Clase 19. Introducción a la programación orientada a Objetos (continuación)

Clases

Funciones miembros de la clase

Una función miembro es una función que tiene su definición o prototipo dentro de la definición de la clase.

```
1  class Circulo
   {
3     public:
        double radio;
5       double area();
   };
7  double Circulo::area(){
      return (3.14159*radio*
      radio/2.0);
9  }
```

```
1  class Circulo
   {
3     public:
        double radio;
5       double area(){
          return (3.14159*
       radio*radio/2.0);
7       }
   }
```

Para definir a la función fuera de la clase se utiliza el **operador de resolución de alcance, ::**.

Modificadores de acceso

El escondimiento de datos **(data hidding)** se realiza por medio de etiquetas denominadas **modificadores de acceso**.

```
class NombreClase
{
  public:
    //Miembros publicos
  protected:
   //Miembros protegidos

   private:
    //Miembros privados
};
```

Modificadores de acceso:public

public: Los miembros públicos son accessibles desde cualquier lado fuera y dentro de la clase en el mismo programa.

```
class Circulo
{
  public:
      double radio;
  protected:
   //Miembros protegidos

   private:
    //Miembros privados
};
```

```
int main()
{
 Circulo C;
 //Se puede acceder a
    radio por ser publico
 C.radio=5.9;
return 0;
}
```

Modificadores de acceso:private

Nota cultural: accessar no existe como palabra, se traduce normalmente como acceder. Las usaré indistintamente.

private: Una variable o función privada NO puede ser accedida por otra función o clase, solo por la misma función y clases o funciones **friend**.


```
1  class Circulo
   {
3     public:
         void setRadio(double
        r){radio=r;}
5        double getRadio(){
       return radio;}
      private:
7        double radio;
      protected:
9  };
```

```
1  int main()
   {
3   Circulo C;
    //NO se puede acceder a
       radio por ser privado
5   //C.radio=5.9;
   C.setRadio(5.9); //Pero
       setRadio si puede
7   cout << "Radio: " << C.
       getRadio() << endl;
   return 0;
9  }
```

```
Radio: 5.9
```

Modificadores de acceso:protected/privado

protected: Es similar a **private**, pero además permite acceder a las clases derivadas.**No se puede lo sig.**


```
1  #include <iostream>
   class Circulo
3  {
    public:
    private:
5    double radio;
     void setRadio(double r){
7      radio=r;}
     double getRadio(){return
       radio;}
9   protected:
      double diametro;
11 };
```

```
In member function
'void Circ::setCRadio(double)':
error: 'double Circulo::radio'
is private
error: within this context
```

```
1  class Circ:Circulo
   {
3   public:
     void setCRadio(double r)
       {
5     //No puede acceder a
        radio por ser privado
       radio=r;
7   }
    private:
9   protected:
   };
11 int main()
   {
13  Circ C;
    //NO puede acceder a
       radio por ser privado
15  C.setCRadio(5.9);
   return 0;
17 }
```

Modificadores de acceso:protected/privado

protected: Es similar a **private**, pero además permite acceder a las clases derivadas. **No se puede lo sig.**


```
1  #include <iostream>
   class Circulo
3  {
    public:
    private:
5    double radio;
     void setRadio(double r){
7      radio=r;}
     double getRadio(){return
        radio;}
9   protected:
      double diametro;
11 };
```

```
In member function
'void Circ::setCRadio(double)':
error: 'double Circulo::radio'
is private
error: within this context
```

```
1  class Circ:Circulo
   {
3   public:
     void setCRadio(double r)
       {
5     //No puede acceder a
       setRadio por ser
       privado
       setRadio(r);
7   }
    private:
9   protected:
   };
11 int main()
   {
13  Circ C;
    //NO puede acceder a
       setRadio por ser
       privado
15  C.setCRadio(5.9);
   return 0;
```

