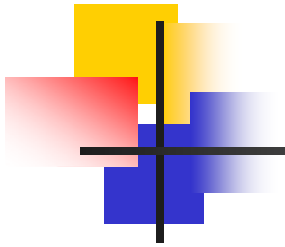
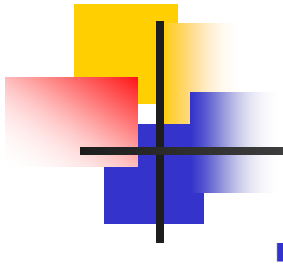


CLASES EN C++

1.- Clases y objetos en C++
2.- Acceso a los miembros de una clase
3.- Métodos de una clase
4.- Ejemplo: Clase CComplejo
5.- Sobrecarga de operadores
6.- El puntero this
7.- Otras características de C++
8.- Resumen de las características de C++



1. Clases y objetos en C++

- Encapsulamiento:
  - Una clase combina datos de diferentes tipos junto a las funciones que los procesan.
- Abstracción:
  - Una clase permite implementar tipos de datos abstractos (generalización).
- Una clase está formada por miembros (datos y funciones) que pueden ser:
  - Privados (`private:`): son miembros que sólo pueden ser utilizados por funciones de la clase.
  - Públicos (`public:`): son miembros que pueden ser utilizados fuera de la clase.
  - Protegidos (`protected:`): son miembros privados que pueden ser utilizados por otra clase derivada (*Herencia*).
- Las funciones de una clase se denominan *métodos*.
- Un objeto es una instancia de una clase (*equivale a una variable de C*).

2. Acceso a los miembros de una clase

- Los datos y funciones (*métodos*) de una clase se invocan de la misma forma que los campos de una estructura:
  - Datos:

```
variable.campo1 = 3;
```

  - Funciones:

```
variable.Visualizar();
```

  - Si utilizamos un puntero:

```
pvariable->campo1 = 3;

pvariable->Visualizar();
```

Objeto de una clase definida por el programador

3. Métodos de una clase

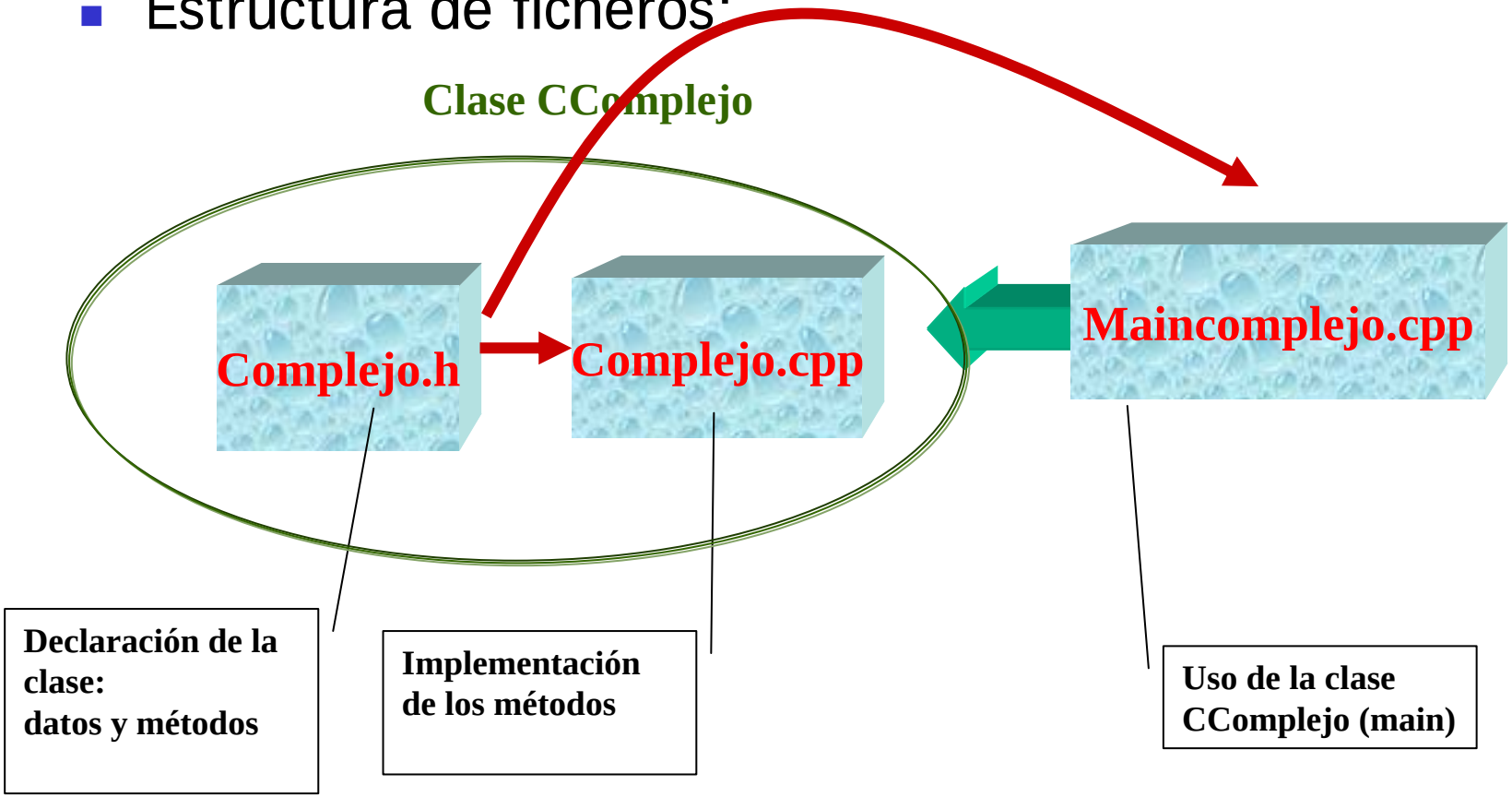
- Métodos predefinidos por el compilador (pueden ser *sobrepasados*):
  - Constructores: son llamados cuando se crea un objeto
    - Constructor por defecto
    - Constructores parametrizados
    - Constructor de copia
  - Destructor: es llamado cuando el objeto desaparece. Debe liberar los recursos que obtuvo el objeto en su construcción. P.e. liberar memoria, cerrar ficheros.
  - Operadores:
    - Operador Asignación

