3. Indicar la salida por pantalla (2 puntos–20 minutos)

NOTA: **La clase string es un tipo especial de contenedor diseñado para manipular secuencias de caracteres.**

a)

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

class Pet
{
	string pname;
public:
	Pet(const string& petName) : pname(petName) {}
	virtual string name()  { return pname; }
	virtual string speak() { return ""; }
};

class Dog : public Pet
{
	string name;
public:
	Dog(const string& petName) : Pet(petName) {}
	virtual string sit()
	{
		return Pet::name() + " sits";
	}
	string speak()
	{
		return Pet::name() + " says 'Ladra!'";
	}
};

int main(void)
{
 Pet* p[] = {new Pet("A"),new Dog("B"),new Dog("C"),new Pet("D")};
 for (int i=0; i<4;i++)
		cout << p[i]->name()<< " " <<p[i]->speak() << endl;
 return 0;
}
```

Salida por pantalla

b) Explique brevemente el resultado de sustituir el bucle por el siguiente:

```
	for (int i=0; i<4;i++)
		if ((i%2)!=0)
			cout << p[i]->name()<< " " <<p[i]->sit() << endl;
```

Salida por pantalla

3.- Resolución

a)

Salida por pantalla
A
B B says ‘Ladra!’
C C says ‘Ladra!’
D

b)

Salida por pantalla
Error XXXX ‘sit’ no es un miembro de la clase ‘Pet’

El método ‘sit’ aparece declarado por primera vez en la clase Dog, por lo que no es accesible por punteros de la clase base aunque estos se encuentren realmente apuntando a objetos de la clase ‘Dog’. El error se produce en tiempo de compilación.

4. Ejercicio de programación (2,5 puntos-60 minutos)

Se desea realizar una pequeña base de datos que almacene los equipos y los resultados del mundial. Para ello se va generando el siguiente conjunto de clases que se describen y que deben ser declaradas y definidas por el alumno. Para las clases que a continuación se indican, se podrán agregar todos los métodos de interfaz, funciones y datos que se consideren oportunas, siempre y cuando tenga sentido y este de acuerdo con la POO.

Se pide:

1.- Terminar de **declarar y definir la clase base Persona**, la cual contiene el nombre y apellido de cualquier persona, los cuales pueden ser de longitud totalmente libre. Los metodos de interfaz pueden ser los que se requeiran, pero al menos la clase contiene:

```
class Persona{
      char *nombre;
      char *apellido;
    public:
      Persona();
      void setDatos(char *n, char *a);
      virtual void imprime();
      //métodos que se decidan como convenientes.
    };
```

2.- **Declarar y definir la clase Jugador**, que además de ser una persona, incuye un puntero a un objeto de tipo Equipo.

3.- **Declarar y definir la clase Arbitro**, que además de ser personas (en serio que lo son) contienen dos datos adicionales. Uno es el número de colegiado, y el otro es una enumeración que puede tomar los valores: ARBITRO, JUEZ_D_LINEA o CUARTO_ARBITRO.

4.- **Definir y declarar la clase Equipo**, la cual contiene dieciseis punteros a jugadores, y una cadena de caracteres que almacena el nombre del pais. Cuando se crea un equipo, este está vacío, y **es obligatorio indicar el pais**. Una vez construido, se van añadiendo jugadores por medio **del operador +=**, hasta alcanzar un máximo de 16. Cuando se añade un jugador, además de almacenarse su dirección, se actualiza el campo de equipo que dicho jugador tiene. Esta clase tendrá además un método **imprimrAlineacion**(), que mostrara por pantalla, el país, seguido por el listado de los jugadores.

5.- **Definir y declarar la clase Partido**, la cual contiene: las referencias (punteros) a dos equipos, uno local y otro visitante;un listado de cuatro arbitros; el resultado del partido; y una función Imprime() que permite visualizar toda esta información.

La puntuación del ejercicio irá no sólo en funcion del resultado, sino de la correcta programación usando las herramientas propias de la POO.Cada apartado vale 2 puntos.

La solución que se pone a continuación no es la única admisible, aunque es básicamente lo que habría que haber puesto, respetando fundamentalmente el encapsulamiento de las distintas clases, y dándoles la funcionalidad mínima que se ha pedido:

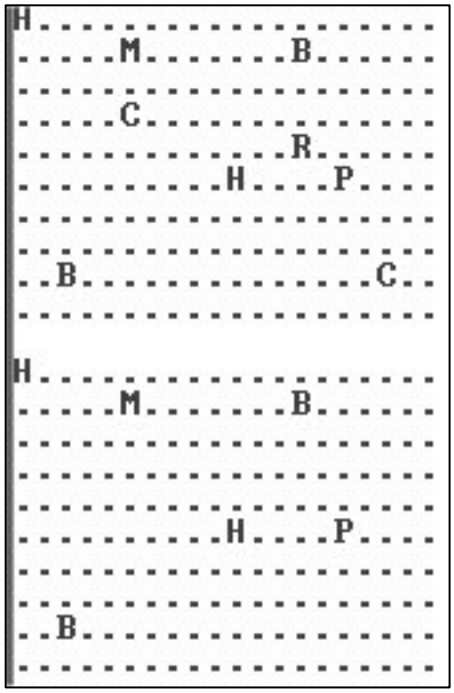
```
//declaración previa
#include <string.h>
#include <iostream>
using namespace std;
class Equipo;

