TP1 - Introduccion C++

Notas:

1.- La definición de la clase se debe realizar en un .h, mientras que la implementación de la misma se debe realizar en el punto cpp (o extensión similar), excepto se solicite lo contrario.

2.- Los ejercicios son evolutivos y dentro de esta serie, un ejercicio depende del ejercicio anterior.

3.- El código debe ser siempre el menor posible. Esto significa que, si una porción de código se repite, realizar una macro o encapsularlo en una función o método. Analizar si la codificación realizada es realmente necesaria.

4.- Siempre que sea posible, aunque el enunciado no lo exprese, utilizar el atributo **const**.

La clase Punto.

Dentro de cada punto, se acompaña el código fuente de la aplicación, y la correspondiente salida en pantalla.

1.- Definir e implementar una clase Punto, que posea los elementos públicos y privados:

Variables miembro privadas:

mx y **my** - del tipo **double**, que representan las coordenadas (x; y) del punto.

Métodos miembro públicos:

setPunto: recibe los valores de **x** e **y** en dos variables **double**.

getPunto: devuelve el valor del punto (en formato class Punto). En formato inline

setX y **setY**; que dan valor a x e y. En formato inline

getX y **getY**; que devuelven los valores de x e y. En formato inline

La creación del objeto debe permitir o no, las siguientes expresiones:

Punto pa; // **x** e **y** se inicializan en cero.

Punto pb (23.3,56.8);

Punto pc (34.4); // error de sintaxis.

Código Fuente:

```
using std::cout;
using std::endl;
int main(int argc, char *argv[])
{
        Punto p(3000.12,4.45);
        Punto r;
        //  Punto q(5.7); -- Error de sintaxis
        cout <<"1. punto p: ("<<p.getX()<<";"<<p.getY()<<")"<<endl;
        cout <<"2. punto r: ("<<r.getX()<<";"<<r.getY()<<")"<<endl;
```

```
        r.setX(-2000.22);
        r.setY(3.33);
        cout <<"3. punto r: ("<<r.getX()<<";"<<r.getY()<<")"<<endl;
        p.setPunto(9900.9,8800.8);
        cout <<"4. punto p: ("<<p.getX()<<";"<<p.getY()<<")"<<endl;
        r=p.getPunto();
        cout <<"5. punto r: ("<<r.getX()<<";"<<r.getY()<<")"<<endl;
        system("pause");
}
```

Salida:

```
1. punto p: (3000.12;4.45)
2. punto r: (0;0)
3. punto r: (-2000.22;3.33)
4. punto p: (9900.9;8800.8)
5. punto r: (9900.9;8800.8)
Presione una tecla para continuar . . .
```

2.- Modificar el punto anterior de verificar que los valores de x;y se encuentren dentro del rango -1000 a 1000, caso contrario su valor debe limitarse a estos valores.

De esta manera, la salida del código anterior sería:

```
1. punto p: (1000;4.45)
2. punto r: (0;0)
3. punto r: (-1000;3.33)
4. punto p: (1000;1000)
5. punto r: (1000;1000)
Presione una tecla para continuar . . .
```

Este control se debe realizar a través de miembros privados de la clase.

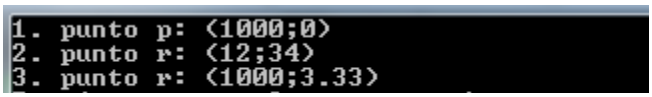
3.- Modificar el punto anterior de manera que acepte también el Método **set_punto (pa)**; en donde **pa** es un objeto Punto y acepte la creación del objeto de manera que la expresión **Punto pc (34.4);** sea aceptada asignando el valor pasado como parámetro a la coordenada **x** y asigne el valor cero a la coordenada y.

Implementar el constructor utilizando lista de inicialización (esto me permite inicializar miembros en forma implicita y a su vez asignar parametros por defecto) → (ejemplo : **FooClass::FooClass(int a=1, int b=2):x(a), y(b){}**)

Código Fuente:

```
int main(int argc, char *argv[])
{
        Punto p(1234.56);
        Punto r(12,34);
        cout <<"1. punto p: ("<<p.getX()<<";"<<p.getY()<<")"<<endl;
        cout <<"2. punto r: ("<<r.getX()<<";"<<r.getY()<<")"<<endl;
        p.setY(3.33);
        r.setPunto(p);
        cout <<"3. punto r: ("<<r.getX()<<";"<<r.getY()<<")"<<endl;
}
```

Salida:



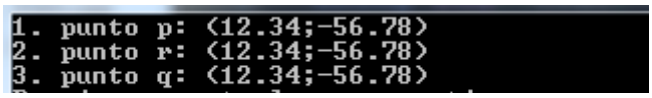
4.- Modificar el punto anterior de manera que acepte la creación de un objeto de la siguiente manera:

Punto pd (pc); o Punto pd=pc; // donde pc es un objeto Punto.

