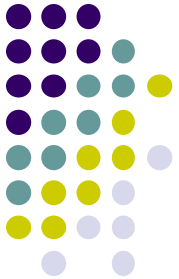
ALGORITMOS Y ESTRUCTURAS DE DATOS 1

Práctica: CUACKER

Sesión 2

Introducción a C++

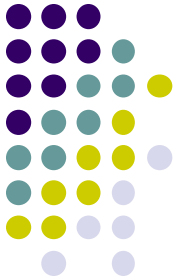


Videotutorial 6

- Clases y objetos:
  - Declaración de las clases.
  - Constructores y destructores.
  - Implementación de los métodos.

- Espacios de nombres y el operador ::

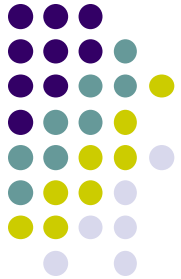
Clases y objetos



¡Que levante la mano quien no esté en POO!

- Las **clases** son un mecanismo para definir tipos abstractos de datos.
  - Una clase es un **tipo de datos**, que define los datos necesarios para representarlo y las operaciones sobre el mismo.
  - Una clase es un **módulo**, que agrupa funcionalidad y ofrece ocultación de la implementación.
- Ejemplo: clase persona, clase lista, clase conjunto, clase árbol, etc.

Clases y objetos

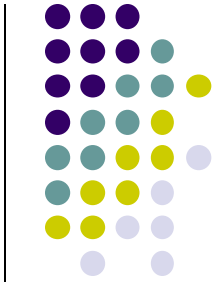


- **Clase**: define un nuevo tipo de datos.
- **Objeto**: una variable de una clase.
- **Método**: una operación de una clase.
- **Atributo**: un dato de una clase.

Miembros

```
class Persona
{
    string nombre;
    long dni, telefono;
    void leer (void);
    void escribir (void);
};
Persona p1;
p1.leer();
p1.escribir();
```

Clases y objetos



- **Declaración de una clase:**

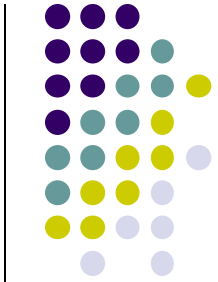
```
class Nombre
{
   ...
};
```

- **Miembros públicos y privados**:
  - **Públicos**: accesibles por los usuarios de la clase.
  - **Privados**: solo accesibles dentro de la propia clase.

```
class Nombre {
  private:
     ...        // Miembros privados
  public:
     ...        // Miembros públicos
};
```

Observar que public y private no se ponen igual que en Java.

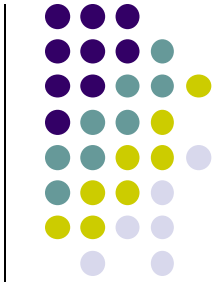
Clases y objetos



- *Normalmente*:
  - Los atributos de la clase son privados y van al principio en la declaración de la clase.
  - Los métodos de la clase son públicos, aunque si hay operaciones solo de uso interno, serán privadas.

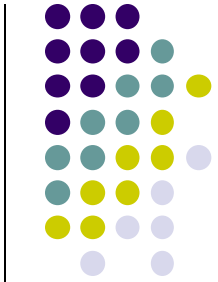
```
class Persona
{
   private:
      string nombre;
      long dni, telefono;
   public:
      void leer (void);
      void escribir (void);
};
```

Constructores y destructores



- Dentro de una clase pueden definirse dos métodos especiales:
  - **Constructor(es)**: operación que se invoca cuando se crea un nuevo objeto de la clase.
  - **Destructor**: operación que se invoca cuando se elimina un objeto de la clase.
- Según el tipo de objeto:
  - **Variables globales**: se crean al inicio del programa y se eliminan al terminar el programa.
  - **Variables locales**: se crean al llamar a la función y se eliminan al acabarla.
  - **Variables dinámicas**: se crean con el **new** y se eliminan con el **delete**.

Constructores y destructores



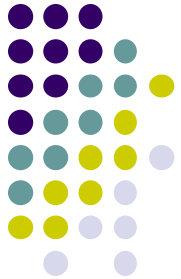
- **Constructor(es)**:
  - Su función es inicializar los atributos del objeto a un estado consistente.
  - Puede haber varios, con distintos parámetros.
  - Deben tener el mismo nombre de la clase y no devuelven nada:

```
class Persona {
    ...
    Persona();
    Persona(string nombre);
};
```

```
Persona p1;
Persona *p2= new Persona;
Persona p3("Perico");
Persona *p4= new
        Persona("Manolo");
```

  - Si existe un constructor sin parámetros, se denomina **constructor por defecto**. Se llamará siempre automáticamente al crear el objeto.

