

Revisión: Apuntadores & Datos Dinámicos

- Una variable apuntadora es una **variable cuyo valor es la dirección de una posición en la memoria.**

```
int x;  
x = 5;  
  
int* ptr1;  
ptr1 = &x;  
  
int* ptr2;  
ptr2 = ptr1;  
*ptr1 = 6;  
  
cout << ptr1 << endl;  
cout << *ptr2 << endl;
```

```
int* ptr3;  
ptr3 = new int;  
*ptr3 = 5;  
delete ptr3;  
ptr3 = NULL;  
  
int *ptr4;  
ptr4 = new int[5];  
ptr4[0] = 100;  
ptr4[4] = 123;  
delete [] ptr4;  
ptr4 = NULL;
```

Revisión: Tipos de referencia

- Tipos de referencia
 - Alias para otra variable
 - Debe inicializarse cuando se declara
 - Son usados principalmente como parámetros en una función

```
int main (void){  
    int a1 = 5, a2 = 10;  
    int *a3 = new int;  
    *a3 = 15;  
    int &a4 = a3;  
    cout << a1 << a2 << a3 << endl;  
    incrementar (a1, a2, a3);  
    cout << a1 << a2 << a3 << endl;  
    delete a3; a3 = NULL;  
    return 0;  
}
```

```
void incrementar(int b1, int &b2, int *b3)  
{  
    b1 += 2;  
    b2 += 2;  
    *b3 += 2;  
}
```

Programación Orientada a Objetos

Introducción a Clases

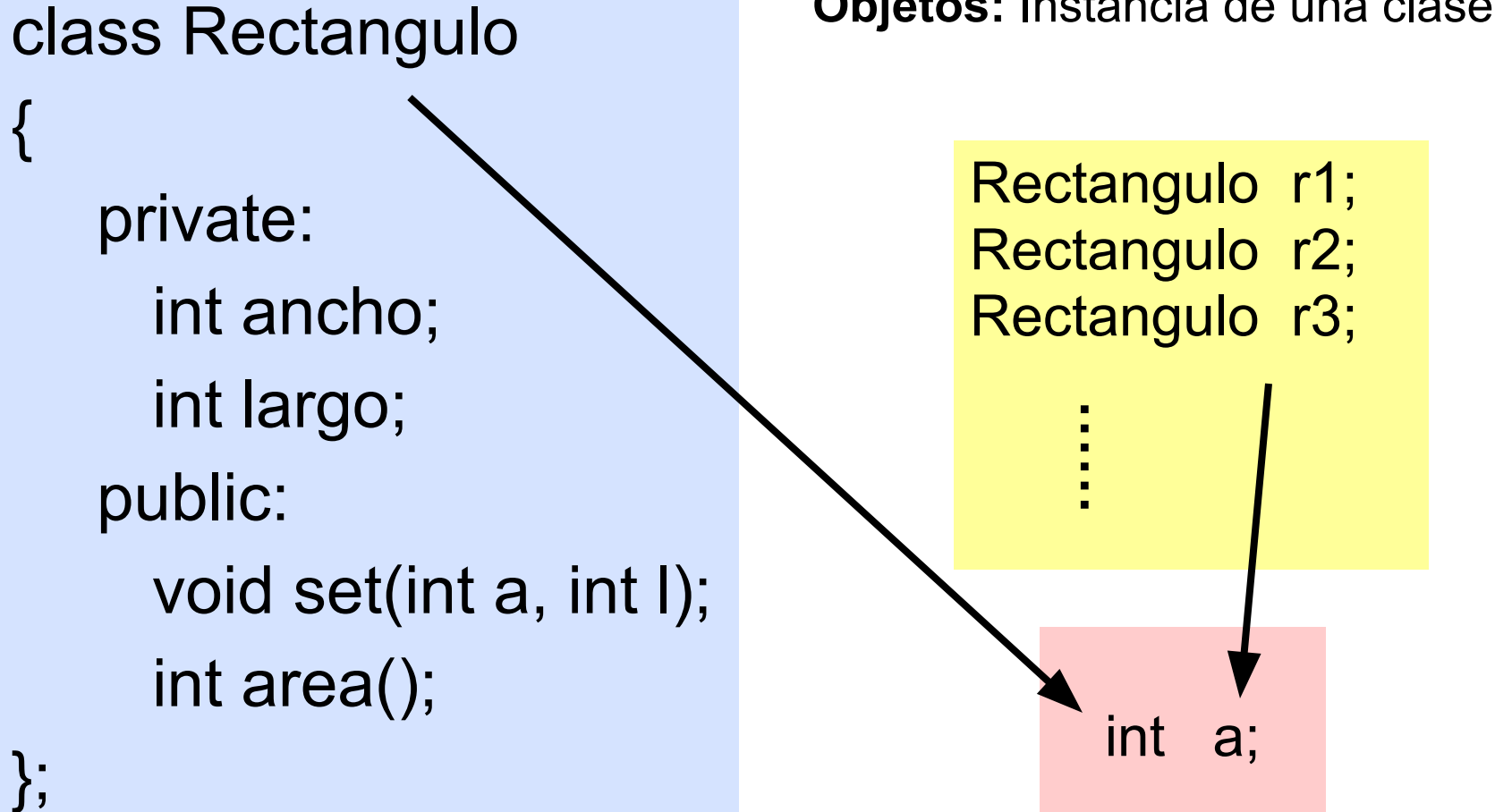
- Definición de Clase
- Ejemplo de Clase
- Objetos
- Constructores
- Destrucción