Modificadores de acceso:protected/privado

protected: Es similar a **private**, pero además permite acceder a las clases derivadas.


```
1  #include <iostream>
   class Circulo{
3   private:
      double radio;
5   protected:
      void setRadio(double r){radio=r;}
7     double getRadio(){return radio;}
   };
9  class Circ:Circulo{
    public:
11    void setCRadio(double r){setRadio(r);}
      double getCRadio(){return getRadio();}
13 };
   int main(){
15  Circ C;
    C.setCRadio(5.9);
17  std::cout << "Radio: "<< C.getCRadio() << std::endl;
   return 0;
19 }
```

Tarea 1

Tarea 10

Nota: No es necesario conservar el nombre de las funciones, clases o datos, solo la funcionalidad. Todas las funciones de C son validas en C++ (no hemos visto lectura/escritura de archivos en C++).

prog10.2 Programar una clase base **SistemaCompleto** sin datos ni métodos publicos. Y una clase derivada **Sistema**. La clase base:

- Todos los datos que se utilicen se declaran en la clase (no declarar datos locales a las funciones).
- Tiene una función **lee_archivo** que lee un sistema de ecuaciones lineales de $n \times n$ (en el formato que quieran).
- Tiene una función **soluciona** que soluciona el sistema de ecuaciones con un método para matrices llenas no simétricas (el que quieran).
- Tiene una función **ImprimeArchivo** que imprime la solución al archivo de salida (en el formato que quieran).

- La clase derivada además tiene una función que verifica si la matríz es simétrica o triangular superior o inferior, y tienen métodos que resuelve el sistema para estos casos partículares.
- En el main, hacen una instancia de la clase derivada y llaman al método **resuelveSistema("entrada.in","salida.out")** que será lo único publico de la clase,y resolverá el sistema. Pueden hacer uso de todas las funciones auxiliares que quieran y declararlas privadas o protegidas según les resulte mas conveniente.

Constructores

El constructor de la clase

- Un **constructor** de la clase, es una funció miembro de la clase que se ejecuta cuando instanciamos un objeto de la misma.

El constructor de la clase

- Un **constructor** de la clase, es una funció miembro de la clase que se ejecuta cuando instanciamos un objeto de la misma.
- El constructor tiene el mismo nombre de la clase y no tiene ningún tipo de retorno ni siquiera **void**.

El constructor de la clase

- Un **constructor** de la clase, es una funció miembro de la clase que se ejecuta cuando instanciamos un objeto de la misma.
- El constructor tiene el mismo nombre de la clase y no tiene ningún tipo de retorno ni siquiera **void**.
- El **constructor** puede usarse para inicializar variables.

El constructor de la clase

- Un **constructor** de la clase, es una funció miembro de la clase que se ejecuta cuando instanciamos un objeto de la misma.
- El constructor tiene el mismo nombre de la clase y no tiene ningún tipo de retorno ni siquiera **void**.
- El **constructor** puede usarse para inicializar variables.
- Si no se declara un constructor, el compilador declara uno siempre **public**, e **inline**.

El constructor de la clase

- Un **constructor** de la clase, es una funció miembro de la clase que se ejecuta cuando instanciamos un objeto de la misma.
- El constructor tiene el mismo nombre de la clase y no tiene ningún tipo de retorno ni siquiera **void**.
- El **constructor** puede usarse para inicializar variables.
- Si no se declara un constructor, el compilador declara uno siempre **public**, e **inline**.
- También se declara un constructor de copia por default.

El constructor de la clase

- Un **constructor** de la clase, es una funció miembro de la clase que se ejecuta cuando instanciamos un objeto de la misma.
- El constructor tiene el mismo nombre de la clase y no tiene ningún tipo de retorno ni siquiera **void**.
- El **constructor** puede usarse para inicializar variables.
- Si no se declara un constructor, el compilador declara uno siempre **public**, e **inline**.
- También se declara un constructor de copia por default.
- Si el programador define un constructor default (sin parámetros) o copia (recibe referencia a un objeto del mismo tipo), estos no se declaran.

