UNIVERSIDAD POLITÉCNICA DE MADRID
ESCUELA TÉCNICA SUPERIOR DE INGENIERÍA Y DISEÑO INDSUTRIAL

Departamento de Ingeniería Eléctrica, Electrónica, Automática y Física Aplicada

APELLIDOS

NOMBRE Nº Mat.

ASIGNATURA: SISTEMAS INFORMÁTICOS INDUSTRIALES

CURSO 4º **GRUPO** **Octubre 2014**

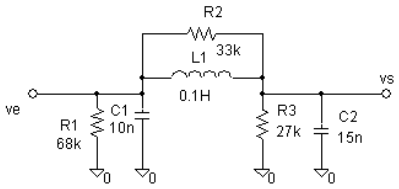
Calificación

2. Problema de Análisis y Diseño Orientado a Objetos (10 puntos - 30 minutos)

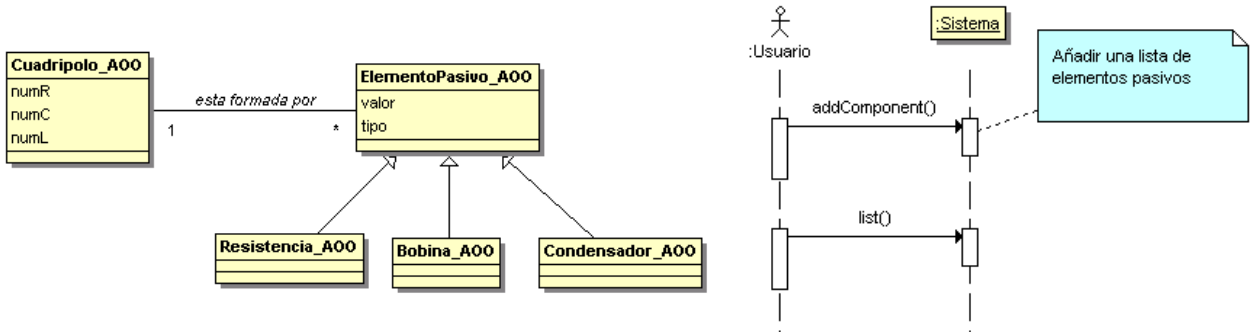
Se pretende realizar un simulador de cuadripolos eléctricos pasivos. Estas redes sólo están formadas por la combinación de resistencias, condensadores y bobinas. En esta primera iteración se pretende guardar la información de los componentes y su listado.

1. Modelo del dominio y diagrama de secuencia del sistema (2.5 puntos).
2. Diagrama de clases de diseño (2.5 puntos).
3. Implementación en C++ de la solución (5 puntos).

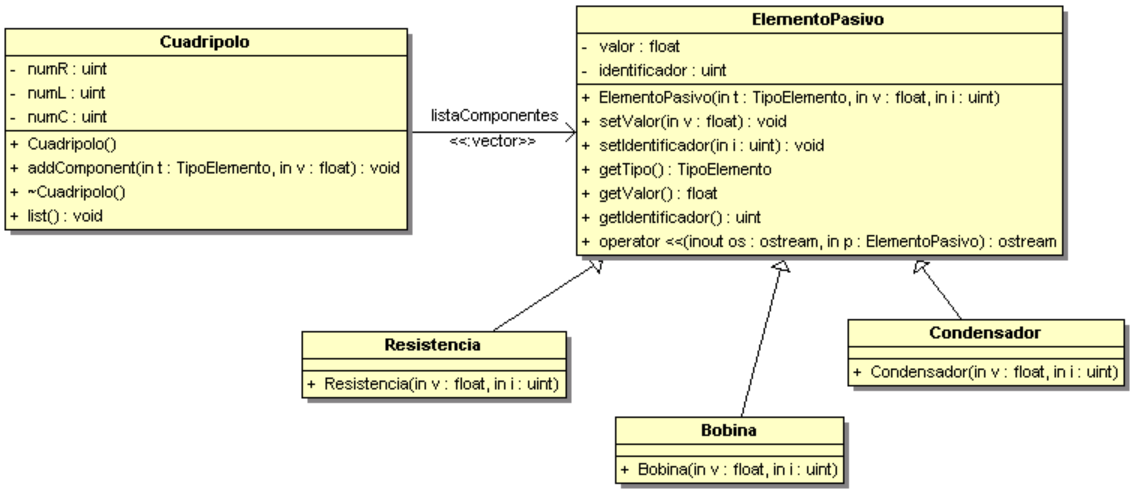
```
// Código de test
int main()
{
        Cuadripolo red;
        red.addComponent(R,68e3);
        red.addComponent(R,33e3);
        red.addComponent(R,27e3);
        red.addComponent(L,0.1);
        red.addComponent(C,1e-8);
        red.addComponent(C,1.5e-8);
        red.list();

        return 0;
}

//////////////////////////////////////
//Resultado en consola
//////////////////////////////////////
Resistencia 1: valor = 68000
Resistencia 2: valor = 33000
Resistencia 3: valor = 27000
Bobina 1: valor = 0.1
Condensador 1: valor = 1e-008
Condensador 2: valor = 1.5e-008
```