Clase

- La clase es la **pie**dra angular de C++
 - Hace posible encapsulación, ocultamiento de datos y herencia
- Tipo
 - Representación Concreta de un concepto
 - Ejm. **float** con operaciones como -, *, + (números reales matemáticos)
- Clase
 - Un tipo definido por el usuario
 - Consiste de datos y métodos
 - Define propiedades y comportamiento de ese tipo
- Ventajas
 - Programa Conciso
 - Fácil análisis de código
 - El compilador puede detectar usos ilegales de tipos
- Abstracción de Datos
 - Separa los detalles de la implementación de sus propiedades esenciales

Clases & Objetos

```
class Rectangulo
{
    private:
        int ancho;
        int largo;
    public:
        void set(int a, int l);
        int area();
};
```



Objetos: Instancia de una clase

```
Rectangulo r1;
Rectangulo r2;
Rectangulo r3;
```

...

```
int a;
```

Definición de un Tipo Clase

Encabezado

class *nombre_clase*

{

etiqueta_permiso:
miembro;

etiqueta_permiso:
miembro;

...

};

Cuerpo

class Rectangulo

{

private:

int ancho;

int largo;

public:

void set(int a, int l);

int area();

};

Definición de Clase

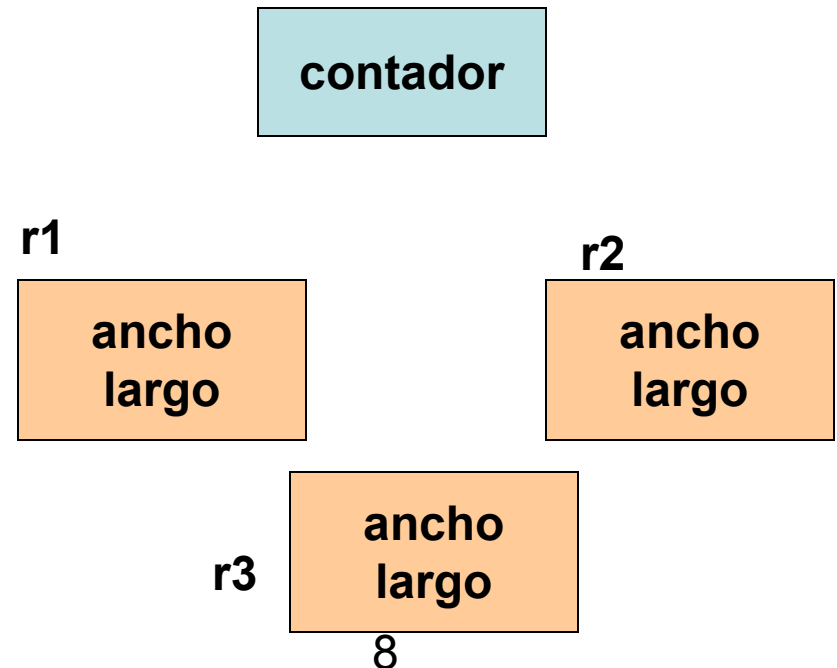
Datos Miembros

- Pueden ser de cualquier tipo, internas o definidas por el usuario
- **datos miembros *no-estáticos***
 - Cada objeto de clase tiene su propia copia
- **datos miembros *estáticos***
 - Actúa como una variable global
 - Una copia por tipo clase, ejm. contador