Redefinidos por el usuario

- Métodos definidos por el programador:
  - Funciones que necesitemos definir para implementar el algoritmo que procesa los datos de la clase o su interacción con otras.
  - Sobrecarga de operadores estándar (+ - * .....)

4. Ejemplo: Clase CComplejo

- Estructura de ficheros:



Ejemplo: Clase CComplejo (complejo.h)

```
#ifndef __COMPLEJO_H__
#define __COMPLEJO_H__

// Definición de tipos de datos y clases
class CComplejo {
private:
    float real;          // parte real
    float imag;          // parte imaginaria

public:
    CComplejo(void);                 // constructor por defecto
    CComplejo(float num_real, float num_imag=0); // constructor a partir de dos reales
    CComplejo(CComplejo &compl_orig);                    // constructor copia
    ~CComplejo(void);                                          // destructor

    void AsignarReal(float num_real);          // Asigna la parte real
    void AsignarImag(float num_imag);          // Asigna la parte imaginaria
    void Visualizar(void);                     // Muestra en pantalla el número complejo

    CComplejo& operator = (CComplejo & compl_orig);   // operador asignación
    CComplejo operator + (CComplejo &a);    // operador suma

};   // fin de la clase CComplejo

#endif
```

Ejemplo: Clase CComplejo (complejo.cpp)

```
#define __COMPLEJO_CPP__

//Librerías Estándar
#include <iostream.h>
#include <math.h>

//Encabezado del archivo
#include "..\inc\complejo.h"

//constructor por defecto
CComplejo::CComplejo(void)
{
    real = 0.0;
    imag = 0.0;
}
// Constructor con parámetros
CComplejo::CComplejo(float num_real, float num_imag)
{
    real = num_real;
    imag = num_imag;
}
```

```
// Constructor copia
CComplejo::CComplejo(CComplejo &compl_orig)
{
    real = compl_orig.real;
    imag = compl_orig.imag;
}

//Destructor
CComplejo::~CComplejo(void)
{
}
```

