & otra)
{
        return (!(*this < otra));
}
/*****************************************************************************/
bool RedMetro :: operator <= (const RedMetro & otra)
{
        return (!(*this > otra));
}
/*****************************************************************************/
```

Nota 1: Debe implementar la función **bool SonIguales** como es de sobra conocido para comparar dos números con decimales.

```
const double UMBRAL_IGUALES = 1e-6;

/*****************************************************************************/
bool SonIguales (double uno, double otro)
{
        return (fabs(uno-otro)<UMBRAL_IGUALES);
}
/*****************************************************************************/
```


WUOLAH
Reservados todos los derechos. No se permite la explotación económica ni la transformación de esta obra. Queda permitida la impresión en su totalidad.
Recordatorio de Wuolah: HOY SI QUE ERES UN CRACK #KeepCalm y #estudiaUnPoquito

EJERCICIO 5. Métodos de cálculo

5.a Método **MejorConectada**

Se pide la parada mejor conectada, en la que confluye el mayor número de líneas. Es evidente que se trata de un método de la clase **RedMetro.** Suponemos que la red NO está vacía.

Para realizar el cálculo se empleará un array de contadores, uno para cada parada de la red. Como las paradas se numeran desde 1 y no hay "huecos" habrán tantas casillas como paradas (el método **GetNumeroTotalParadas** -disponible- lo proporciona). Después, se calcula el valor máximo del array. La casilla donde se encuentra el máximo está asociada a la parada buscada.

```
/*****************************************************************************/
// PRE: La red no está vacía

int RedMetro :: MejorConectada (void) const
{
        // Calcular contadores

        int num_total_paradas = GetNumTotalParadas();
        int * lineas_pasan = new int [num_total_paradas];

        for (int p=0;p<num_total_paradas; p++)
                lineas_pasan[p] = 0;

        for (int l=0; l<num_lineas; l++) {

                Linea la_linea = lineas[l]; // acceso al array de objetos Linea

                int paradas_linea = la_linea.GetNumParadas();

                for (int p=1; p<=paradas_linea; p++) {
                        int indice_parada = (la_linea[p]).GetIndice(); // Operator [] sobre Linea
                        lineas_pasan[indice_parada-1]++;
                } // for p
        } // for l

        // Cálculo del máximo

        int num_paradas_mejor_conectada = lineas_pasan[0];
        int indice_mejor_conectada = 1;

        for (int p=2; p<=num_total_paradas; p++) {

                if (lineas_pasan[p-1] > num_paradas_mejor_conectada) {
                        num_paradas_mejor_conectada = lineas_pasan[p-1];
                        indice_mejor_conectada = p;
                }
        } // for p

        delete [] lineas_pasan;

        return (indice_mejor_conectada);
}
/*****************************************************************************/
```


WUOLAH
Reservados todos los derechos. No se permite la explotación económica ni la transformación de esta obra. Queda permitida la impresión en su totalidad.
Recordatorio de Wuolah: HOY SI QUE ERES UN CRACK #KeepCalm y #estudiaUnPoquito

5.b Método **EstaTotalmenteOperativa**

Nos indica si una línea está totalmente operativa así que debe ser un método de la clase **Linea.**

La manera más eficaz de implementar el método es con un ciclo while para poder detener la búsqueda cuando se encuentre una parada inactiva. Dos maneras de hacerlo:

```
/*****************************************************************************/
// PRE: La línea no está vacía

bool Linea :: EstaTotalmenteOperativa () const
{
      bool sigo = true;
      int p=0;

      while ((sigo) && (p<num_paradas)) {

          // acceso al array de datos InfoParada
          // con el puntero "paradas"

          InfoParada parada = paradas[p];

          if (!(parada.EstaActiva())) sigo = false;
          else p++;

      }

      return (sigo);

}

/*****************************************************************************/
```

```
      bool sigo = true;
      int p = 1

      while ((sigo) && (p<=num_paradas)) {

          // acceso al array de datos InfoParada
          // con el operador [] de Linea

          InfoParada parada = (*this) [p];

          if (!(parada.EstaActiva())) sigo = false;
          else p++;

      }

      return (sigo);
```

Otra solución está basada en calcular el número de paradas activas y compararlo con el número de paradas de la línea. Es preciso escribir el método que calcule el número de paradas activas:

```
/*****************************************************************************/
int Linea :: GetNumParadasActivas () const
{
        int cont_activas = 0;

        for (int p=0; p<num_paradas; p++)

          if (paradas[p].EstaActiva())
              cont_activas++;

        return (cont_activas);

}
/*****************************************************************************/
```


WUOLAH
Reservados todos los derechos. No se permite la explotación económica ni la transformación de esta obra. Queda permitida la impresión en su totalidad.
Recordatorio de Wuolah: HOY SI QUE ERES UN CRACK #KeepCalm y #estudiaUnPoquito

Gana dinerito extra.
Recomienda a tus negocios favoritos que se anuncien en Wuolah y llévate 50€.
Te daremos un código promocional para que puedan anunciarse desde 99€.
1 Ve a tu negocio favorito • 2 Dales tu código de promo • 3 Diles que nos llamen o nos escriban.


