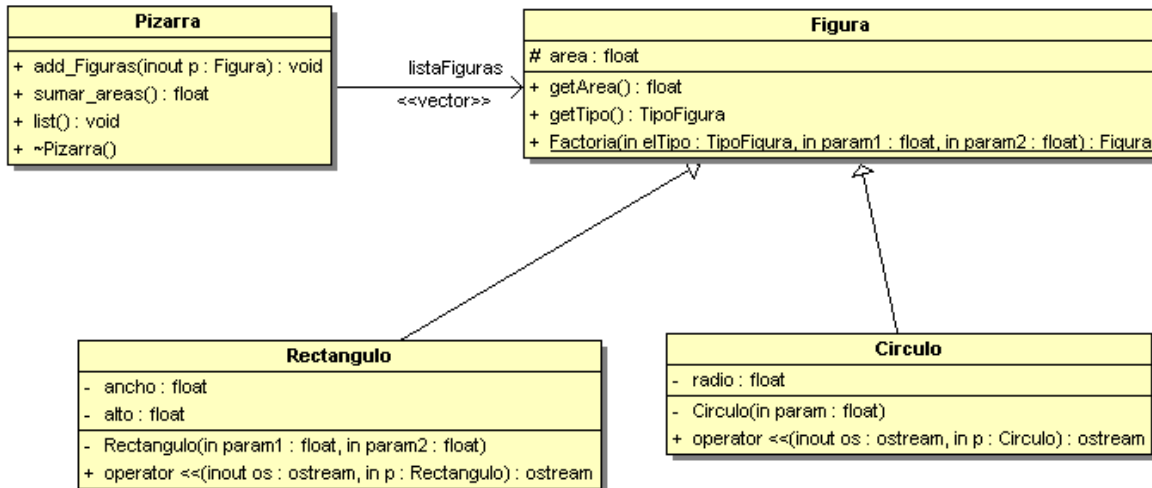


2. Problema de Análisis y Diseño Orientado a Objetos (10 puntos - 20 minutos)

Para el código de test adjunto, se pide.

1. Diagrama de clases de diseño (2.5 puntos).
2. Implementación en C++ de la solución (7.5 puntos).



```

// Código de test
int main()
{
    Pizarra pizarra;
    pizarra.add_Figuras(Figura::Factoria(CIRCULO,1));
    pizarra.add_Figuras(Figura::Factoria(RECTANGULO,1,2));

    cout << "Lista de figuras:" << endl;
    pizarra.list();
    cout<<"Suma total area: "<<pizarra.sumar_areas()<<endl;

    return 0;
}

//Resultado en consola
//Resultados:
Lista de figuras:
Circulo de radio 1
Rectangulo de ancho 1 y alto 2
Suma total area: 5.14152
    
```

```

#include <vector>
#include <iostream>
#include <algorithm>
using namespace std;
#define PI 3.141516f

typedef enum{CIRCULO, RECTANGULO} TipoFigura;
class Figura {
protected:
    TipoFigura elTipo;
    float area;
public:
    float getArea() {return area;}
    TipoFigura getTipo() {return elTipo;}
    static Figura *Factoria(TipoFigura, float, float);
};

class Circulo : public Figura
{
    friend class Figura;
    float radio;
    Circulo(float param):radio(param)
        {elTipo=CIRCULO;area=PI*radio*radio;}
public:
    friend ostream& operator<<(ostream& os, const Circulo * p) {
        return os << "Circulo de radio " << p->radio << endl;
    }
};

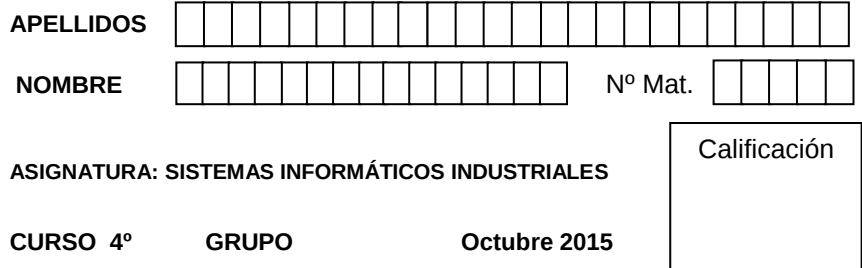
class Rectangulo : public Figura
{
    friend class Figura;
    float ancho, alto;
    Rectangulo(float param1, float param2):ancho(param1),alto(param2)
        {elTipo=RECTANGULO;area=ancho*alto;}
public:
    friend ostream& operator<<(ostream& os, const Rectangulo * p) {
        return os << "Rectangulo de ancho: " << p->ancho << " y alto: "
            << p->alto << endl;
    }
};

void visualizar(Figura *);
class Pizarra{
    vector<Figura *> listaFiguras;
public:
    void add_Figuras(Figura *p){
        listaFiguras.push_back(p);
    }
    float sumar_areas(){
        float area = 0;
        for(int i=0;i<listaFiguras.size();i++)
            area+=listaFiguras[i]->getArea();
        return (area);
    }
    void list(){
        for_each(listaFiguras.begin(),listaFiguras.end(),visualizar);
    }
    ~Pizarra(){
        for(int i=0;i<listaFiguras.size();i++)
            delete listaFiguras[i];
    }
};

Figura * Figura::Factoria(TipoFigura elTipo, float param1, float param2 = 0)
{
    if(elTipo == CIRCULO) return new Circulo(param1);
    else return new Rectangulo(param1,param2);
}

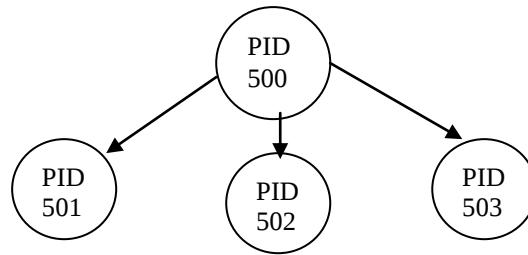
void visualizar(Figura *p)
{
    if(p->getTipo() == CIRCULO) cout << (Circulo *)p;
    else cout << (Rectangulo *)p;
}