Datos Miembro Estáticos

```
class Rectangulo
{
    private:
        int ancho;
        int largo;
    → static int contador;
    public:
        void set(int a, int l);
        int area();
}
```

```
Rectangulo r1;
Rectangulo r2;
Rectangulo r3;
```



Definición de Clase

Funciones Miembro

- Usada para
 - acceder a los valores de los datos miembro (**accesor**)
 - realizar operaciones en los datos miembros (**implementador**)
- Son declarados al interior del cuerpo de la clase
- Su definición puede ubicarse al interior del cuerpo de la clase, o al exterior del cuerpo de la clase
- Puede acceder tanto a miembros públicos como privados de la clase
- Puede ser referido usando el operador de acceso punto o flecha

Definición de una Función Miembro

```
class Rectangulo
{
    private:
        int ancho, largo;
    public:
        void set (int a, int l);
        int area() {return ancho*largo; }
};
```

nombre clase

nombre función miembro

inline

```
r1.set(5,8);
rp->set(8,10);
```

```
void Rectangulo :: set (int a, int l)
{
    ancho = a;
    largo = l;
}
```

operador de ambito

Definición de Clase

Funciones Miembro

- **const función miembro**

- **declaración**

- *tipo_retorno* *nombre_func* (*lista_param*) const;

- **definición**

- *tipo_retorno* *nombre_func* (*lista_param*) const { ... }

- *tipo_retorno* *nombre_clase* :: *nombre_func_name* (*lista_param*) const { ... }

- **No hace ninguna modificación sobre los datos miembros (función segura)**

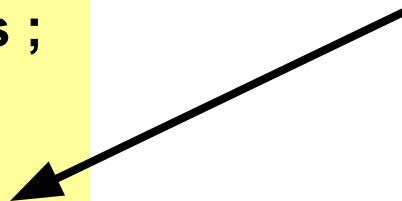
- **Es ilegal para una función miembro const modificar un dato miembro de una clase.**

Función Miembro Const

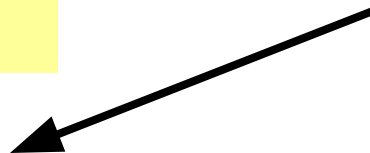
```
class Tiempo
{
    private :
        int    hrs, mins, segs ;

    public :
        void    Escribe ( )
        const ;
};
```

declaración de la función



definición de la función



```
void Tiempo :: Escribe( ) const
{
    cout <<hrs << ":" << mins << ":" << segs << endl;
}
```

Definición de la Clase - Control de Acceso

- **Ocultación de la Información**
 - Para evitar que la representación interna tenga acceso directo desde fuera de la clase.
- **Especificadores de Acceso**
 - **public**
 - Puede ser accesible desde cualquier lugar dentro de un programa
 - **private**
 - Solo pueden acceder las funciones miembros y amigas de esta clase.
 - **protected**
 - Actúa como público para las clases derivadas.
 - Se comporta como privado para el resto del programa.