Código Fuente:

```
int main(int argc, char *argv[])
{
        Punto p(12.34,-56.78);
        Punto r(p);
        Punto q=p;
        cout <<"1. punto p: ("<<p.getX()<<";"<<p.getY()<<")"<<endl;
        cout <<"2. punto r: ("<<r.getX()<<";"<<r.getY()<<")"<<endl;
        cout <<"3. punto q: ("<<q.getX()<<";"<<q.getY()<<")"<<endl;
}
```

Salida:



Justificar el código de la clase resultante.

5.- Modificar el punto anterior para permitir la suma (+), resta (-) y asignación (=) de objetos tipo Punto.

Nota: tener en cuenta que las operaciones de deben controlar que los valores de x e y, no desborden el rango de +/- 1000.

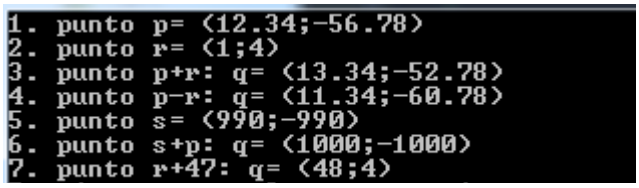
No utilizar **friend**, para realizar la sobrecarga de los operadores suma (+) y resta (-).

Código Fuente:

```
int main(int argc, char *argv[])
{
        Punto p(12.34,-56.78);
        Punto r(1,4);
        Punto q;
        cout <<"1. punto p= ("<<p.getX()<<";"<<p.getY()<<")"<<endl;
        cout <<"2. punto r= ("<<r.getX()<<";"<<r.getY()<<")"<<endl;
        q=p+r;
        cout <<"3. punto p+r: q= ("<<q.getX()<<";"<<q.getY()<<")"<<endl;
        q=p-r;
        cout <<"4. punto p-r: q= ("<<q.getX()<<";"<<q.getY()<<")"<<endl;
        Punto s(990,-990);
        cout <<"5. punto s= ("<<s.getX()<<";"<<s.getY()<<")"<<endl;
        q=s+p;
        cout <<"6. punto s+p: q= ("<<q.getX()<<";"<<q.getY()<<")"<<endl;
        q=r+47;
        cout <<"7. punto r+47: q= ("<<q.getX()<<";"<<q.getY()<<")"<<endl;
```

```
}
```

Salida:



Preguntas:

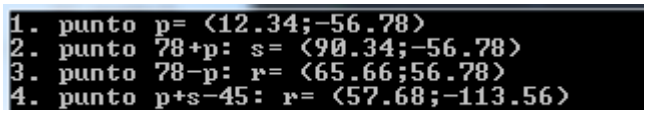
- ¿Requiere sobrecargar el operador = (asignación)? ¿Por qué?
- Justifique la ejecución de la línea 7. `7. punto r+47: q= (48;4)`

6.- Modificar el punto anterior para permitir la ejecución del siguiente código. Recordar realizar siempre la menor cantidad de código. En este caso, sería que la implementación de la suma y resta se realice cada una con un solo método miembro.

Justificar la implementación.

Código Fuente:

```
int main(int argc, char *argv[])
{
        Punto p(12.34,-56.78);
        Punto r,s;
        s=78+p;
        r=78-p;
        cout <<"1. punto p= ("<<p.getX()<<";"<<p.getY()<<")"<<endl;
        cout <<"2. punto 78+p: s= ("<<s.getX()<<";"<<s.getY()<<")"<<endl;
        cout <<"3. punto 78-p: r= ("<<r.getX()<<";"<<r.getY()<<")"<<endl;
        r=p+s-45;
        cout <<"4. punto p+s-45: r= ("<<r.getX()<<";"<<r.getY()<<")"<<endl;
}
```



7.- Modificar el punto anterior para permitir la comparación entre puntos: igualdad (==); desigualdad (!=), mayor (>) y menor (<) , sobrecargar el flujo de salida y entrada (<< y >>).

En el caso de mayor y menor, realizar la comparación a nivel módulos. Si lo desea, puede sobrecargar también >= y <=.

