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

Departamento Electrónica, Automática e Informática Industrial

APELLIDOS

NOMBRE Nº Mat.

Calificación

ASIGNATURA: SISTEMAS INFORMÁTICOS INDUSTRIALES

CURSO 4º GRUPO Diciembre 2013

2. Problema de ADOO (10 puntos - 40 minutos)

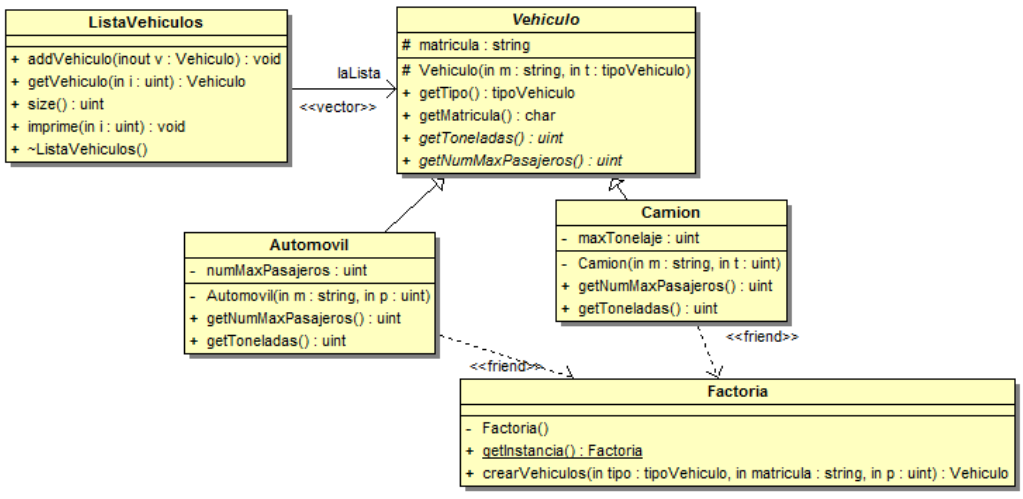
Para una prueba de concepto sobre la gestión de vehículos de un municipio se ha diseñado el siguiente código, al cual se ha añadido el resultado de la ejecución de esta versión. En todos los vehículos se registra la matrícula. Si el vehículo es un camión se anota el máximo de toneladas que puede cargar. Para el resto se almacena el número de pasajeros del automóvil.

1. Diagrama de clase de diseño DCD en UML de las clases *Vehiculo*, *Automovil*, *Camion*, *ListaVehiculos* y *Factoria*. Indique los patrones empleados (5 puntos).
2. Implementación de estas clases en C++ (5 puntos).

```
#include <iostream>
#include <vector>
using namespace std;
typedef enum{AUTO,CAMION} tipoVehiculo;
class Vehiculo{…};

class Automovil{…};

class Camion{…};

class ListaVehiculos{…};

class Factoria{…};

void add_autos(ListaVehiculos &);
void add_camiones(ListaVehiculos &);

int main()
{
        ListaVehiculos lista;
        add_autos(lista);
        add_camiones(lista);
        for(unsigned i=0;i<lista.size();i++)
                lista.imprime(i);

        return 0;
}

void add_autos(ListaVehiculos & lista)
{
 Factoria laFactoria = Factoria::getInstancia();
 lista.addVehiculo(laFactoria.crearVehiculos(AUTO,"1234 AAA", 5));
 lista.addVehiculo(laFactoria.crearVehiculos(AUTO,"5678 EEE", 7));
}

void add_camiones(ListaVehiculos & lista)
{
 Factoria laFactoria = Factoria::getInstancia();
 lista.addVehiculo(laFactoria.crearVehiculos(CAMION,"4321 BBB", 20));
 lista.addVehicul
}
```

```
Matricula: 1234 AAA
Auto con numero maximo pasajeros: 5
Matricula: 5678 EEE
Auto con numero maximo pasajeros: 7
Matricula: 4321 BBB
Camion con maximo de toneladas: 20
Matricula: 8765 CCC
Camion con maximo de toneladas: 15
```

1.