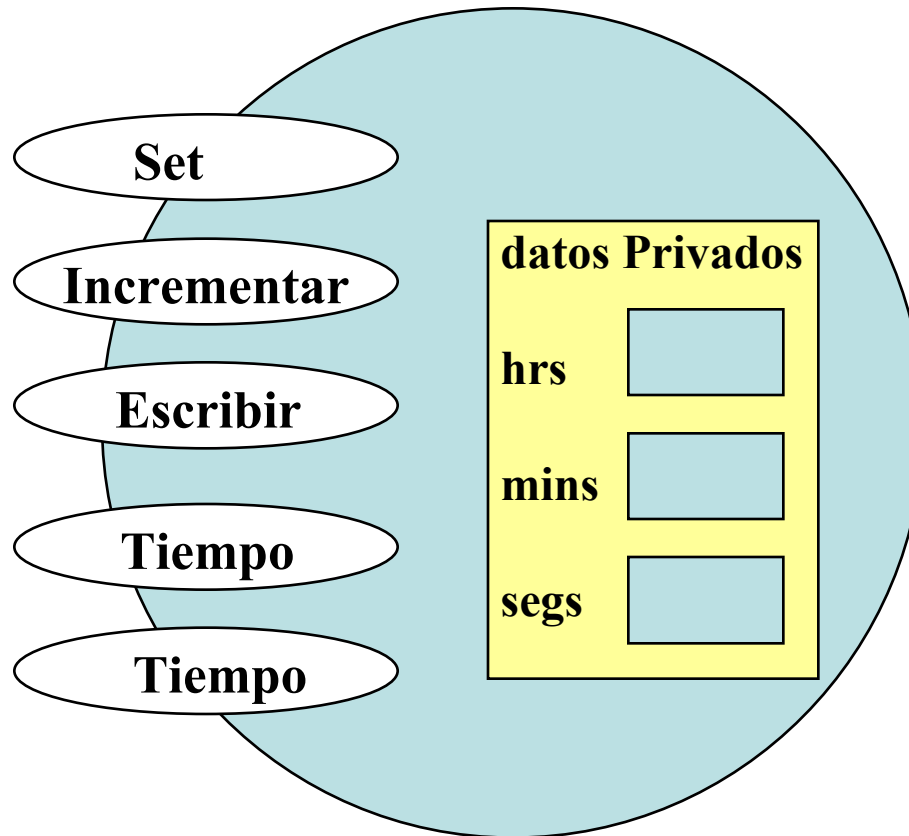
Especificación de la clase Tiempo

```
class Tiempo
{
    public :
        void    Set ( int horas , int minutos , int segundos ) ;
        void    Incrementar ( ) ;
        void    Escribir ( ) const ;
        Tiempo  ( int iniHrs, int iniMins, int iniSegs ) ; //
constructor
        Tiempo  ( ) ; // constructor predeterminado

    private :
        int     hrs ;
        int     mins ;
        int     segs ;
};
```