Constructores y destructores



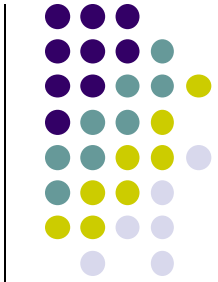
- **Destructor**:
  - Su función es liberar la memoria dinámica que haya reservado el objeto, cuando éste se elimina. No es necesario si la clase no usa memoria dinámica.
  - Solo puede haber uno.
  - Debe tener el nombre de la clase precedido de ~:

```
class Persona {
    ...
    Persona();
    ~Persona();
};
```

```
Persona p1;
Persona *p2= new Persona;
...
delete p2;
```

  - El destructor **nunca debe llamarse de forma explícita**, sino que se llama automáticamente con delete (o, si es estático, cuando desaparezca).

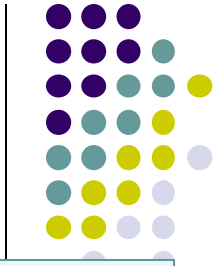
Implementación de los métodos



- La implementación de los métodos debe ir **fuera** de la definición de la clase.
- El nombre del método *fuera* de la clase es: clase::método.

```
class Persona
{
   ...
   void escribir (void);
};
...
void Persona::escribir (void) {
   cout << "Nombre: " << nombre << ...
}
```

Implementación de los métodos



- La implementación también puede ir dentro de la clase: **métodos inline**.

```
class Persona
{
   ...
   void escribir (void) {
      cout << "Nombre: " << nombre << ...
   }
};
```

Ojo, el significado es distinto de Java. En C++ lo normal es implementar los métodos fuera de la clase.

- **Significado**: al hacer la llamada, se *pega* el código en todos los sitios donde se llame.
- Más eficiencia, pero se duplica el código objeto.
- Utilizarlo solo con los métodos triviales.
- Se puede usar la palabra clave **inline** para declararlas explícitamente: `inline int maximo (int a, int b) {...}`

Espacios de nombres

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

- ¿Qué es eso del **using namespace …**?
- Espacios de nombres: agrupación de variables, procedimientos y tipos de datos bajo un nombre común.
- Es útil en proyectos muy grandes, con muchos programadores, módulos, etc.
- Evita colisiones de un mismo nombre en distintos bloques.

Espacios de nombres

- Ejemplo, programa de gestión de una gran empresa.

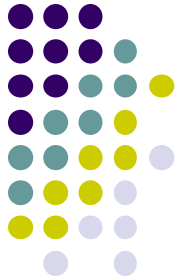
interfaz.cpp

```
// Módulo de Interfaz //
// -------------------//
namespace INTERFAZ {
  int errorCode= 7;
  void initialize() {
     ...
  }
  bool initGUI(string window){
     ...
  }
}
```

gestion.cpp

```
// Módulo de Gestión //
// ------------------//
namespace GESTION {
  int errorCode= 0;
  void initialize() {
     ...
  }
  bool connectDB(string name){
     ...
  }
}
```

Espacios de nombres



- ¿Cómo se accede a las variables y procedimientos dentro del espacio de nombres?
- Dos opciones:
  - Añadiendo como prefijo el nombre del espacio:

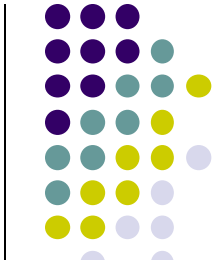
```
GESTION::initialize();
if (GESTION::errorCode>0) ...
```

  - Usando el using namespace:

```
using namespace GESTION;
initialize();
if (errorCode>0) ...
```

Por eso para las librerías estándar usamos:
using namespace std;

El operador de resolución de visibilidad

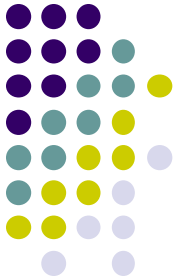


- En general, el operador `::` se usa para identificar una propiedad dentro de un ámbito.

`Espacio::variable`
`Clase::metodo`
`::variable_global`

- Ejemplo, resolver ambigüedad de nombres:

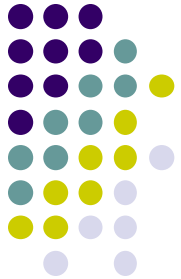
```
double a= 1.0;
int main ()
{
   int a= 4;
   a= 7;       // Acceso a la variable local
   ::a= 3.0;   // Acceso a la variable global
}
```



Semana 2: ejercicios 003 y 004

Planificación práctica

003 – Leyendo la fecha y la hora

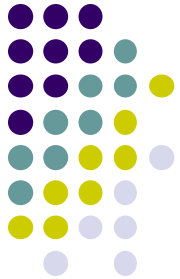


- Añadir a nuestro programa la funcionalidad de procesamiento de fechas y horas.
- Formato: `día/mes/año hora:minuto:segundo`

`26/6/2003 07:05:15  28/10/2018 10:00:01`

- Cuestiones a tratar:
  - Definir la **clase Fecha**.
  - Implementar un método para leer fechas.
  - Implementar un método para escribir fechas.
  - Implementar un método para comparar dos fechas.