class Persona{
char *nombre;
char *apellido;
public:
	Persona();
	void setDatos(char *n, char *a);
	virtual void imprime();
//añadidos por el alumno

	~Persona(){delete [] apellido; delete [] nombre;}

};
Persona::Persona(){
	apellido=nombre=0;
}
void Persona::setDatos(char *n,char *a){
delete [] nombre;
delete [] apellido;
nombre=new char[strlen(n)+1];
strcpy(nombre,n);
apellido=new char[strlen(a)+1];
strcpy(apellido,a);
}
void Persona::imprime()
{
cout<<nombre<<", "<<apellido;
}
//////////////FIN PRIMER APARTADO//////////////////
class Jugador:public Persona
{
Equipo *equipo;
public:
	void setEquipo(Equipo *eq){equipo=eq;}
	Jugador():equipo(0){}
};
//////////////FIN SEGUNDO APARTADO//////////////////

enum TipoArbitro{ARBITRO,JUEZ_D_LINEA, CUARTO_ARBITRO};
class Arbitro:public Persona
{
int numeroColegiado;
TipoArbitro tipo;
public:
	Arbitro(TipoArbitro tip):tipo(tip){numeroColegiado=0;}
	void setNumero(int num){numeroColegiado=num;}
	void imprime();
};
void Arbitro::imprime()
{
	switch(tipo){
	case ARBITRO:cout<<"ARBITRO: ";break;
	case JUEZ_D_LINEA:cout<<"JUEZ DE LINEA: ";break;
	case CUARTO_ARBITRO:cout<<"CUARTO ARBITRO: ";break;
	}
	Persona::imprime();
}
//////////////FIN TERCER APARTADO//////////////////
```

