

## Metodologías y Técnicas de Programación II

# Programación Orientada a Objeto (POO) C++

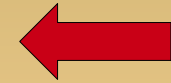
## Funciones “inline”

# Estado del Programa

## Introducción a la POO

Historia de la Programación  
Conceptos de POO

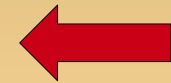
C++  
Mi primera Clase



## Repaso de Conceptos

Estándares de Programación  
Punteros y Memoria

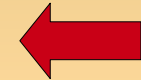
E/S  
Control y Operadores



## Clases y Objetos en C++

Uso y aplicación  
Constructores  
Constantes e "inline"

Funciones Amigas  
Sobrecarga de Funciones



## Sobrecarga

De Operadores

Creación Dinámica de Objetos

## Herencia.

Tipos de Visibilidad

Herencia Múltiple

## Polimorfismo

Funciones Virtuales

Polimorfismo y Sobrecarga.

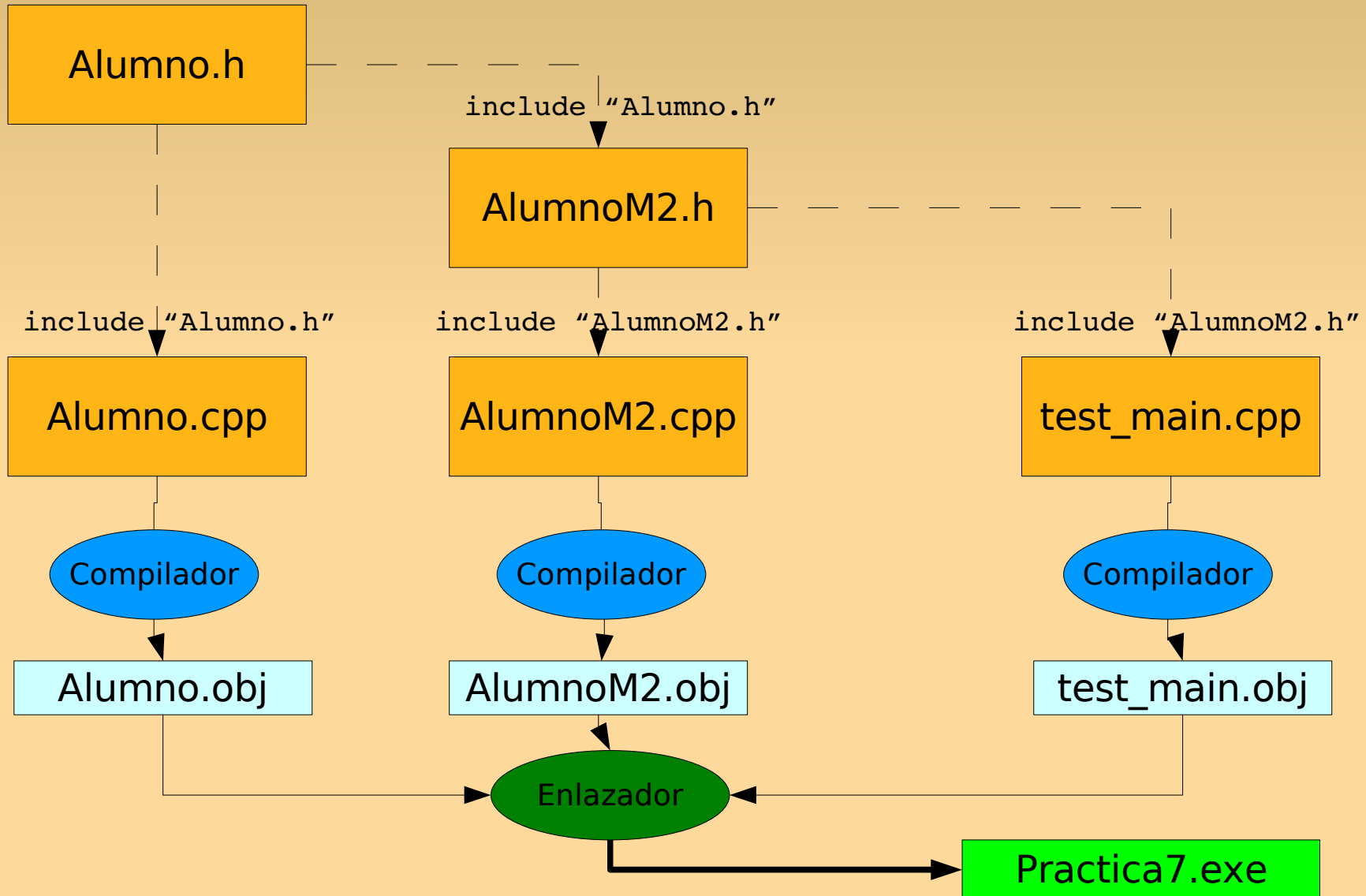
## Plantillas

Contenedores

Iteradores

# 10.1 Repaso de Sesiones Anteriores

## Compilación y Enlazado por separado



# 10.1 Repaso de Sesiones Anteriores

## **const**

El motivo unicial para usar *const* fue para eliminar el uso de la directiva de compilación `#define` para sustitución de valores. Tiene ventajas sobre `#define`.

```
#define NOTA_ESPERADA 8
```

```
const float nota_esperada = 8;
```

```
const int cantidad = 19;
```

```
Alumnos curso[cantidad];
```

Punteros constantes:

```
// "p es un puntero que apunta //  
a: un entero constante.
```

```
const int* p;
```

```
const int *p;
```

```
// "w es un puntero constante  
// que apunta a: un entero.
```

```
int i=3;
```

```
int* const w = &i;
```

```
*w = 2; // ¿Está bien o está mal?
```

En C++ lo mejor es utilizar una referencia constante en lugar de paso por valor.