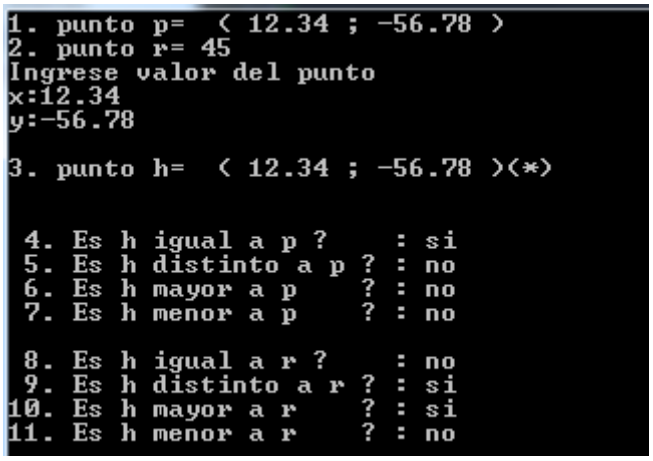
Código Fuente:

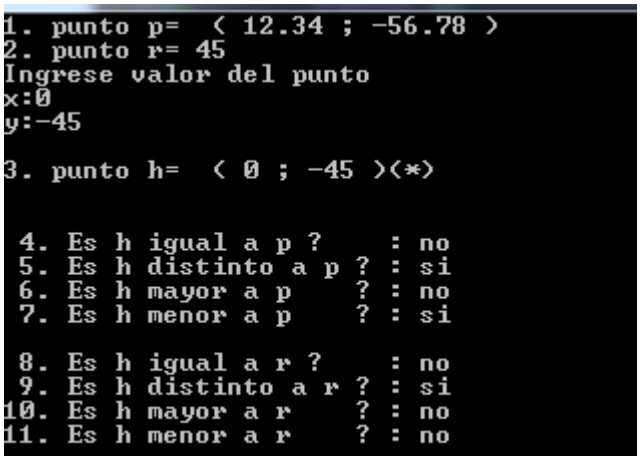
```
int main(int argc, char *argv[])
{
        Punto p(12.34,-56.78);
        double r=45;
```

```
        cout <<"1. punto p= "<<p<<endl;
        cout <<"2. punto r= "<<r<<endl;

        cout <<"Ingrese valor del punto"<<endl;
        Punto h;
        cin >>h;
        cout <<"3. punto h= "<<h<< "(*)" <<endl<<endl<<endl;

        cout <<" 4. Es h igual a p ?    : "<<((h==p)?"si":"no")<< endl;
        cout <<" 5. Es h distinto a p ? : "<<((h!=p)?"si":"no")<< endl;
        cout <<" 6. Es h mayor a p    ? : "<<((h>p)?"si":"no") << endl;
        cout <<" 7. Es h menor a p    ? : "<<((h<p)?"si":"no") << endl<<endl;

        cout <<" 8. Es h igual a r ?    : "<<((h==r)?"si":"no")<< endl;
        cout <<" 9. Es h distinto a r ? : "<<((h!=r)?"si":"no")<< endl;
        cout <<"10. Es h mayor a r    ? : "<<((h>r)?"si":"no") << endl;
        cout <<"11. Es h menor a r    ? : "<<((h<r)?"si":"no") << endl<<endl;
}
```

(*) La salida luego del punto 3, dependerá del valor ingresado. Se muestran dos posibles salidas.

Salidas:





8.- Agregar a la clase Punto, del ejercicio anterior, dos métodos públicos del tipo inline que indiquen la cantidad de instancias del tipo objeto que han sido creadas y cuantas se encuentran en memoria.

Nota: los contadores debe ser miembros privados de la clase.

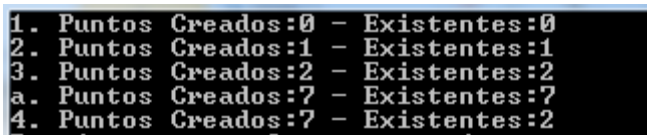
Justificar la invocación de los métodos **getCantCreada** y **getCantExistente** antes de la creación de cualquier objeto.

Código Fuente:

```
void ff (void)
{
        Punto p,q,w;
        Punto h(34);
        Punto r=h;
        cout <<"a. Puntos Creados:"<<Punto::getCantCreada()<< " - Existentes:"<< Punto::getCantExistente()<<endl;
}
```

```
int main(int argc, char *argv[])
{
        cout <<"1. Puntos Creados:"<<Punto::getCantCreada()<< " - Existentes:"<< Punto::getCantExistente()<<endl;
        Punto p(12.34,-56.78);
        cout <<"2. Puntos Creados:"<<Punto::getCantCreada()<< " - Existentes:"<< Punto::getCantExistente()<<endl;
        Punto h(p);
        cout <<"3. Puntos Creados:"<<Punto::getCantCreada()<< " - Existentes:"<< Punto::getCantExistente()<<endl;
        ff();
        cout <<"4. Puntos Creados:"<<Punto::getCantCreada()<< " - Existentes:"<< Punto::getCantExistente()<<endl;
}
```

Salida:



9.- Agregar un método público "**set_limite (float,float)**", que modifique el rango válido de los valores en x e y. Este método no debe modificar los valores de coordenadas x;y. (No importa que éstos queden fuera de rango). El primer parámetro corresponde al límite inferior y el segundo al superior. Si el límite inferior no es menor al superior, se debe omitir el cambio del rango.