1.



2.



3.

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

typedef enum{R,L,C} TipoElemento;

class ElementoPasivo {
        TipoElemento tipo;
        float valor;
        unsigned identificador;
public:
        ElementoPasivo(TipoElemento t,float v,unsigned i):
          tipo(t), valor(v),identificador(i) {}
        void setValor(float v) {valor=v;}
        void setIdentificador(unsigned i) {identificador=i;}
        TipoElemento getTipo() { return tipo;}
        float getValor() {return valor;}
        unsigned getIdentificador() {return identificador;}
        friend ostream& operator<<(ostream& os, const ElementoPasivo * p) {
                if(p->tipo == R) return os << "Resistencia " << p->identificador
                        << ": valor = " << p->valor << endl;
                else if(p->tipo == L) return os << "Bobina " << p->identificador
                        << ": valor = " << p->valor << endl;
                else return os << "Condensador " << p->identificador
                        << ": valor =" << p->valor << endl;
    }
};

class Resistencia: public ElementoPasivo {
public:
        Resistencia(float v,unsigned i): ElementoPasivo(R,v,i) {}
};

class Bobina: public ElementoPasivo {
public:
        Bobina(float v,unsigned i): ElementoPasivo(L,v,i) {}
};

class Condensador: public ElementoPasivo {
public:
        Condensador(float v,unsigned i): ElementoPasivo(C,v,i) {}
};

void visualizar(ElementoPasivo *p){
        if(p->getTipo() == R) cout << "Resistencia " << p->getIdentificador()
                << ": valor = " << p->getValor() << endl;
        else if(p->getTipo() == L) cout << "Bobina " << p->getIdentificador()
                << ": valor = " << p->getValor() << endl;
        else cout << "Condensador " << p->getIdentificador()
                << ": valor =" << p->getValor() << endl;
}

class Cuadripolo {
        unsigned numR;
        unsigned numL;
        unsigned numC;
        std::vector<ElementoPasivo *> listaComponentes;
public:
        Cuadripolo():numR(0),numL(0),numC(0){}
        void addComponent(TipoElemento t, float v){
                if(t==R)
                        listaComponentes.push_back(new Resistencia(v,++numR));
                else if(t==L)
                        listaComponentes.push_back(new Bobina(v,++numL));
                else
                        listaComponentes.push_back(new Condensador(v,++numC));
        };
        ~Cuadripolo(){
                for(unsigned i=0;i<listaComponentes.size();i++)
                        delete listaComponentes[i];
        }
        void list(){
//              for(unsigned i=0;i<listaComponentes.size();i++)
//                      cout << listaComponentes[i];
                for_each(listaComponentes.begin(),listaComponentes.end(),visualizar);

        }
};
```

escuela técnica superior de ingeniería y diseño industrial

UNIVERSIDAD POLITÉCNICA DE MADRID
ESCUELA TÉCNICA SUPERIOR DE INGENIERÍA Y DISEÑO INDSUTRIAL

Departamento de Ingeniería Eléctrica, Electrónica, Automática y Física Aplicada

APELLIDOS

NOMBRE Nº Mat.

ASIGNATURA: SISTEMAS INFORMÁTICOS INDUSTRIALES

CURSO 4º **GRUPO** **Octubre 2014**

Calificación

3. Problema de Sistemas Operativos (10 puntos - 30 minutos)

En una interfaz alfanumérica de UNIX (*shell*) se desea ejecutar tres mandatos enlazados mediante tuberías. El formato de la ejecución es el siguiente:

```
filtro1 | filtro2 | filtro3
```

Así los mandatos están programados como **filtros**. Cogen los datos de la entrada estándar y dan los resultados por la salida estándar. Lo que se consigue de esta forma es poder enlazar la salida de un mandato con la entrada del siguiente. De esta manera podemos asociar diversos **filtros** para realizar un procesamiento en línea (*pipeline*). El último de los mandatos presentará los resultados por la salida estándar. Así por ejemplo, si ponemos en una interfaz de mandatos las siguientes órdenes se observan los siguientes resultados:

Caso 1: Ejecuto mandato 'ls' que lista los ficheros y directorios del directorio donde me encuentro.

```
$ ls
Descargas   Dropbox     Imágenes  mnt   Música      Público  Ubuntu One
```

Caso 2: Reenvío la salida de 'ls' al mandato 'sort' que ordena alfabéticamente lo recibido por la entrada estándar y lo envía a su salida estándar.

```
$ ls | sort
Descargas
Dropbox
Imágenes
mnt
Música
Público
Ubuntu One
```

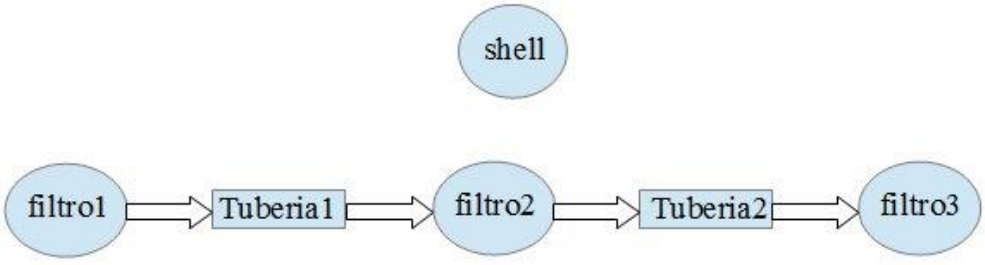
Caso 3: Reenvío la salida de 'sort' a 'head' con un 2 como parámetro, mostrando por salida estándar un número de líneas, en este caso 2, de la cabecera de lo que recibe por su entrada estándar.

```
$ ls | sort | head -2
Descargas
Dropbox
```

Caso 4: Reenvío la salida de 'sort' al mandato 'tail' con 3 como parámetro, mostrando por salida estándar las 3 últimas líneas de lo recibido por 'tail' en su entrada estándar.