El constructor de la clase, ejemplo

Definición dentro de la clase.

```
1  #include <iostream>
class Circulo
3  {
 public:
5    Circulo(){
       cout<< "Construyendo..."<<endl;
7    }
};
9  int main()
{
11  Circulo C;
return 0;
13 }
```

```
Construyendo...
```

El constructor de la clase, ejemplo

Definición fuera de la clase.

```
1  #include <iostream>
class Circulo
3  {
 public:
5    Circulo();
};
7  Circulo::Circulo(){
       std::cout<< "Construyendo..."<< std::endl;
9    }

11 int main()
{
13  Circulo C;
return 0;
15 }
```

```
Construyendo...
```

El constructor de la clase, ejemplo

Inicializar en el constructor.


```
1  #include <iostream>
   class Circulo{
3     double radio;//radio es private
    public:
5     Circulo();
      void setRadio(double r){radio=r;};
7     double getRadio(){return radio;};
   };
9  Circulo::Circulo(){
        radio=5;
11      std::cout<< "Construyendo..."<<std::endl;
   }
13 int main(){
    Circulo C;
15  std::cout << "Radio= " << C.getRadio() << std::endl;
   return 0;
17 }
```

```
Construyendo...
Radio= 5
```

El constructor de la clase, ejemplo

Constructor privado.


```
1  #include <iostream>
   class Circulo{
3     double radio;
      Circulo(); //Circulo es private
5   public:
      void setRadio(double r){radio=r;};
7     double getRadio(){return radio;};
   };
9  Circulo::Circulo(){
        radio=5;
        std::cout<< "Construyendo..."<<std::endl;
   }
13 int main(){
    Circulo C;
15 return 0;
   }
```

```
In function ‘int main()’:
error: ‘Circulo::Circulo()’ is private
error: within this context
```

El constructor de la clase, ejemplo

Constructor con parámetros


```
#include <iostream>
class Circulo{
   double radio;
 public:
   Circulo(double);
   void setRadio(double r){radio=r;};
   double getRadio(){return radio;};
};
Circulo::Circulo(double r){
     radio=r;
}
int main(){
 Circulo C(4.0);
 std::cout<< "Radio= "<< C.getRadio() <<std::endl;
return 0;
}
```

```
Radio= 4
```

El constructor de la clase, ejemplo

Una vez declarado un único constructor con parámetros, ya se eliminó el constructor por default. (ver constructor de copia)


```
class Circulo{
   double radio;
 public:
   Circulo(double);
};
Circulo::Circulo(double r){
     radio=r;
}
int main(){
 Circulo C;
return 0;
}
```

```
In function ‘int main()’:
error: no matching function for call to ‘Circulo
note: candidates are:
Circulo::Circulo(double)
candidate expects 1 argument, 0 provided
Circulo::Circulo(const Circulo&)
candidate expects 1 argument, 0 provided
```

Inicializaciones

Inicialización directa

La inicialización directa ocurre cuando se llama al constructor con argumentos y esos argumentos sirven para inicializar el objeto. sin embargo también puede ser utilizada para tipos primitivos:

```
#include <iostream>
class Circulo{
   double radio;
 public:
   Circulo(){std::cout<< "Construye sin parametros"<<
    std::endl; }
   Circulo(double r){radio=r;}
   void setRadio(double r){radio=r;};
   double getRadio(){return radio;};
};
int main(){
  int x(3);
  double y(9.845);
  Circulo C(4);
  std::cout << "x= " << x << " y= " << y<<std::endl;
return 0;
}
```

Inicialización con valor (Value initialization)

C++ (since C++11), proveé un tipo de inicialización por defecto (sin parámetros a 0). Referencia:

`http://en.cppreference.com/w/cpp/language/value_init`


```
T();
new T ();
Class::Class(...) : member() {...
T object {};
T{};
new T {};
Class::Class(...) :member{} {...