```



Responda de forma breve y concisa las siguientes cuestiones teniendo en cuenta que el proceso padre P tiene el PID 500, que los procesos siguientes incrementan en una unidad el PID según se van creando y que durante la ejecución de P no se crean más procesos en el sistema:

1. (3 puntos) Dibuje el diagrama jerárquico de procesos creados indicando los PID de los procesos al invocar la ejecución del proceso desde una shell de la forma siguiente: **./P 4**



2. (2 puntos) Indique una posible traza de ejecución obtenida en la invocación anterior.

Se supone que el proceso bash tiene el PID 100.

```
Proceso padre: 100, P: 500
Padre: 500, Padre: 100
Hijo: 501, Padre: 500
Finalizado el hijo: 501
Finalizado el proceso P: 500
Hijo: 502, Padre: 100
Hijo: 503, Padre: 100
Finalizado el hijo: 502
Finalizado el hijo: 503
```

3. (2 puntos) Razone si en el caso anterior podría existir algún proceso huérfano y/o algún proceso zombie. En caso de que pudiera existir, escriba el código necesario para solucionarlo, indicando las líneas donde lo incluiría.

Sí que pueden existir procesos huérfanos ya que, como el padre no hace un wait por todos los hijos, si el padre finaliza antes que los hijos, éstos se quedan huérfanos y son heredados por el proceso Init.

También pueden existir procesos en estado zombie ya que pueden finalizar antes que el padre y desde que los hijos terminan hasta que el padre hace un wait por ellos se quedan en dicho estado.

Para solucionarlo, en la línea 35 se pondría este código para que el padre espere a que todos los hijos finalicen:

```
35     for (i = 1; i < n; i++) {
36         wait (&estado);
37     }
```

4. (3 puntos) Si se añade la siguiente línea de código en la primera ejecución del hijo (línea 24):

```
exec1p(argv[2], argv[2], NULL);
```

y se ejecuta el programa desde la shell mediante: **./P 4 date**, indique cómo cambiaría la jerarquía de procesos respecto a la del apartado 1 e indique una posible traza de ejecución.

NOTA: El mandato “date” imprime la hora del sistema.

 UNIVERSIDAD POLITÉCNICA DE MADRID ESCUELA TÉCNICA SUPERIOR DE INGENIERÍA Y DISEÑO INDUSTRIAL Departamento de Ingeniería Eléctrica, Electrónica, Automática y Física Aplicada	APELLIDOS	
	NOMBRE	Nº Mat.
	ASIGNATURA: SISTEMAS INFORMÁTICOS INDUSTRIALES	
	CURSO 4º	GRUPO
		Octubre 2015
		Calificación

La jerarquía de procesos no cambia respecto al apartado 1. Sin embargo, la traza de ejecución sí. Una posible traza es:

```
Proceso padre: 100, P: 500
Padre: 500, Padre: 100
<<Hora del sistema>>
Finalizado el proceso P: 500
<<Hora del sistema>>
<<Hora del sistema>>
```

5. (3 puntos) Escriba el código necesario para que tanto el proceso P como sus hijos ignoren la señal SIGINT (interrupción desde teclado mediante Control+C).

Hay que declarar la siguiente variable:

```
struct sigaction act;
```

Las siguientes líneas hay que ponerlas antes del bucle for para que tanto el proceso P como los hijos hereden el armado de la señal:

```
act.sa_handler=SIG_IGN;
act.sa_flags=0;
act.sa_mask=NULL;
sigaction(SIGINT, act, NULL);
```

6. (2 puntos) Si en la línea 35 se incluye la siguiente sentencia:

```
alarm(5);
```

Indique qué significado tiene y qué comportamiento se podría obtener en la ejecución de los procesos.

El proceso P genera la recepción de la señal SIGALRM pasados 5 segundos. Si transcurridos esos segundos los procesos no hubieran terminado, como no han armado esa señal, se ejecutaría la acción por defecto que es la de abortar proceso.