```
$ ls | sort | tail -3
Música
Público
Ubuntu One
```

Para el caso genérico propuesto tendremos el esquema de procesos representado en la siguiente figura, donde el proceso "shell" creará ***n*** procesos y ***n-1*** tuberías para comunicarlos. Además el proceso *shell*, antes de hacer la ejecución de los programas ***mandato-i***, deberá redireccionar las entradas y salidas estándar y cerrar aquellas entradas y salidas de las tuberías que no se vayan a utilizar.



Responda de forma breve y concisa las siguientes cuestiones:

1. (0,5 pto) para los casos 1, 2, 3 y 4, diga cuántos procesos nuevos se crean.

- Caso 1: Se crea uno, el que ejecuta el programa 'ls'.
- Caso 2: Se crean dos, el que ejecuta 'ls' y el que ejecuta el programa 'sort.
- Caso 3: Se crean tres; 'ls', 'sort' y 'head'.
- Caso 4: Se crean tres; 'ls'. 'sort' y 'tail'.

2. (0,5 pto) para los casos 1, 2, 3 y 4 diga qué proceso crea qué otro.

◦ Para todos los casos todos los procesos nuevos son creados por el proceso *shell.*

3. (1 pto) Qué llamada del sistema operativo se usa para crear un proceso. Explique cómo funciona y ponga un pequeño ejemplo.

◦ La única llamada al sistema operativo para crear un proceso es fork()

◦ Al invocarse se crea un nuevo proceso hijo y por tanto aparece un nuevo hilo de ejecución. Para el padre, fork() devuelve el pid del hijo. Para el hijo fork() devuelve un cero. En caso de error fork() devuelve un menos 1.

4. **(0,5 pto) Para los casos 1, 2, 3 y 4 diga cuantos pipes se crean.**
   - Caso1. Ninguno
   - Caso 2. Uno
   - Caso 3 y 4: Dos
5. **(0,5 pto) para los casos 1, 2, 3 y 4 diga que proceso crea cada una de los pipes.**
   - Todos los pipes son creados por el proceso shell
6. **(0,5 pto) Para el caso 1 diga qué llamada al sistema realiza el proceso '*shell'* para esperar por la terminación del proceso que ejecuta *'ls'*. Diga también qué llamada al sistema invoca el proceso 'ls' para terminar su ejecución. ¿El hijo pasa algún valor al padre?**

• El shell habrá invocado a la llamada al sistema wait() para esperar por la terminación del hijo. Al terminar 'ls' le pasará el estado al padre por el parámetro enviado por la llamada exit().

7. **(0,5 pto) Para el caso 2 explique en qué orden mueren los procesos y cómo**
   - Lógicamente hasta que no haya terminado 'ls' no podrá sacar los resultados de ordenación 'sort'. Por tanto seguro que terminará primero 'ls' con exit() y luego 'sort' con exit().
8. **(1,5 pto) Para el caso 3 explique en qué orden mueren los procesos y cómo**
   - Idem que antes, 'ls' tiene que terminar antes de 'sort'.
   - Entre 'sort' y 'head' no se sabe seguro. Depende del planificador. Hay dos posibilidades.
   - Como 'head' sólo saca las dos primeras líneas de lo que le pasa 'sort', 'head' podría leer esas dos líneas y terminar antes que 'sort'. 'head' terminará con exit() y 'sort' recibira la señal SIGPIPE al intentar escribir en una tubería donde no hay lectores al otro lado. La respuesta por defecto de esta señal es matar el proceso.
   - Otro posible caso es que 'sort' se ejecutara del tirón y escribiera en el segundo pipe todo su resultado y terminara. Seguidamente se ejecutara 'head', pero sólo leería las dos primeras líneas, las sacara por su salida estándar y terminara. En este caso los dos procesos mueren con la llamada exit()
9. **(1 pto) Para el caso 4 explique en qué orden mueren los procesos y cómo**
   - Idem que antes, 'ls' tiene que terminar antes de 'sort'. Ambos mueren por exit()
   - Entre 'sort' y 'tail' ahora solo hay una posibilidad. Ya que 'tail' tiene que esperar a recibir todos las líneas para sacar las tres últimas. Por tanto siempre terminará primero 'sort' y luego 'tail'. Ambos mueren por exit().
10. **(0,5 pto) Diga cuál es la llamada al sistema que invocará cada proceso para cargar el programa 'ls', 'sort', 'head' o 'tail'.**
    - El conjunto de llamadas exec()
11. **(1 pto) Si la tabla de descriptores de fichero para el proceso 'ls' para el caso 1 es:**

índice	dispositivo
0	**STDIN**
1	**STDOUT**
2	**STDERR**
3	**NULL**

Indique como queda dicha tabla cuando las tuberías están conectadas para los procesos 'ls' y ' sort' en el caso 2.

Proceso ls		Proceso sort	
índice	**dispositivo**	**índice**	**dispositivo**
0	STDIN	0	PIPE_OUT
1	PIPE_IN	1	STDOUT
2	STDERR	2	STDERR
3	NULL	3	NULL

12 **(2 pto) En el caso anterior explique los pasos a seguir para interconectar adecuadamente la tubería con los dos procesos, desde que se crean estos y la tubería hasta que comienza la ejecución de los programas 'ls' y 'sort'.**

1. El proceso shell ejecutará las siguientes llamadas:
1. Crea tubería con pipe()
2. Crea proceso para 'ls' con fork(). Éste hereda descriptores de la tubería.
3. Crea proceso para 'sort' con fork(). Éste hereda descriptores de la tubería.
2. El proceso para 'ls':
1. Cierra STDOUT con close(STDOUT)
2. Duplica la entrada de la tubería con dup(PIPE_IN) asignándole el descriptor 1.
3. Cierra entrada de la tubería con close(PIPE_IN)
4. Cierra salida de la tubería con close(PIPE_OUT)
5. Ejecuta el programa 'ls' con execl("ls",NULL)
3. El proceso para 'sort':
1. Cierra STDIN con close(STDIN)
2. Duplica la salida de la tubería con dup(PIPE_OUT) asignándole a esta el descriptor cero.
3. Cierra entrada de la tubería con close(PIPE_IN)
4. Cierra salida de la tubería con close(PIPE_OUT)
5. Ejecuta el programa 'sort' con execl("sort",NULL)

UNIVERSIDAD POLITÉCNICA DE MADRID
ESCUELA TÉCNICA SUPERIOR DE INGENIERÍA Y DISEÑO INDSUTRIAL

Departamento de Ingeniería Eléctrica, Electrónica, Automática y Física Aplicada

APELLIDOS

NOMBRE Nº Mat.

ASIGNATURA: SISTEMAS INFORMÁTICOS INDUSTRIALES

CURSO 4º **GRUPO** **Octubre 2014**

Calificación

3. Problema de STL (5 puntos - 15 minutos)

Complete adecuadamente el código que figura a continuación sustituyendo *** por la LINEA de código adecuada. La función MAIN no requiere modificación. Conteste en la hoja del enunciado.