003 – Leyendo la fecha y la hora



- Ejemplo de clase Fecha:

```
class Fecha {
  private:
     int dia, mes, ano;
     int hora, minuto, segundo;
  public:
     Fecha();
     bool leer();
     void escribir();
     bool es_menor(Fecha &otra);
     bool es_igual(Fecha &otra);
};
```

Todos los atributos son estáticos → No hace falta destructor.

Constructor, para inicializar a 0 los atributos.

Devuelve bool para saber si se ha leído bien.

¿Por qué por referencia? Para evitar que haya una copia en la llamada.

003 – Leyendo la fecha y la hora

programa.cpp

```
#include <iostream>
using namespace std;
const int NUM_PCUACS= 30;
...
string pcuac_a_texto (int num)
{   return cadenas_pcuac[num-1];}
```

Código ejercicios anteriores

```
class Fecha {
    private:
          int dia, mes, ano;
          int hora, minuto, segundo;
    public:
          Fecha();
          bool leer();
          void escribir();
          bool es_menor(Fecha &otra);
          bool es_igual(Fecha &otra);
};
```

Definición de la clase Fecha

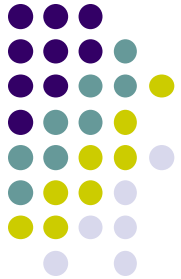
```
Fecha::Fecha ()
{ ... }
bool Fecha::leer()
{ ... }
void Fecha::escribir()
{ ... }
bool Fecha::es_menor(Fecha &otra);
{ ... }
bool Fecha::es_igual(Fecha &otra);
{ ... }
```

Implementación de los métodos de la clase Fecha

```
int main (void) {
    int ncasos;
    Fecha factual, fanterior;
    cin >> ncasos;
    for (...) { ... }
}
```

Procedimiento main

003 – Leyendo la fecha y la hora



Entrada

```
6
6/8/2002 09:32:22
6/8/2001 12:31:55
26/6/2003 13:15:01
26/6/2003 07:05:15
26/6/2003 07:05:15
23/3/2004 15:47:34
```

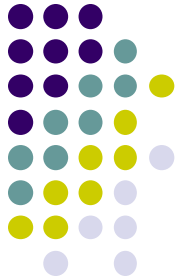
Salida

```
6/8/2001 12:31:55 ES ANTERIOR A 6/8/2002 09:32:22
26/6/2003 13:15:01 ES POSTERIOR A 6/8/2001 12:31:55
26/6/2003 07:05:15 ES ANTERIOR A 26/6/2003 13:15:01
26/6/2003 07:05:15 ES IGUAL A 26/6/2003 07:05:15
23/3/2004 15:47:34 ES POSTERIOR A 26/6/2003 07:05:15
```

- Procedimiento main: Leer la primera fecha. Para cada caso: leer fecha; comparar con la anterior; escribir mensaje según la comparación.
- Recordar:

```
g++ programa.cpp
./a.out < 003a.in > salida
diff 003a.out salida
```

004 – El tipo de datos Cuac



- Crear una **clase Cuac**, que almacena los mensajes de nuestro sistema.
- Formato:

```
mcuac NOMBRE_USUARIO
FECHA_HORA
MENSAJE_DE_TEXTO
```

```
pcuac NOMBRE_USUARIO
FECHA_HORA
NUMERO
```

- Cuestiones a tratar:
  - Definir la **clase Cuac**. ¿Cómo almacenar los pcuac?
  - Implementar un método para leer cuacs.
  - Implementar un método para escribir cuacs.
  - Implementar un método para comparar dos cuacs.

004 – El tipo de datos Cuac

- Ejemplo de la clase Cuac.

```
class Cuac {
  private:
    Fecha fecha;
    string usuario;
    string texto;
  public:
    bool leer_mcuac();
    bool leer_pcuac();
    void escribir();
    bool es_anterior(Cuac &otro);
};
```

También todos los atributos son estáticos. Cada vez que se cree un objeto Cuac, se llama a sus constructores.

El main lee el tipo de comando (mcuac o pcuac) y llama al método adecuado.

004 – El tipo de datos Cuac

Entrada

```
mcuac RafaelNaval
25/10/2011 13:45:11
¡Feliz Navidad!
pcuac RafaelNaval
28/11/2011 11:27:08
5
mcuac GinesGM
6/5/2012 16:00:00
Dicen en #eltiempo que este...
pcuac Gutierrez
1/1/2013 00:00:00
27
```

Salida

```
1 cuac
RafaelNaval 25/10/2011 13:45:11
   ¡Feliz Navidad!
2 cuac
RafaelNaval 28/11/2011 11:27:08
   Enhorabuena, campeones!
3 cuac
GinesGM 6/5/2012 16:00:00
   Dicen en #eltiempo que este...
4 cuac
Gutierrez 1/1/2013 00:00:00
   Me despido hasta la proxima. Buen viaje!
```

- El programa principal:

```
while (cin >> comando) {
   Cuac cuac;
   if (comando=="mcuac") cuac.leer_mcuac();
   else if (comando=="pcuac") cuac.leer_pcuac();
   cout << ++num << " cuac" << endl;
   cuac.escribir();
}
```