

## PARADIGMAS DE PROGRAMACIÓN

Licenciatura en Informática Programador Universitario



## **UNIDAD III**

# PARADIGMAS DE PROGRAMACIÓN

Constructores y Destructores

## Constructores y Destructores

Funciones miembros especiales de una clase que especifican la forma en que los objetos de la clase son: creados, inicializados, copiados y destruidos

```
class nuevaClase{
...
public:
    nuevaClase(...);
    ~nuevaClase();
};
```

2

### Constructores

#### Constructores

Método especial de una clase que crea un objeto e inicializa su estado

- → Se invocan automáticamente cada vez que se crea un O de la clase
- → No especifican ni tipo ni valores de retorno
- → Pueden ser sujetos de homonimia
- → No puede ser heredado por una clase derivada
- Cuando se declara un O, se pueden dar inicializadores, entre paréntesis, que serán pasados como parámetros al constructor Racional p(1,2); //p se inicializará en ½
- Pueden tener argumentos por omisión y garantizar así que el O esté en estado consistente

```
Hora(int h=0, int m=0, int s=0);//Clase Hora
Hora h1;//Desde el main
```

3

### Constructores

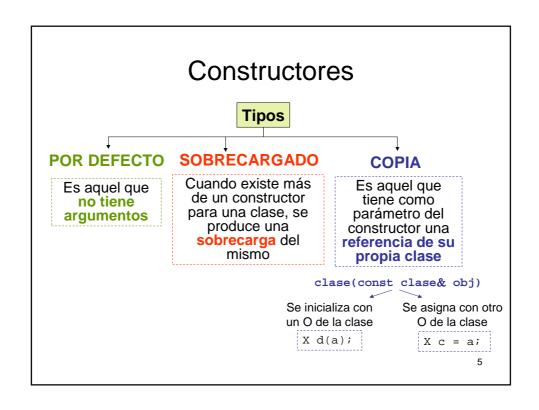
#### Constructores

Si una clase no define un constructor, el compilador creará un constructor por omisión, como public, que NO inicializará los datos miembros. La función del constructor será la de crear y copiar objetos

No se puede acceder a la dirección de los constructores y NO se pueden referenciar como una función miembro

```
Racional p(1,2);
p.Racional(1,2); //ERROR!!!
```

- Los miembros datos de la clase no pueden ser inicializados en la definición de la clase
- Un constructor no puede ser: const, volatile o static
- No se puede definir un constructor de una clase que tenga O de la misma clase como parámetros



### Tipos de constructores ${\tt class} \ {\tt X} \{$ int i,j,k; public: X (); // constructor por defecto X(const X&); // constructor copia // constructor sobrecargado X(int,int,int);// constructor sobrecargado }; main ( ){ Xa; // X :: X( ) X c(a); // X :: X(const X&) x b(20); // X :: X(int) X x(5,20,38); // X :: X(int,int,int)

```
Tipos de constructores

CONTRUCTOR

Método especial de una clase que se invoca automáticamente cada vez que se crea un objeto de dicha clase

Pila(int dim=10);
Pila(const Pila &);
...

**Pila();
};
```

```
Tipos de constructores
         Cada declaración de un objeto Pila, invocará al
                    constructor apropiado
                        Crea una pila con un arreglo de
Pila P; _____
                         tamaño 10 (valor por defecto)
                                                    Pila(int dim=10);
                     __ Crea una pila con un arreglo -
Pila Q(20); ←
                       de tamaño 20
                         Crea una pila cuyos valores son

Pila (const Pila &)
Pila copiaP(P);
             Cuando se invoca new también se invoca,
                  automáticamente, el constructor
Pila *ptrP, *ptrQ, *ptrCopia;
ptrP = new Pila();
ptrQ = new Pila(20);
 ptrCopia = new Pila(P);
                                                                 8
```

## Inicialización de los objetos y miembros de una clase

#### Asignación de valores a los objetos

Si la clase es un agregado, los O pueden ser inicializados mediante una *lista inicializadora* class agregado {

#### Agregado: clase que no tiene:

- constructor explícito de usuario
- Attb. no estáticos privados o protegidos
- funciones virtuales
- jerarquía de clases

```
class agregado {
  public: Lista
   int i,j; Inicializadora
  ...};
agregado z = (20,2);
// z.i == 20, z.j == 2;
```

