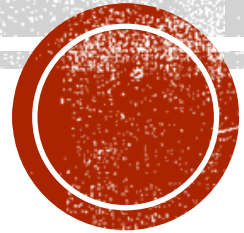
SOBRECARGA DEL CONSTRUCTOR



QUÉ ES LA SOBRECARGA DE CONSTRUCTOR

- Algunas veces hay una necesidad de inicializar un objeto de diferentes maneras.

- Esto se puede hacer usando la sobrecarga de constructores en C++. Hacerlo le permite construir objetos de varias maneras.

- A veces queremos es necesario tener diferente número de parámetros para inicializar el constructor

QUÉ ES LA SOBRECARGA DE CONSTRUCTOR

La sobrecarga se realiza la sobrecarga???

- Se puede genera más de un método constructor
- Conservan el mismo nombre de la clase
- La diferencia es que tienen diferente número de parámetros o diferente tipo de parámetros
- Y cómo es esto ….

QUÉ ES LA SOBRECARGA DE CONSTRUCTOR

Por ejemplo:

Imagina que deseamos generar un programa que tiene desea mostrar la fecha, pero puede tener como entrada de datos: día, mes y año o solo recibir la fecha en una sola variable.

Al final, en ambos casos se deberá imprimir la fecha con formato

Quedando el código de la siguiente manera:

```
class Fecha{
        private:
                int dia, mes, anio;
        public:
                Fecha(int,int,int); //Constructor 1
                Fecha(long); //Constructor 2
                void mostrarFecha();
};

//Constructor 1
Fecha::Fecha(int _dia,int _mes,int _anio){
        dia= _dia;
        mes=_mes;
        anio=_anio;
}
//Constructor 2
Fecha::Fecha(long fecha){
        anio=int(fecha/10000);//Extraer el anio
        mes=int((fecha-anio*10000)/100); //extraer el mes
        dia=int(fecha-anio*10000-mes*100); //extraer el dia
}
```

```
void Fecha::mostrarFecha(){
        cout<<"La fecha es: "<<dia<<"/"<<mes<<"/"<<anio<<endl;

}

int main(){
        Fecha hoy(18,11,2022);
        Fecha hoy2(20221118);

        hoy.mostrarFecha();
        hoy2.mostrarFecha();

        return 0;
}
```

```
La fecha es: 18/11/2022
La fecha es: 18/11/2022

--------------------------------
```

DUDAS CON LA SOBRECARGA DEL CONSTRUCTOR

Práctica en parejas - 30 min

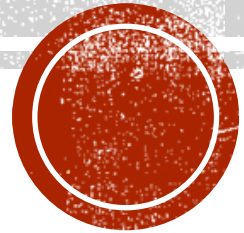
Construya una clase **Tiempo** que contenga:

1. Atributos: horas, minutos y segundos
2. Haga que la clase contenga 2 constructores:
   - El primer constructor deberá tener 3 parámetros de tiempo (int, int, int) //horas, minutos y segundos
   - El segundo constructor deberá tener 1 parámetro de tiempo (int) //tiempo en seg

     Para después convertir estos segundos a horas, minutos y segundos
3. Deberá contar con una función que muestre el resultado.

```
#import <iostream>
#import <stdlib.h>

using namespace std;

class Tiempo{
	private:
		int horas,minutos,segundos;
	public:
		Tiempo (int,int,int);
		Tiempo (int);
		void muestraTiempo();
};
//Constructor 1
Tiempo::Tiempo(int _horas,int _minutos, int _segundos){
	horas=_horas;
	minutos=_minutos;
	segundos=_segundos;
}
//Constructor 2
Tiempo::Tiempo(int tiempoSeg){
	horas=tiempoSeg/3600;
	tiempoSeg%=3600;
	minutos=tiempoSeg/60;
	segundos=tiempoSeg%60;
}

void Tiempo::muestraTiempo(){
	cout<<"La hora es: "<<horas<<" : "<<minutos<<" : "<<segundos<<endl;
}
int main(){
	Tiempo t1(9,35,50);
	t1.muestraTiempo();

	Tiempo t2(15650);
	t2.muestraTiempo();
	return 0;
}
```

DESTRUCTOR



QUÉ ES EL DESTRUCTOR

- Un destructor es una función miembro que se invoca automáticamente cuando el objeto sale del ámbito o se destruye explícitamente mediante una llamada a delete. Un destructor tiene el mismo nombre que la clase precedido por una tilde (~). Por ejemplo, el destructor de la clase String se declara como: ~String().