```
/*****************************************************************************/

bool Linea :: EstaTotalmenteOperativa () const
{
        return (GetNumParadas()==GetNumParadasActivas());
}
/*****************************************************************************/
```

Como comentario general a los dos ejercicios de la pregunta 5 he observado (nuevamente) accesos a los **campos privados** de las otras clases, lo que es inaceptable.

Para finalizar, reescribo el método **ValorCalidad** usando el nuevo método de la clase **Linea** GetNumParadasActivas():

```
/*****************************************************************************/

double RedMetro :: ValorCalidad (void) const
{
        int num_lineas = GetNumLineas();

        int cont_total_paradas_activas = 0;

        for (int l=0; l<num_lineas; l++) {

                Linea la_linea = lineas[l];  // Nota 1: Asignación entre objetos de clase Linea

                cont_total_paradas_activas  += la_linea.GetNumParadasActivas();
        }

        // Calcular valor de calidad
        double valor = 0.7 * cont_total_paradas_activas  + 0.3 * num_lineas;

        return (valor);
}

/*****************************************************************************/
```

653
811
910

WUOLAH

WUOLAH

Reservados todos los derechos. No se permite la explotación económica ni la transformación de esta obra. Queda permitida la impresión en su totalidad.

EJERCICIO 6. Aplicación

```
/**************************************************************************/
// METODOLOGIA DE LA PROGRAMACION
// GRADO EN INGENIERIA INFORMATICA
//
// CURSO 2017-2018
// (C) FRANCISCO JOSE CORTIJO BON
// DEPARTAMENTO DE CIENCIAS DE LA COMPUTACION E INTELIGENCIA ARTIFICIAL
//
// Examen ordinario 2017/2018
//
// Fichero: examen.cpp
//
/**************************************************************************/

#include <iostream>
#include <fstream>
#include <iomanip>
#include <cstdlib>

#include "InfoParada.h"
#include "Linea.h"
#include "RedMetro.h"

using namespace std;

/**************************************************************************/
/**************************************************************************/

bool ExisteFichero (const char *nombre); // De sobra conocido

/**************************************************************************/
/**************************************************************************/

int main (int argc, char **argv)
{
        if (argc != 3) {
                cerr << "Error: numero de argumentos incorrecto." << endl;
                cerr << "Uso: " << argv[0] << " <fich1> <fich2>" << endl;
                exit (1);
        }

        if (!ExisteFichero(argv[1])) {
                cerr << "Error: No se puede abrir " << argv[1] << endl;
                exit (2);
        }

        if (!ExisteFichero(argv[2])) {
                cerr << "Error: No se puede abrir " << argv[2] << endl;
                exit (3);
        }

        ifstream fi1 (argv[1]);
        ifstream fi2 (argv[2]);
```


WUOLAH
Reservados todos los derechos. No se permite la explotación económica ni la transformación de esta obra. Queda permitida la impresión en su totalidad.
Recordatorio de Wuolah: HOY SI QUE ERES UN CRACK #KeepCalm y #estudiaUnPoquito

```
RedMetro red1 (argv[1]);
if (red1.EstaVacia()) {
        cerr << "Error: Formato incorrecto en " << argv[1] << endl;
        exit (4);
}

RedMetro red2 (argv[2]);
if (red2.EstaVacia()) {
        cerr << "Error: Formato incorrecto en " << argv[2] << endl;
        exit (6);
}

// Todo correcto : empezamos

cout << endl;
cout << "-------------------------------" << endl;
cout << endl;

cout << "red1 se  forma a partir de " << argv[1] << endl;
cout << "red2 se  forma a partir de " << argv[2] << endl;

cout << endl;
cout << "-------------------------------" << endl;
cout << endl;

cout << endl;
cout << "red1 y red2 son " << ((red1 == red2) ? "iguales" : " diferentes");
cout << endl;

cout << endl;
cout << "-------------------------------" << endl;
cout << endl;

RedMetro la_mejor;

if (red1 > red2) {
        cout << "La mejor red es red1" << endl;
        la_mejor = red1;
}
else {
        cout << "La mejor red es red2" << endl;
        la_mejor = red2;
}

cout << endl;
cout << "-------------------------------" << endl;
cout << endl;

cout << "Informacion sobre la operatividad de la mejor red" << endl;
cout << endl;

for (int l=1; l<=la_mejor.GetNumLineas(); l++) {

        cout << "\tLinea " << setw(3) << l << ": ";

        if (la_mejor[l].EstaTotalmenteOperativa())  cout << "totalmente operativa." << endl;
        else cout << "tiene paradas no operativas." << endl;
}
```


WUOLAH
Reservados todos los derechos. No se permite la explotación económica ni la transformación de esta obra. Queda permitida la impresión en su totalidad.
Recordatorio de Wuolah: HOY SI QUE ERES UN CRACK #KeepCalm y #estudiaUnPoquito

```
        cout << endl;
        cout << "--------------------------------" << endl;
        cout << endl;

        cout << "Parada mejor conectada de la mejor red" << endl;
        cout << endl;

        cout << "\tEl indice de la parada mejor conectada es: "
             << la_mejor.MejorConectada() << endl;

        cout << endl;
        cout << "--------------------------------" << endl;
        cout << endl;

        return (0);
}

/*****************************************************************************/
/*****************************************************************************/
```


WUOLAH

Reservados todos los derechos. No se permite la explotación económica ni la transformación de esta obra. Queda permitida la impresión en su totalidad.

Recordatorio de Wuolah: HOY SI QUE ERES UN CRACK #KeepCalm y #estudiaUnPoquito