```
int main(){
 vector<Student> v;
 v.push_back(Student("juan", 4.0, 10010));
 v.push_back(Student("pedro", 3.0, 10006));
 v.push_back(Student("lucas", 2.0, 10001));

 //ordenamiento por notas crecientes
 sort(v.begin(), v.end(), less_marks() );
 printCol(v); cout<<endl;

 //ordenamiento por id decreciente
 sort(v.begin(), v.end(), less_ids() );
 printCol(v); cout<<endl;

 //cálculo de la media de notas de todos los estudiantes
 double average=accumulate(v.begin(), v.end(), 0.0 , sum_students());
 cout<<"nota media:"<<average/v.size()<<endl;

 //incremento de 0.5 puntos en todas las calificaciones
 for_each(v.begin(), v.end(), inc_mark(0.5));
 printCol(v);

return 0;
}
```

```
//… includes (no completar)
using namespace std;

template<class Collection>
void printCol(Collection& col, ostream& o=cout){
   ***
}

class Student{
public:
 friend ostream & operator<<(ostream& o, const Student& s){
   ***
   return o;
}

 Student(string name, int mark, int id): name(name), mark(mark), id(id){}
 double get_mark() const {return mark;}
 void set_mark(double mark_new){ mark=mark_new;}
 int get_id() const {return id;}
 string get_name() const {return name;}

private:
 string name;
 double mark;
 int id;
};
```

```
//reordemaniento de estudiantes por nota creciente
struct less_marks{
public:
 bool operator() (const Student& a, const Student& b){
       ***
 }
};

//reordemaniento de estudiantes por id decreciente
struct less_ids{
public:
 bool operator() (const Student& a, const Student& b){
       ***
 }
};

//functor para incremento de la nota de los estudiantes
struct inc_mark{
    ***
 void operator()(Student& s){
    ***
  }
private:
  double inc;
};

//functor para sumar la nota de todos los estudiantes
struct sum_students{
    ***

};
```

SOLUCION

```
#include <iostream>
#include <vector>
#include <algorithm>
#include <list>
#include <numeric>
#include <iterator>

using namespace std;

template<class Collection>
void printCol(Collection& col, ostream& o=cout){
       copy(col.begin(), col.end(), ostream_iterator<Collection::value_type>(cout, " "));
}

class Student{
public:
       friend ostream & operator<<(ostream& o, const Student& s){
              o<<s.name.c_str()<<":"<<s.mark<<":"<<s.id<<endl;
              return o;
       }

       Student(string name, int mark, int id): name(name), mark(mark), id(id){}
       double get_mark() const {return mark;}
       void set_mark(double mark_new){ mark=mark_new;}
       int get_id() const {return id;}
       string get_name() const {return name;}
```

```
private:
    string name;
    double mark;
       int id;
};

struct less_marks{
public:
       bool operator() (const Student& a, const Student& b){
              return (a.get_mark()<b.get_mark());
       }
};

struct less_ids{
public:
       bool operator() (const Student& a, const Student& b){
              return (a.get_id()>b.get_id());
       }
};

struct inc_mark{
       inc_mark(double inc_new):inc(inc_new){}
       void operator()(Student& s){
              s.set_mark(s.get_mark()+inc);
       }
private:
       double inc;
};

struct sum_students{
       double operator()(double d, const Student& s){ return (s.get_mark()+d);}
};
```