```
class Equipo
{
Jugador *jugadores[16];
int numJugadores;
char pais[100];
public:
	Equipo(char * _pais){
			numJugadores=0;
			strcpy(pais,_pais);
	}
	void operator+=(Jugador *in){
		if(numJugadores<16){
			jugadores[numJugadores++]=in;
			in->setEquipo(this);
		}
	}
	void imprimePais(){cout<<pais;}
	void imprimirAlineacion(){
		for(int i=0;i<numJugadores;i++){
			jugadores[i]->imprime();
			cout<<endl;
		}
	}
};
/////////////////FIN APARTADO 4//////////////////
class Partido{
Equipo *local;
Equipo *visitante;
Arbitro *arbitros[4];
int numArbitros;
int golesLocal;
int golesVisitante;
public:
	void setEquipos(Equipo *loc,Equipo *vis){
	 local=loc; visitante=vis;
	}
	void operator+=(Arbitro *in){
		if(numArbitros<4)
			arbitros[numArbitros++]=in;
	}
	void serResultado(int gloc,int gvis){
		golesLocal=gloc;
		golesVisitante=gvis;
	}
	void imprime();
};
void Partido::imprime(){
	local->imprimePais();
	cout<<" vs. ";
	visitante->imprimePais();
	cout<<": "<<golesLocal<<" - "<<golesVisitante<<endl;
	cout<<"alineacion Local"<<endl;
	local->imprimirAlineacion();
	cout<<endl<<"alineacion Visitante"<<endl;
	visitante->imprimirAlineacion();
	cout<<endl<<"Equipo Arbitral:"<<endl;
	for(int i=0;i<numArbitros;i++){
		arbitros[i]->imprime();
		cout<<endl;
		}
}
```

5. Problema de Análisis y Diseño Orientado a Objetos (2.5 puntos - 50 minutos)

Se desea crear un SIG (Sistema de Información Geográfica). Este sistema contendrá información sobre la localización de servicios en un **mapa**. Para ello se tendrán varias **capas** según el tipo de **servicio**. Habrá una capa de **hospitales** y **policía** llamada PUBLICO. Otra capa de **restaurantes** y **cafés** denominada HOSTELERIA y otra de TRANSPORTE con **autobuses** y **metros**. El programa permitirá mostrar el mapa con una o varias capas en pantalla. La prueba de test queda con el siguiente resultado:

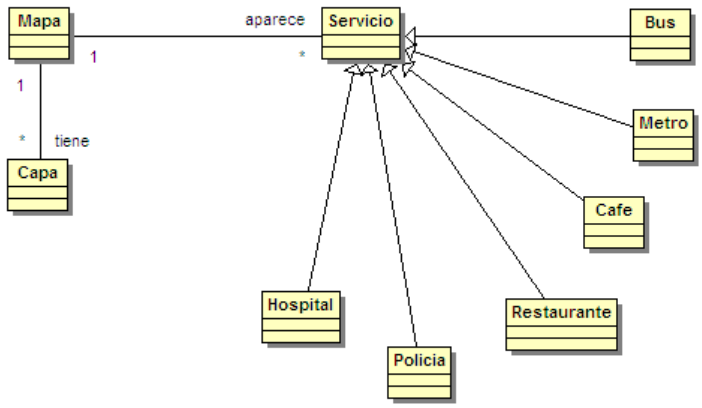
```
#include "Mapa.h"

int main()
{
      Mapa mapa(10,20);

      mapa.anyadirElemento(HOSPITAL, 0, 0);
      mapa.anyadirElemento(HOSPITAL, 5,10);
      mapa.anyadirElemento(POLICIA,  5,15);

      mapa.anyadirElemento(RESTAURANTE,4,13)
;
      mapa.anyadirElemento(CAFE, 3, 5);
      mapa.anyadirElemento(CAFE,8,17);


      mapa.anyadirElemento(BUS, 1,13);
      mapa.anyadirElemento(BUS, 8, 2);
      mapa.anyadirElemento(METRO,1,5);

      mapa.pintar();

      mapa.desactivarCapa(HOSTELERIA);

      mapa.pintar();