- Si no define un destructor, el compilador proporcionará uno predeterminado; para muchas clases, esto es suficiente. Solo tiene que definir un destructor personalizado cuando la clase almacena los identificadores de los recursos del sistema que deben liberarse, o los punteros que poseen la memoria a la que apuntan.

- Los destructores son especialmente útiles para destruir objetos de almacenamiento dinámico, es decir, aquellos para los que se reserva memoria con ayuda de un apuntador y el operador new. Ese tema se verá más adelante cuando se estudie el manejo dinámico de memoria en C++. En el siguiente ejemplo se puede observar la ejecución del destructor de una clase.

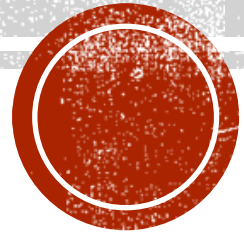
```
#import <iostream>
#import <stdlib.h>

using namespace std;

class Perro{
        private:
                string nombre,raza;
        public:
                Perro(string,string);
                ~Perro();
                void mostrarDatos();
                void jugar();
};
//Constructor
Perro::Perro(string _nombre, string _raza){
        nombre=_nombre;
        raza=_raza;
}
//Destructor
Perro::~Perro(){
}

void Perro::mostrarDatos(){
        cout<<"El nombre del perro es: "<<nombre<<endl;
        cout<<"Es de raza: "<<raza<<endl;
}

void Perro::jugar(){
        cout<<"El perro " <<nombre<<" esta jugando"<<endl;
}

int main(){
        Perro perro1("Bruno","Pug");
        perro1.mostrarDatos();
        perro1.jugar();
        perro1.~Perro();
        return 0;
}
```

El Destructor:

1. Se declara
2. Se inicializa
3. Se manda a llamar

```
El nombre del perro es: Bruno
Es de raza: Pug
El perro Bruno esta jugando

--------------------------------
```

CETTERS Y SETTERS



MÉTODOS GETTER Y SETTER

- Cuando queremos tener acceso a propiedades de un objeto, muchas veces se nos hace complicado hacerlo de manera directa ya que por alguna regla tenemos que definirla dentro de la clase como propiedades protegida o privada. Los métodos get y set nos proveen la habilidad de acceder a estas propiedades.

  - **get** Este método presta atención al momento que hacemos una solicitud de alguna propiedad dentro de la clase que no sea pública.

  - **set** No permite efectuar cambio a propiedades el cual tenemos protegidas o privada dentro de nuestra definición de clases.

  Veamos un ejemplo ….

1. Declaración de la clase

```
//Ejemplo de Getter y Setter

#import <iostream>
#import <stdlib.h>

using namespace std;

class Puntos{
      private:
            int x,y;
      public:
            Puntos( );
            void setPuntos(int,int);
            int getPuntoX();
            int getPuntoY();
};
```

Los métodos **set** llevan parámetros, los métodos **get** regresan valor.

2. Inicialización de los métodos

```
Puntos::Puntos( ){
}
```

Los métodos **constructor nos lleva parámetros ni cuerpo**

```
//Establecemos valores a los atributos
void Puntos::setPuntos(int _x,int _y){
        x=_x;
        y=_y;
}
```

Los métodos **Setter** se utiliza para inicializar los atributos

```
//Obtener el valor del punto X
int Puntos::getPuntoX( ){
        return x;
}

//Obtener el valor del punto Y
int Puntos::getPuntoY( ){
        return y;
}
```

Los métodos **Getter** se utiliza obtener los valores de los atributos

2. Inicialización de los métodos

```
int main(){
        int puntoX,puntoY;
        Puntos c1; //Se crea el objeto
        c1.setPuntos(10,15); //Se inicializan los valores de los atributos
        puntoX=c1.getPuntoX(); //Se obtiene el valor x
        puntoY=c1.getPuntoY(); //Se obtiene el valor y
        cout<<"Las coordenadas del punto en el plan son: "<<puntoX<<" , "<<puntoY<<endl;
}
```

```
Las coordenadas del punto en el plan son: 10 , 15
```

MÉTODOS GETTER Y SETTER

Getters y Setter vs Constructor clásico

MÉTODOS GETTER Y SETTER

- Ahora si, resolvamos la actividad 5 …

REFLEXIÓN DE HOY

- Sobrecarga de Constructores:
  - Cuándo se implementa?
- Destructores
  - Siempre es necesario usarlo?
- Getters y Setters
  - Por qué se debe usar.
  - Lo usamos siempre?