 Si la clase tiene un constructor explícito, los O se inicializan con una lista expresión con ()

```
class X{
  int i,j,k;
  public:
    X ( );
    X(const X&);
    X(int);
...};
```

# Inicialización de los objetos y miembros de una clase

## Asignación de valores a datos miembros con el constructor

```
class clase {
  int dato1,dato2;
  public:
  clase (int i, int j);
};
```

 Los valores de inicialización se aceptan como parámetros del constructor y se los asigna a los miembros datos dentro del cuerpo del constructor

```
clase:: clase(int i, int j){
    dato1 = i;
    dato2 = dato1 + j;
}
```

→ Usando lista de parámetros inicializadores de expresión

```
clase:: clase(int i, int j): dato1(i), dato2(dato1 + j)
{/*cuerpo del constructor */}
```

Figura en la definición del constructor

Se escribe antes del cuerpo de la función

Cada dato miembro sólo puede ser invocado una sola vez en la lista

10

## **Destructores**

**Destructores** 

Método especial de una clase que libera el estado de un O y (opcionalmente) destruye el propio O

- Su nombre es igual al de la clase pero precedida por el carácter tilde ~
- → Se llama automáticamente cuando el O de la clase se sale de alcance
- → No recibe parámetros ni tiene valor de retorno
- → El destructor es único. No se permite la homonimia de destructores
- No puede ser heredado por una clase derivada
- No destruye el O sino que **realiza tareas de terminación** antes de que el sistema recupere el espacio de memoria ocupado por el O
- →No puede ser static, ni ser declarado const ni volatile
- Puede invocarse explícitamente para terminar un O, sin embargo el sistema lo referencia automáticamente cuando el O sale de su ámbito
- Los O creados dinámicamente con new precisan que se invoque al operador delete para que el destructor sea referenciado

**Destructores** 

Invocación

#### Indirectamente

Usando el operador delete con un O creado con el operador new

```
class X {
   public:
      X( ); //Constructor
   ~X( ); //Destructor
};

main( ) {
   X *x = new X;
   delete x; // llama a~X( )
...
...
```

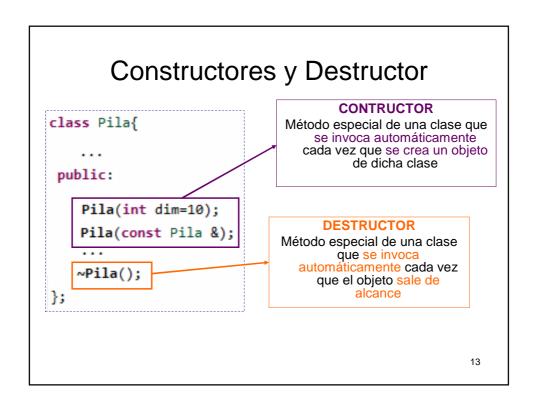
**Directamente** 

11

Usando el nombre del destructor

```
main( ) {
   X p;//crea un 0 de la clase X
   p.~X( ); // p.X :: ~X( );
}
```

La llamada explícita a un destructor se puede efectuar para un O de cualquier tipo, pero si el O no tiene destructor, no produce ningún efecto



#### Constructores y Destructor Constructor con parámetros Pila::Pila(int dim){ MAX = dim > 0 ? dim : 10;tope=-1; arreglo = reservarMemoria(MAX); **Destructor** Pila::~Pila(){ **Constructor Copia** tope = -1; MAX = 0;Pila:: Pila(const Pila &p){ delete [] arreglo; tope=p.tope; MAX = p.MAX;arreglo = reservarMemoria(MAX); for(int i=0; i<tope;i++ )</pre> arreglo[i]=p.arreglo[i]; } 14

## Constructores y Destructores

## Orden de llamada

- → Depende del orden en el cual los O entran y salen de alcance
- → El destructor se invoca, en general, en orden inverso a las llamadas a constructor
- → La persistencia de los O puede modificar el orden de llamada:
  - •Alcance global: el constructor se llama al principio de la ejecución del programa y el destructor correspondiente a la terminación.
  - •Locales automáticos: el constructor se invoca cuando el O es declarado y el destructor cuando el O sale de alcance.
  - •Locales estáticos: se llama al constructor una vez cuando se declara el O y el destructor correspondiente a la terminación del programa.
- Cuando se crea un **array de O**, los constructores de sus elementos son llamados en orden creciente de subíndice

Ver ejemplo ConstructorYDestructor 15