Diagrama de Interfaz de la Clase

clase Tiempo



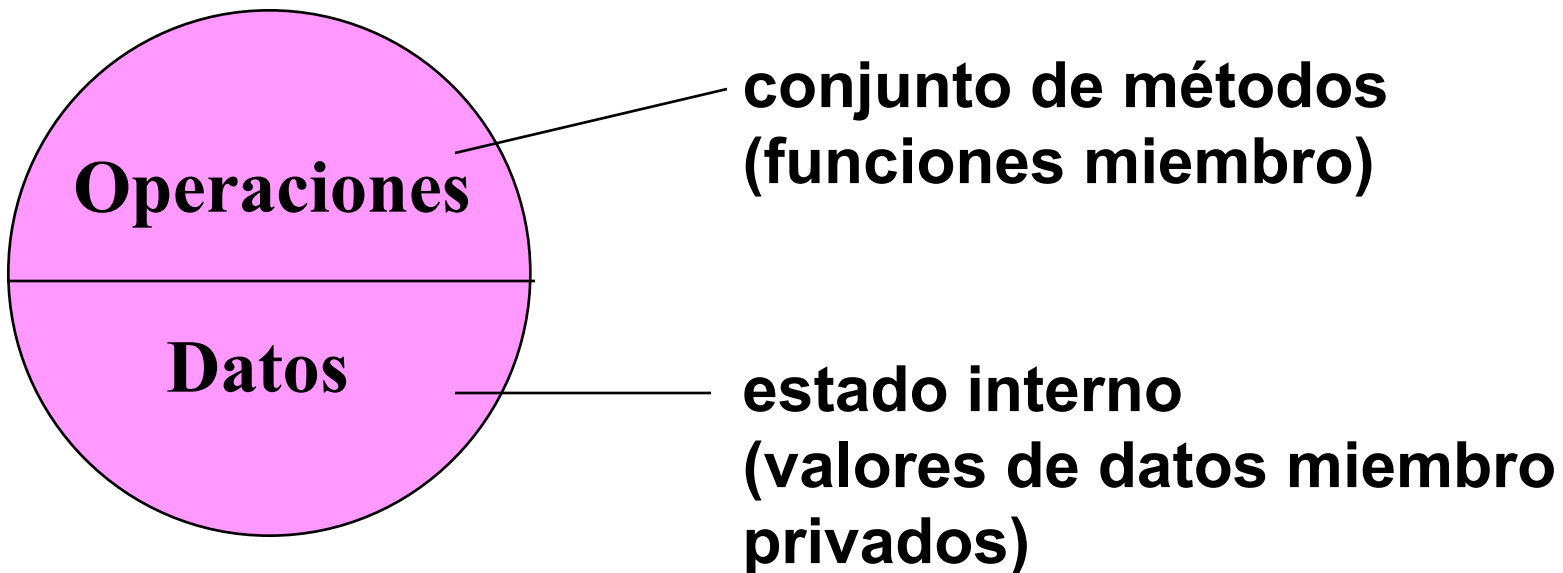
Definición de la Clase

Control de Acceso

- El especificador de acceso predeterminado es *privado*
- Los datos miembros son usualmente **private** o **protected**
- Una función miembro **private** puede solo ser accesada por otra función miembro de la misma clase (a excepción de la función *friend*)
- Las funciones miembro **public** son parte de la *interfaz de la clase*
- Cada sección de control de acceso es opcional, repetible, y las secciones pueden aparecer en cualquier orden

¿Qué es un objeto?

OBJETO



Declaración de un Objeto

```
class Rectangulo
{
    private:
        int ancho;
        int largo;
    public:
        void set(int a, int l);
        int area();
};
```

```
main()
{
    Rectangulo r1;
    Rectangulo r2;

    r1.set(5, 8);
    cout<<r1.area()<<endl;

    r2.set(8,10);
    cout<<r2.area()<<endl;
}
```

Otro Ejemplo

```
#include <iostream.h>
class circulo
{
    private:
        double radio;

    public:
        void asignar(double);
        double area(void);
        void mostrar(void);
};
```

```
// definiciones de funciones miembro
void circulo::asignar(double r)
{
    radio = r;
}

double circulo::area(void)
{
    return 3.14*radio*radio;
}
void circulo::mostrar(void)
{
    cout << "r = " << radio << endl;
}
```

```
int main(void) {
    circulo c; // un objeto de la clase circulo
    c.asignar(5.0);
    cout << "El area del circulo c es " << c.area() << endl;
    c.mostrar();
}
```

Declaración de un Objeto

```
class Rectangulo
{
    private:
        int ancho;
        int largo;
    public:
        void set(int a, int l);
        int area();
};
```

r1 se asigna estáticamente

```
main()
{
    Rectangulo r1;
    ➔ r1.set(5, 8);
}
```

r1

**ancho = 5
largo = 8**

Declaración de un Objeto

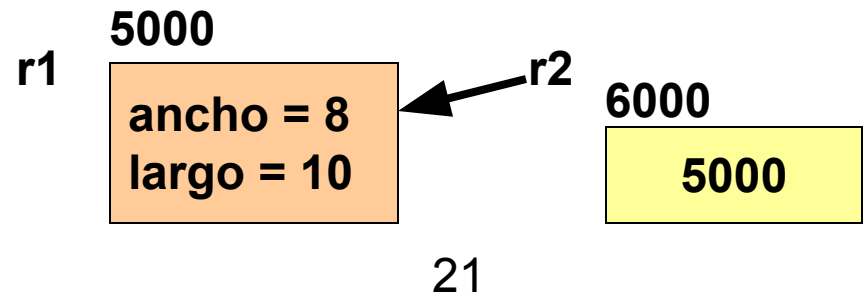
```
class Rectangulo
{
    private:
        int ancho;
        int largo;
    public:
        void set(int a, int l);
        int area();
};
```

r2 es un apuntador a un objeto Rectangulo

```
main()
{
    Rectangulo r1;
    r1.set(5, 8);
    Rectangulo *r2;
    r2 = &r1;
    r2->set(8,10);
}
```

// notación punto

//notación flecha



Declaración de un Objeto

```
clase Rectangulo
{
    private:
        int ancho;
        int largo;
    public:
        void set(int a, int l);
        int area();
};
```