Un método que no modifica ningún atributo se debería escribir como constante y así se podría usar desde objetos constantes.

# 10.2 Funciones inline

## Conceptos previos

Cuando escribimos el nombre de una función dentro de un programa decimos que "llamamos" a esa función.

Esto quiere decir que lo que hace el programa es "saltar" a la función, ejecutarla y retornar al punto en que fue llamada.



Con funciones "inline" en lugar de existir una única copia de la función dentro del código, cuando se declara una función como "inline" lo que se hace es insertar su código en el lugar en que se realiza la llamada, en lugar de invocar a la función.

# 10.2 Funciones inline

## Eficiencia

C++ se pensó como un lenguaje que aunara técnicas de **Programación Orientada a Objeto** (para reutilizar de forma sistemática), y eficiencia, **rapidez**.

## Funciones inline

El código generado para la función cuando el programa se compila, **se inserta en el punto donde se invoca a la función**, en lugar de hacerlo en otro lugar y hacer una llamada.

El código de estas funciones **se ejecuta más rápidamente**, ya que se evita usar la pila para pasar parámetros y se evitan las instrucciones de salto y retorno

También tiene un inconveniente: se generará el código de la función tantas veces como ésta se use, con lo que el programa **ejecutable** final puede ser mucho **más grande**.

Es por esos dos motivos por los que sólo se usan funciones inline cuando las **funciones son pequeñas**.

# 10.2 Funciones inline

## Declaración de Funciones inline

Hay dos maneras como hemos visto:

```
class Ejemplo
{
    public:
        Ejemplo(int a = 0) : a_(a) {}
    private:
        int a_;
}
```

1

```
class Ejemplo
{
    public:
        Ejemplo(int a = 0);
    private:
        int a_;
};
inline Ejemplo::Ejemplo(int a) : A(a)
{
}
```

2

Cada vez que declaremos un objeto de la clase Ejemplo se insertará el código correspondiente a su constructor.

Si no es “inline” cuando declaremos objetos de la clase Ejemplo se hará una llamada al constructor y sólo existirá una copia del código del constructor en nuestro programa.

# 10.2 Funciones inline

## Funciones inline

Usar funciones inline **es transparente en su uso** (main() o donde sea), y así debe ser.

El comportamiento lógico de una función debe ser idéntico aunque sea inline (de otro modo su compilador no funciona).

La única diferencia visible es el **rendimiento**.

La tentación es usar declaraciones inline **en cualquier parte** dentro de la clase porque ahorran el paso extra de hacer una definición de método externa. **¡MAL!**

La idea de una función inline es **dar al compilador mejores oportunidades** de optimización, pero si declaramos inline una función grande provocaremos que el código se duplique allí donde se llame la función, y anularemos el beneficio de velocidad obtenido.

# 10.2 Funciones inline

## Ejemplo

```
#include <iostream.h>

//#define ENLINEA inline
#define ENLINEA
class Contador
{
private:
    int x_;
public:
    Contador();
    void poner(const int& x);
    int leer(void) const;
    void incrementar(void);
};

ENLINEA    Contador::Contador()    :
x_(0) {};
```

```
ENLINEA    void    Contador::poner(const
int& x)
{
    if (x > 0)
        x_ = x;
    else
        x_ = 0;
}
ENLINEA int Contador::leer(void) const
{
    return x_;
};
ENLINEA
void Contador::incrementar(void)
{
    x_++;
}
```

# 10.2 Funciones inline

## Ejemplo

```
// Utilizamos const en lugar de #define
const int num_contadores = 20000;
// Definimos un array de objetos de la clase
// Contador. Se llama al constructor por cada uno.
Contador contadores[num_contadores];
for (int i=0;i<num_contadores;i++)
{
    contadores[i].poner(10);
    for (int j=0;j<num_contadores;j++)
    {
        contadores[i].poner(j);
        contadores[i].incrementar();
    }
    //cout << "Contador numero" << i <<
    //      " con Valor: " <<
    //      contadores[i].leer() << endl;
}
```

```
jlmarina@PWPRT005:~/tmp/inline$ time ./si
real    0m1.009s
user    0m1.004s
sys     0m0.004s

jlmarina@PWPRT005:~/tmp/inline$ time ./no
real    0m3.696s
user    0m3.688s
sys     0m0.008s
```

## 10.3 Resumen

### **inline**

Sustituye la utilización de macros que es muy común en la programación en C.

Se pensó para dotar al lenguaje C++ de mayor eficiencia.

Es una sugerencia para el compilador.

Hay dos maneras de definir un método de una clase como “inline”. Es preferible la que se utiliza en la implementación y no en la declaración de la clase.