```

```
int = 0,
bool = false,
float = 0.0f,
enum = (enum type)0,
pointer = null pointer
pointer to member = null member pointer

```

Value initialization, ejemplo.


```
T();
new T ();
Class::Class(...) : member() {...
T object {};
T{};
new T {};
Class::Class(...) :member{} {...

```

Sobre new

La memoria con **new** no está inicializada por default, pero se puede inicializar:


```
1  #include <iostream>
   void f(){
3    int *x=new int[100];
     for (int i=0;x[i]=i+1, i<100; i++);
5    delete []x;
   }
7  int main(){
     int *z=new int[4]();
9    for (int i=0; i<4; i++) std::cout << z[i] <<" ";
     std::cout <<std::endl;
11   f();//Intento forzar al heap a tener algo
     int *w=new int[4];
13   for (int i=0; i<4; i++) std::cout << w[i] <<" ";
     std::cout <<std::endl;
15   delete []w;  delete []z;
   return 0;
17 }
```

```
0 0 0 0
1 2 3 4
```

Sobrecarga del constructor

Sobrecarga del constructor, ejemplo

La sobrecarga del constructor da flexibilidad de inicializar el objeto de acuerdo a los parametros del constructor.


```
1  #include <iostream>
   class Circulo
3  {
      double radio;
5   public:
      Circulo(){std::cout<< "Construye sin parametros"<<
       std::endl; }
7     Circulo(double r){radio=r;}
      void setRadio(double r){radio=r;};
9     double getRadio(){return radio;};
   };
11 int main()
   {
13  Circulo C, B(8.0);
   return 0;
15 }
```

```
Construye sin parametros
```

Constructor de copia

El constructor de copia está definido por default, se elimina si la clase define uno específicamente. Solo se manda llamar si:

- Se crea la instancia a partir de otro objeto como parámetro.
- Se asigna como inicialización otro objeto al momento de la instancia.
- NO se llama en la asignación en cualquier otro momento que no sea la instancia.

```
1 #include <iostream>
  using namespace std;
3 class Circulo{
   public:
5    double radio;
     Circulo(){cout<< "Cons. noparam"<<endl; }
7    Circulo(double r){radio=r;}
     Circulo(Circulo &X){cout<< "Construye copia"<<endl;}
9 };
```

Constructor de copia


```
#include <iostream>
using namespace std;
class Circulo{
 public:
   double radio;
   Circulo(){cout<< "Cons. noparam"<<endl; }
   Circulo(double r){radio=r;}
   Circulo(Circulo &X){cout<< "Construye copia"<<endl;}
};
int main(){
 Circulo B(8.0),D(B);
 Circulo C=B;
 Circulo E;
 E=B;
 cout<< "radios: B="<< B.radio << " C= "<< C.radio << \
 " D= " << D.radio << " E= "<< E.radio << endl;
return 0;
}
```

```
Construye copia
Construye copia
Cons. noparam
radios: B=8 C= 6.95335e-310 D= 9.88131e-324 E= 8
```

Tarea 2

Tarea 10

prog10.3 Repita la clase base de la tarea 10.2, pero:

- Defina variables públicas que sean nombres de archivos de entrada y de salida.
- Defina un constructor por default que inicializa los nombres con caracteres nulos.
- Sobrecargue el constructor con uno que reciba los nombres de entrada y de salida, resuelva el sistema de EL, e imprima la solución (puede usar la misma función de la tarea 10.2 llamada desde el constructor).
- Sobrecargue el constructor con uno de copia que, en caso de que los nombres de los archivos no sean nombre de caracteres nulos, resuelvan (o intenten resolver) el sistema, pero al nombre del archivo de salida le agregue la palabra **copia** al final.
- Ejemplifique el uso de la clase en el main.