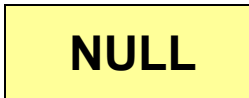
r3 se asigna dinámicamente

```
main()
{
    Rectangulo *r3;
    r3 = new Rectangulo();

    r3->set(80,100)//notacion flecha

    delete r3;
    ➡ r3 = NULL;
}
```

r3
6000
NULL



Inicialización de un Objeto

1. Por Asignación

```
#include <iostream.h>

class circulo
{
    public:
        double radio;
};
```

- Solo trabaja para datos miembro públicos
- No hay control sobre las operaciones en los datos miembro

```
int main()
{
    circulo c1;           // Declara una instancia de la clase circulo
    c1.radio = 5;         // Inicializa por asignación
}
```

Inicialización de un Objeto

```
#include <iostream.h>

class circulo
{
private:
    double radio;

public:
    void set (double r)
        {radio = r;}
    double get_r ()
        {return radio;}
};
```

2. Por Funciones Miembro Públicas

```
int main(void) {
    circulo c;           // un objeto de la clase circulo
    c.set(5.0);          // inicializa un objeto con una función miembro publica
    cout << "El radio del circulo c es " << c.get_r() << endl;
    // acceso a un dato miembro privado con un accesor
}
```


Inicialización de un Objeto

```
class Rectangulo
{
    private:
        int ancho;
        int largo;
    public:
        void set(int a, int l);
        int area();
}
```

r2 es un apuntador a un objeto Rectangulo

```
main()
{
    Rectangulo r1;
    r1.set(5, 8); //notación punto

    Rectangulo *r2;
    r2 = &r1;
    r2->set(8,10); //notación flecha
}
```

r1 y r2 están inicializados por la función miembro pública **set**

Inicialización de un Objeto

```
class Rectangulo
{
    private:
        int ancho;
        int largo;
    public:
        Rectangulo();
        Rectangulo(const Rectangulo
&r);
        Rectangulo(int a, int l);
        void set(int a, int l);
        int area();
}
```

3. Por Constructor

- constructor predeterminado
- constructor copia
- Constructor con parámetros

Ellos son accesibles públicamente

Tienen el mismo nombre que el de la clase

No tienen tipo de retorno

Son usados para inicializar los datos miembros de la clase

Tienen diferentes parámetros

Inicialización de un Objeto

```
class Rectangulo
{
    private:
        int ancho;
        int largo;
    public:
        void set(int a, int l);
        int area();
};
```

Cuando una clase se declara sin constructores el compilador automáticamente asume el constructor predeterminado y **copia el** constructor.

- constructor predeterminado

```
Rectangulo :: Rectangulo() { };
```

- constructor copia

```
Rectangulo :: Rectangulo (const
    Rectangulo & r)
{
    ancho = r.ancho; largo = r.largo;
};
```

Inicialización de un Objeto

- Inicializar con constructor predeterminado

```
class Rectangulo
{
    private:
        int ancho;
        int largo;
    public:
        void set(int a, int l);
        int area();
}
```

```
Rectangulo r1;
Rectangulo *r3 = new Rectangulo();
```

- Inicializar con constructor **copia**

```
Rectangulo r4;
r4.set(60,80);

Rectangulo r5 = r4;
Rectangulo r6(r4);

Rectangulo *r7 = new Rectangulo(r4);
```

Inicialización de un Objeto

```
class Rectangulo
{
    private:
        int ancho;
        int largo;
    public:
        Rectangulo(int a, int l)
            {ancho =a; largo=l;}
        void set(int a, int l);
        int area();
}
```

Si se declara cualquier constructor con cualquier número de parámetros, no existirá ningún constructor predeterminado, a menos que lo defina.

```
Rectangulo r4; // error
```

- Inicializar con constructor

```
Rectangulo r5(60,80);
```

```
Rectangulo *r6 = new Rectangulo(60,80);
```