Implementar también las funciones "**getLimiteSup**" y "**getLimiteInf**", del tipo **inline**, para saber cuáles son los valores de estos límites. Por omisión, los límites son +/- 1000.

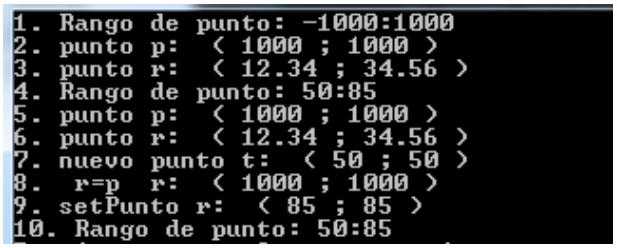
Si bien no se debe modificar los valores de x;y. Todos los objetos Punto (ya creados o por crear) se verán afectados por este nuevo rango.

Código Fuente:

```
int main(int argc, char *argv[])
{
        cout <<"1. Rango de punto: "<<Punto::getLimiteInf()<< ":"<<Punto::getLimiteSup()<<endl;

        Punto p(3000.12,5000);
        Punto r(12.34,34.56);
        cout <<"2. punto p: "<<p<<endl;
        cout <<"3. punto r: "<<r<<endl;

        Punto::setLimites(50,85);
        cout <<"4. Rango de punto: "<<p.getLimiteInf()<< ":"<<p.getLimiteSup()<<endl;
        cout <<"5. punto p: "<<p<<endl;
        cout <<"6. punto r: "<<r<<endl;

        Punto t;
        cout <<"7. nuevo punto t: "<<t<<endl;

        r=p;     // como la igualdad no esta redefinida, no se ve afectada por el nuevo límite
        cout <<"8.  r=p  r: "<<r<<endl;

        r.setPunto(p);
        cout <<"9. setPunto r: "<<r<<endl;

        r.setLimites(500,-85);
```

```
        cout <<"10. Rango de punto: "<<Punto::getLimiteInf()<< ":"<<Punto::getLimiteSup()<<endl;
}
```

Salida:

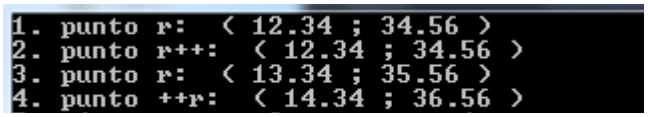


Pregunta: ¿por qué no se pude definir las funciones **getLimiteSup** y **gqtLimiteInf** como **const**?

10.- Sobrecargar el pre y post incremento de manera que incremente en una unidad tanto el valor de x como de y.

```
int main(int argc, char *argv[])
{
        Punto r(12.34,34.56);
        cout <<"1. punto r: "<<r<<endl;
        cout <<"2. punto r++: "<<r++<<endl;
        cout <<"3. punto r: "<<r<<endl;
        cout <<"4. punto ++r: "<<++r<<endl;
}
```

Salida:



Opcional: si desea puede realizar la sobrecarga del post y pre decremento.

Pregunta: que inconvenientes encuentra en una operación como la que sigue:

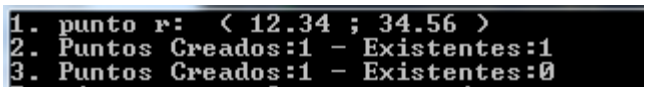
```
                Punto x(10,10);
                Punto y;

                        Y=x++ + x++;
```

11.- Modificar el punto anterior de manera que acepte los operadores new y delete.

Código Fuente:

```
int main(int argc, char *argv[])
{
        Punto *r=new Punto(12.34,34.56);
        cout <<"1. punto r: "<<*r<<endl;
        cout <<"2. Puntos Creados:"<<Punto::getCantCreada()<< " - Existentes:"<< Punto::getCantExistente()<<endl;
        delete (r);
        cout <<"3. Puntos Creados:"<<Punto::getCantCreada()<< " - Existentes:"<< Punto::getCantExistente()<<endl;
}
```

Salida:



Justificar el código de la clase resultante.