      return 0;
}
```



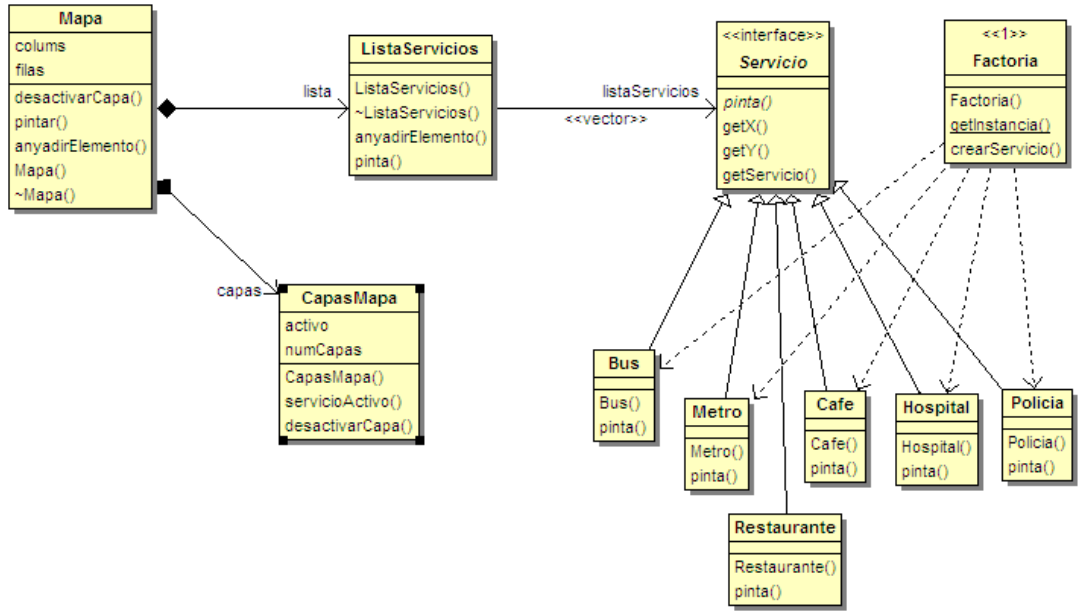
Se pide:

1. Modelo del dominio (1 punto).
2. Diagrama de Clases de Diseño indicando los patrones empleados (2.5 puntos).
3. Diagrama de secuencia del servicio **Mapa::pintar()** (2.5 puntos).
4. Implementación en C++ de la solución (4 puntos).

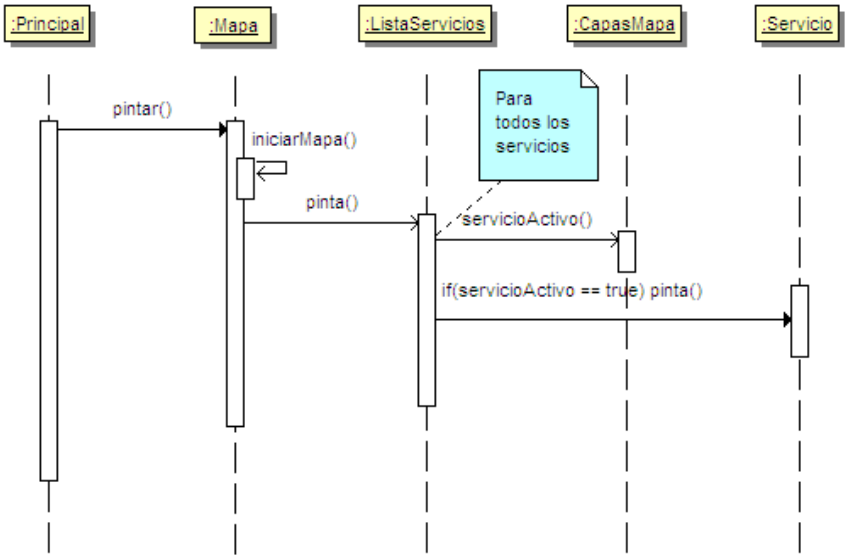
Resolución:
1.



2. Habrá una Factoría única para la creación de los objetos (Patrón Factoria y Singlenton). A los Servicios se les colocará un punto de Variaciones Protegidas (Polimorfismo, Indirección), ya que en el futuro se puede colocar nuevos servicios. Mapa tendrá una instancia de ListaServicios y de CapasMapa (Experto Información):



3.



4.

```
typedef enum{HOSPITAL, POLICIA, RESTAURANTE, CAFE, METRO, BUS} tipoServicio;

class Servicio
{
protected:
        tipoServicio elTipo;
        unsigned x,y;
public:
        virtual char pinta() = 0;
        unsigned getX() {return x;}
        unsigned getY() {return y;}
        tipoServicio getServicio() {return elTipo;}
};

class Hospital : public Servicio
{
        Hospital(unsigned ax,unsigned ay) { x=ax;y=ay; elTipo = HOSPITAL; }
        friend class Factoria;
public:
        char pinta() { return 'H'; }
};

class Policia : public Servicio
{
        Policia(unsigned ax,unsigned ay) { x=ax;y=ay; elTipo = POLICIA; }
        friend class Factoria;
public:
        char pinta() { return 'P'; }
};

class Restaurante : public Servicio
{
        Restaurante(unsigned ax,unsigned ay) { x=ax;y=ay; elTipo = RESTAURANTE; }
        friend class Factoria;
public:
        char pinta() { return 'R'; }
};

class Cafe : public Servicio
{
        Cafe(unsigned ax,unsigned ay) { x=ax;y=ay;elTipo = CAFE; }
        friend class Factoria;
public:
        char pinta() { return 'C'; }
};
```