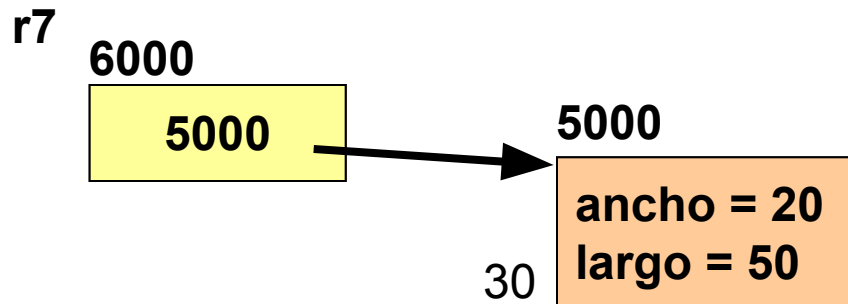
Inicialización de un Objeto

Escribe tus propios constructores

```
class Rectangulo
{
    private:
        int ancho;
        int largo;
    public:
        Rectangulo();
        Rectangulo(int a, int l);
        void set(int a, int l);
        int area();
}
```

```
Rectangulo :: Rectangulo()
{
    ancho = 20;
    largo = 50;
};
```

```
Rectangulo *r7 = new Rectangulo();
```



Inicialización de un Objeto

```
class Cuenta
{
    private:
        char *nombre;
        double saldo;
        unsigned int id;
    public:
        Cuenta();
        Cuenta(const Cuenta &c);
        Cuenta(const char
*persona);
}
```

```
Cuenta :: Cuenta()
{
    nombre = NULL; saldo = 0.0;
    id = 0;
};
```

Con constructores, tenemos más control sobre los datos miembro

```
Cuenta :: Cuenta(const Cuenta &c)
{
    nombre = new char[strlen(c.nombre)+1];
    strcpy (nombre, c.nombre);
    saldo = c.saldo;
    id = c.id;
};
```

```
Cuenta :: Cuenta(const char *persona)
{
    nombre = new char[strlen(persona)+1];
    strcpy (nombre, persona);
    saldo = 0.0;
    id = 0;
};
```

Hasta aquí, ...

- Un objeto puede ser inicializado por un constructor
 - constructor predeterminado
 - constructor copia
 - constructor con parámetros
- Los recursos se asignan cuando se inicializa un objeto.
- Los recursos deben ser revocados cuando un objeto está a punto de terminar su vida útil

Limpieza de un Objeto

Destructor

```
class Cuenta
{
    private:
        char *nombre;
        double saldo;
        unsigned int id; //unique
    public:
        Cuenta();
        Cuenta(const Cuenta &a);
        Cuenta(const char
*persona);
        ~Cuenta();
}
```

```
Cuenta :: ~Cuenta()
{
    delete[] nombre;
}
```

- Su nombre es el nombre de la clase precedido por una ~ (tilde)
- **No tiene argumento**
- Se utiliza para liberar memoria asignada dinámicamente y para realizar otras actividades de "limpieza"
- **Se ejecuta automáticamente cuando el objeto queda fuera de alcance.**

Juntándolos

```
class Frase
{
    char *pDato;
    int Longitud;
public:
    //constructores
    Frase();
    Frase(char *s);
    Frase(const Frase
&cadena);

    //accesores
    char* get_Dato();
    int get_Longitud();

    //destructor
    ~Frase();
};
```

```
Frase :: Frase() {
    pDato = new char[1];
    *pDato = '\\0';
    Longitud = 0;
};
```

```
Frase :: Frase(char *s) {
    pDato = new
char[strlen(s)+1];
    strcpy(pDato, s);
    Longitud = strlen(s);
};
```

```
Frase :: Frase (const Frase &cadena) {
    int n = cadena.Longitud;
    pDato = new char[n+1];
    Longitud = n;
    strcpy(pDato, cadena.pDato);
};
```

Juntándolos

```
class Frase
{
    char *pDato;
    int Longitud;
public:
    //constructores
    Frase();
    Frase(char *s);
    Frase(const Frase &cadena);

    //accesores
    char* get_Dato();
    int get_Longitud();
    //destructor
    ~Frase();
};
```

```
char* Frase :: get_Dato()
{
    return pDato;
};
```

```
int Frase :: get_Longitud()
{
    return
    Longitud;
};
```

```
Frase :: ~Frase()
{
    delete[] pDato;
};
```

Juntándolos

```
class Frase
{
    char *pDato;
    int Longitud;
public:
    //constructores
    Frase();
    Frase(char *s);
    Frase(const Frase
&cadena);

    //accesores
    char* get_Dato();
    int get_Longitud();

    //destructor
    ~Frase();
};
```

```
int main()
{
    int x=3;
    Frase *pFrase1 = new
Frase("Jose");
    Frase *pFrase2 = new Frase();
}
```