2.

```
class Vehiculo{
protected:
        string matricula;
        tipoVehiculo tipo;
        Vehiculo(const string m,tipoVehiculo t): matricula(m),tipo(t){}
public:
        tipoVehiculo getTipo() {return tipo;}
        const char * getMatricula() {return matricula.c_str();}
        virtual unsigned getToneladas() = 0;
        virtual unsigned getNumMaxPasajeros() = 0;
};
class Automovil : public Vehiculo{
        friend class Factoria;
        unsigned numMaxPasajeros;
        Automovil(const string m,const unsigned p):Vehiculo(m,AUTO),numMaxPasajeros(p) {}
public:
        unsigned getNumMaxPasajeros() { return numMaxPasajeros; }
        unsigned getToneladas() { return 0; }
};
class Camion : public Vehiculo{
        friend class Factoria;
        unsigned maxTonelaje;
        Camion(const string m,const unsigned t):Vehiculo(m,CAMION),maxTonelaje(t) {}
public:
        unsigned getNumMaxPasajeros() { return 0; }
        unsigned getToneladas() { return maxTonelaje; }

};
class ListaVehiculos{
        vector<Vehiculo *> laLista;
public:
        void addVehiculo(Vehiculo *v) {laLista.push_back(v);}
        Vehiculo * getVehiculo(unsigned i) { return laLista[i];}
        unsigned size() {return laLista.size();}
        void imprime(unsigned i) {
         cout<<"Matricula: "<< laLista[i]->getMatricula()<<endl;
         if(laLista[i]->getTipo()==AUTO)
           cout<<"Auto con numero maximo pasajeros: " << laLista[i]->getNumMaxPasajeros()<<endl;
         else
          cout<<"Camion con maximo de toneladas: " << laLista[i]->getToneladas()<<endl;}
        ~ListaVehiculos() { for(unsigned i=0; i<laLista.size();i++) delete laLista[i]; }
};
class Factoria
{
        Factoria(){}
public:
    static Factoria& getInstancia(){
             static Factoria unicaInstancia;
             return unicaInstancia;
                }
    Vehiculo* crearVehiculos (tipoVehiculo tipo, const string matricula, const unsigned p )
        {
         if(tipo == AUTO) return new Automovil(matricula,p);
         else if(tipo == CAMION ) return new Camion(matricula,p);
         else return NULL;
        }
};
```

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

Departamento Electrónica, Automática e Informática Industrial

APELLIDOS

NOMBRE Nº Mat.

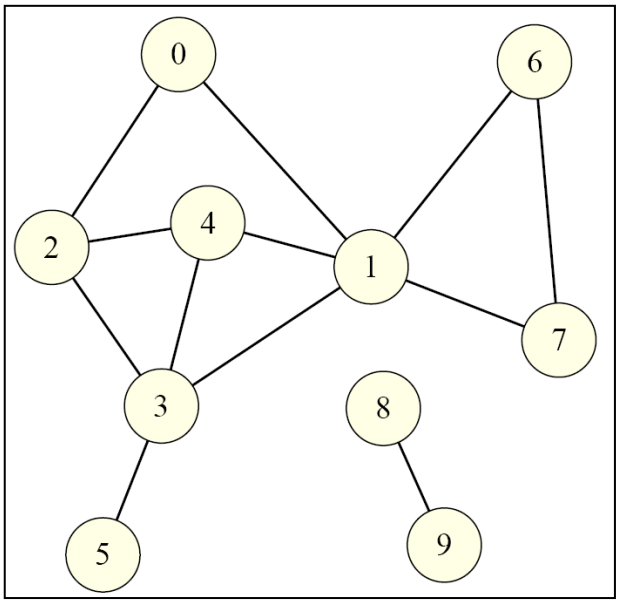
Calificación

ASIGNATURA: SISTEMAS INFORMÁTICOS INDUSTRIALES

CURSO 4º GRUPO Diciembre 2013

3. Problema de algoritmia (5 puntos - 30 minutos)

El grafo de la figura representa una configuración de trayectorias en una red local de transporte de una planta. Los números representan índices. Se pide:

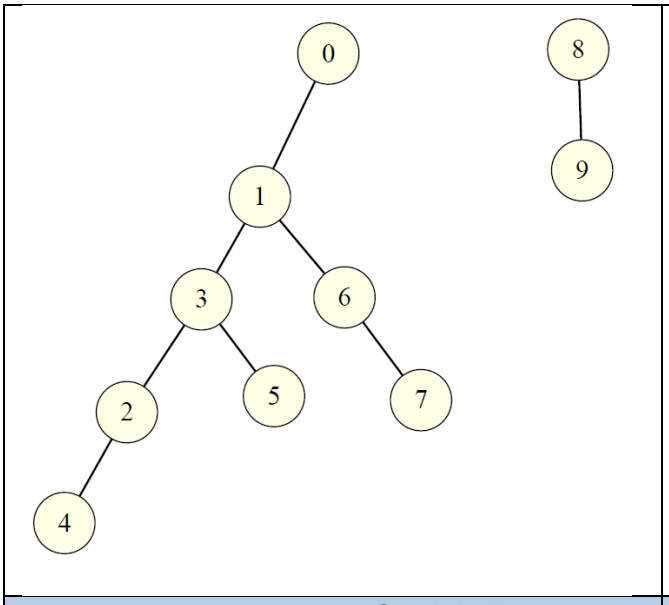


1. Árboles G_π primero en profundidad y en anchura resultado de lanzar los algoritmos correspondientes. Asuma que, a igualdad de criterio, el algoritmo siempre elige aquel vértice de numeración más baja.
2. Se pretende ordenar los nodos de la red por facilidad de suministro: se situarán primero aquéllos que sean más accesibles. Para ello se dispone de una función int *degree* (int v, const Graph& g) que calcula el grado del vértice v pasado. Implemente el algoritmo de ordenamiento en C++ (STL) mediante la función *sort* (nota: incluya el *functor* adecuado).
3. Implemente el pseudocódigo que aparece a continuación en una función C++, considerando que dispone de una colección de datos a ordenar en vector<int> *data*.

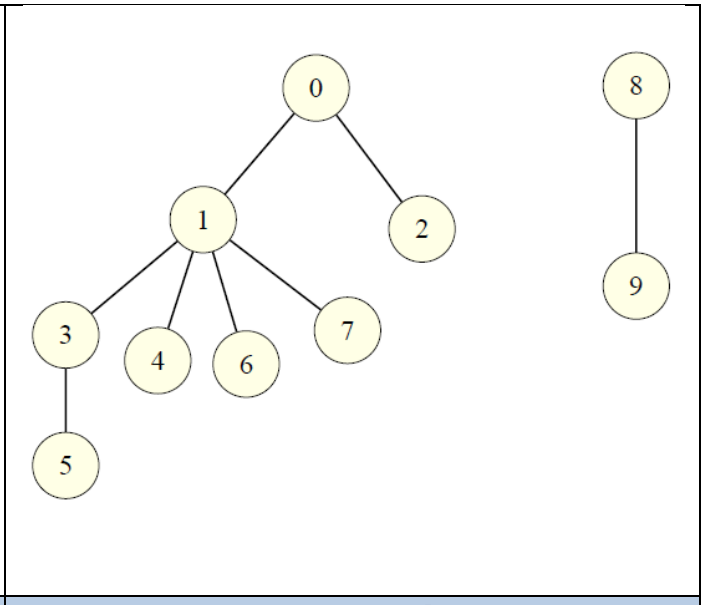
```
Input:
Output:
<PROCEDURE NAME> (A)
Initial step:
1. for j = 2 to A.length
2.    key = A[j]
 //Insert A[j] into the sorted sequence A[1]…A[j-1]
3.    i ← j - 1
4.    while i > 0 and A[i] > key
5.      A[i + 1] = A[i]
6.      i = i-1
7.    A[i + 1] = key
```

SOLUCION

3.1



Primero en profundidad



Primero en anchura

3.2

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

#define DEFAULT_SIZE 10
class GraphBB{
public:
	int size(){ return DEFAULT_SIZE;}
};

int degree(int v, const GraphBB& g){
//mock degree
static int k=0;
	return k++;
}

//functor
struct max_deg_order{
	bool operator() (int v1, int v2){
		return (v1>v2);
	}
};
int main(){
	GraphBB g;
	vector<int> vdeg(g.size(), 0);
	for(int v=0; v<vdeg.size(); v++){
		vdeg[v]=degree(v, g);
	}

	sort(vdeg.begin(), vdeg.end(), max_deg_order());
	copy(vdeg.begin(), vdeg.end(), ostream_iterator<int>(cout, " "));
}
```

3.3

```
void Sort::insert_sort(vector<int> data){
  if(data.empty()) return;
  int key;
  for(int j=1; j<data.size(); j++){
    key=_data[j];
    int i=j-1;
    while(i>=0  && _data[i]>key){
      data[i+1]=data[i];
      i=i-1;
    }
   data[i+1]=key;
  }
}
```

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

Departamento Electrónica, Automática e Informática Industrial

APELLIDOS

NOMBRE Nº Mat.

Calificación

ASIGNATURA: SISTEMAS INFORMÁTICOS INDUSTRIALES

CURSO 4º GRUPO Diciembre 2013

4. Problema de Sistemas Operativos (10 puntos - 40 minutos)

PARTE A

Se desea desarrollar una aplicación que debe realizar dos tareas que se pueden ejecutar de forma independiente. Los códigos de estas dos tareas se encuentran definidos en dos funciones cuyos prototipos en lenguaje de programación C son los siguientes:

```
void tarea_A(void);
void tarea_B(void);
```

Se pide programar la aplicación anterior utilizando dos modelos distintos:

a) Procesos (pesados) para aprovechar paralelismo. Plantear una solución que minimice el número de procesos creados y solicite el mínimo número de servicios al sistema operativo. **(2 puntos)**

```
int main(void)
{
   pid_t pid;

   pid = fork();
   if (pid == 0)
         tarea_B();  // el hijo ejecuta la tarea B
   else {
         tarea_A();  // el hijo ejecuta la tarea A
         wait(NULL); // el padre espera a que el hijo termine
   }
   return 0;
}
```

b) Procesos ligeros para aprovechar paralelismo. En este caso también se minimizará el número de procesos ligeros creados y el número de servicios. **(2 puntos)**

```
int main(void)
{
   pthread_t t;

   pthread_create(&t, NULL, tarea_B, NULL); // un proceso ligero ejecuta la
                                            // tarea B
   tarea_A();  // el proceso ligero principal ejecuta la tarea A
   pthread_join(t, NULL);  // el proceso principal espera a que termine el
                           // proceso que ejecuta la tarea B
   return 0;
}
```

PARTE B

Se pretende resolver un problema de comunicación y sincronización entre procesos ligeros empleando semáforos. Se trata del típico problema de lectores y escritores. Se dan las siguientes directrices:

- El proceso ligero principal debe crear 'MAX_LECTORES' procesos ligeros lectores y 'MAX_ESCRITORES' procesos ligeros escritores, que van a competir por un recurso común. En este caso, el recurso común es un simple dato de tipo entero (dato).

- Cada proceso ligero lector deberá leer el recurso en exclusión mutua frente a escritores. Es decir, mientras un lector este accediendo al recurso compartido ningún proceso ligero escritor podrá acceder a éste.

- De manera contraria, si el recurso está ocupado por un escritor, el lector deberá esperar que el escritor termine. Sin embargo el proceso ligero lector no debe mantener exclusión mutua frente a otros procesos ligeros lectores, ya que la lectura no es destructiva.

- El proceso ligero escritor debe mantener la exclusión mutua frente a procesos lectores, como ya se dijo, y también frente a procesos escritores.

Se plantea en un primer momento el código adjunto, pero la experiencia comprueba que es erróneo. Se pide:

a) Indica razonadamente qué fallo tiene el código. **(1 punto)**

El fallo es que se está haciendo exclusión mutua entre lectores y el enunciado dice explícitamente que no debe hacerse. Varios lectores deberían poder acceder simultáneamente a la sección crítica.

b) Escribe el código corregido. Solamente escribe las líneas de código que hay que añadir, indicando entre qué líneas deben ser colocadas. **(5 puntos)**

```
01 int dato = VALOR_INICIAL; /* recurso */
02 sem_t semaforo; /* acceso a dato */
03 int n_lectores = 0; /* número de lectores */
   sem_t sem_lec;      /* acceso n_lectores */
04 int main(void)
05 {
06     pthread_t th_lector[MAX_LECTORES],th_escritor[MAX_ESCRITORES];
07     int i;
08
09     sem_init(&semaforo, 0, 1);
       sem_init(&sem_lec, 0, 1);
10     for (i=0; i<MAX_LECTORES; I++)
11       pthread_create(&th_lector[i], NULL, Lector, NULL);
12     for (i=0; i<MAX_ESCRITORES; I++)
13       pthread_create(&th_escritor[i], NULL, Escritor, NULL);
14
15     for (i=0; i<MAX_LECTORES; i++)
16       pthread_join(th_lector[i], NULL);
17     for (i=0; i<MAX_ESCRITORES; i++)
18       pthread_join(th_escritor[i], NULL);
19
20     /*Cerrar todos los semáforos*/
21     sem_destroy(&semaforo);
22     sem_destroy(&sem_lec);
23     return 0;
24 }
25
26 void Lector(void) /* código del lector */
27 {
28     sem_wait(&sem_lec);
       n_lectores = n_lectores + 1;
       if (n_lectores == 1)
             sem_wait(&semaforo);
       sem_post(&sem_lec);

       printf("%d\n", dato); // Leer dato y mostrar el valor

       sem_wait(&sem_lec);
       n_lectores = n_lectores - 1;
       if (n_lectores == 0)
             sem_post(&semaforo); /* Libero el recurso */
       sem_post(&sem_lec);
       pthread_exit(0);
32 }
33
34 void Escritor(void) /* código escritor */
35 {
36     sem_wait(&semaforo);
37     dato = dato + 2; /* modificar recurso */
38     sem_post(&semaforo);
39
40     pthread_exit(0);
41 }
```