```
class Factoria{
	Factoria() {}
public:
	static Factoria & getInstancia(){
		static Factoria factoria;
		return factoria;
	}
	Servicio * crearServicio(tipoServicio,unsigned,unsigned);
};
////////////////////////////////////////////////////
Servicio * Factoria::crearServicio(tipoServicio elTipo, unsigned x, unsigned y)
{
	if(elTipo == HOSPITAL) return new Hospital(x,y);
	else if(elTipo == POLICIA) return new Policia(x,y);
	else if(elTipo == RESTAURANTE) return new Restaurante(x,y);
	else if(elTipo == CAFE) return new Cafe(x,y);
	else if(elTipo == METRO) return new Metro(x,y);
	else if(elTipo == BUS) return new Bus(x,y);
	else return 0;

}
```

```
typedef enum{PUBLICO, HOSTELERIA, TRANSPORTE} tipoCapa;

class CapasMapa{
	bool *activo;
	unsigned numCapas;
public:
	CapasMapa(unsigned n)
	{ numCapas=n;activo = new bool[n];
		for(unsigned i=0;i<n;i++)
			activo[i]=true;
	}
	bool servicioActivo(Servicio *pServicio)
	{
	if(pServicio->getServicio() == HOSPITAL) return activo[PUBLICO];
	else if (pServicio->getServicio() == POLICIA) return activo[PUBLICO];
	else if (pServicio->getServicio() == CAFE) return activo[HOSTELERIA];
	else if (pServicio->getServicio() == RESTAURANTE) return activo[HOSTELERIA];
	else if (pServicio->getServicio() == METRO) return activo[TRANSPORTE];
	else if (pServicio->getServicio() == BUS) return activo[TRANSPORTE];
	else return false;
	}
	void desactivarCapa(tipoCapa elTipo) {activo[elTipo]=false;}
};


class ListaServicios
{
	vector<Servicio *> listaServicios;
public:
	ListaServicios();
	virtual ~ListaServicios();
	void anyadirElemento(tipoServicio elTipo,unsigned x, unsigned y)
	{	Factoria factoria = Factoria::getInstancia();
		listaServicios.push_back(factoria.crearServicio(elTipo,x,y)); }
	void pinta(char ***pImg,CapasMapa &capa)
	{
		char **pImagen = *pImg;
		for (unsigned i=0;i<listaServicios.size();i++)
		  if(capa.servicioActivo(listaServicios[i]))
			pImagen[listaServicios[i]->getX()][listaServicios[i]->getY()]
			=listaServicios[i]->pinta();
	}

};
```

```
class Mapa
{
	ListaServicios lista;
	CapasMapa capas;
	unsigned colums;
	unsigned filas;

public:
	void desactivarCapa(tipoCapa);
	void pintar();
	void anyadirElemento(tipoServicio,unsigned,unsigned);
	Mapa(unsigned f,unsigned c) : colums(c),filas(f),capas(3){}
	virtual ~Mapa();

};
////////////////////////////////////////////////
void Mapa::anyadirElemento(tipoServicio elTipo, unsigned int x, unsigned int y)
{
	lista.anyadirElemento(elTipo,x,y);

}

void Mapa::pintar()
{
	char **pImagen = new char *[filas];
	for(unsigned i=0;i<filas;i++)
		pImagen[i] = new char [colums];

	for(unsigned i=0;i<filas;i++)
		for(unsigned j=0;j<colums;j++)
			pImagen[i][j] = '.';

	lista.pinta(&pImagen,capas);


	for(unsigned i=0;i<filas;i++)
	{
		for(unsigned j=0;j<colums;j++)
			cout << pImagen[i][j];
		cout<<endl;
	}
	cout<<endl;

	for(unsigned i=0;i<filas;i++)
		 delete pImagen[i];

	delete pImagen;

}

void Mapa::desactivarCapa(tipoCapa elTipo)
{
	capas.desactivarCapa(elTipo);

}
```