Ejemplo: Clase CComplejo (complejo.cpp)

```
// .... Continúa complejo.cpp

void CComplejo::AsignarReal(float num_real)
{
    real = num_real;
}
void CComplejo::AsignarImag(float num_imag)
{
    imag = num_imag;
}
```

```
void CComplejo::Visualizar(void)
{
    cout << real << “ + ”<< imag << "i\n";
}
```

Notas:

- Al definir la clase se debe definir un constructor. Éste es llamado cada vez que se crea un nuevo objeto.
- Funciones del constructor:
  - Inicializar variables.
  - Reservar memoria para tipos de datos.

```
void main(void)
{
   CComplejo c1,c4;            // Constructor por defecto
   CComplejo c2(3,2);          // Constructor con parámetros
   CComplejo c5(3);            // Constructor con parámetros
   CComplejo c3(c1);                 //Constructor copia
}
```

- Cuando el objeto deja de existir se llama al destructor. Éste deberá liberar la memoria que haya solicitado el objeto.

Notas:

- Las variables real e imag se han declarado como private. Únicamente pueden acceder a ellas los métodos de la clase. El siguiente código no es válido.

```
void main(void)
{
  CComplejo c1, c4;          // Constructor por defecto
  c1.real = 3.0;
  c1.imag = 4.0; //No puedo acceder porque es un miembro private
}
```

- De esta manera se impide que los datos de la clase se modifiquen en funciones externas a la clase.
- Al escribir la función Visualizar, hemos utilizado el operador de resolución de ámbito (::). Indica que la función es un miembro de la clase CComplejo.

```
float CComplejo::Visualizar(void)
```

5. Sobrecarga de operadores

- Los operadores del lenguaje (+ - * / ++ etc) se pueden redefinir para tratar tipos no estándar.
- Ejemplo: Definimos una nueva clase denominada CComplejo y queremos usar expresiones del tipo a+b, donde a y b son dos complejos

```
// O como un método de la clase
CComplejo CComplejo::operator +(CComplejo &a)
{
   CComplejo suma;
   suma.real = real + a.real;
   suma.imag = imag + a.imag;
   return suma;
}

void main()
{
   CComplejo x(1,1), y(1,1), z;
   z = x + y;
   z.Visualizar();
}
```

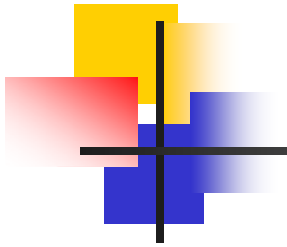
5. Sobrecarga de operadores

- Para entender el funcionamiento, podemos sustituir **operator +** por **Sumar**, resultando en el siguiente método de la clase CComplejo.

```
// O como un método de la clase
CComplejo CComplejo::Sumar(CComplejo &a)
{
  CComplejo suma;
  suma.real = real + a.real;
  suma.imag = imag + a.imag;
  return suma;
}
```

- Ahora queda más claro. El objeto que se ejecuta es ‘x’ y recibe como parámetro a ‘y’. El resultado se asigna a ‘z’.

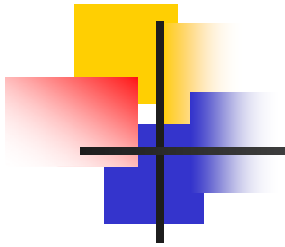
```
void main()
{
  Ccomplejo x(1,1), y(1,1), z;
  z = x.Sumar(y); // z = x + y; //Utilizando operator +
  z.Visualizar();
}
```



6. El puntero this

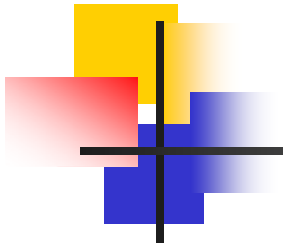
- C++ dispone de un puntero predefinido denominado this
- Es utilizado por los métodos definidos en una clase
- Apunta al objeto desde el cual se llamó a la función (método).
- Por ejemplo, para sobrecargar el operador =

```
CComplejo CComplejo::operator = (const CComplejo &a)
 {
   real = a.real; //real e imag son los datos privados del
   imag = a.imag; //objeto que hizo la llamada

   // this apunta al objeto que hizo la llamada
   // Se devuelve el contenido de la dirección this
   return (*this);
 }
```



Uso de la clase:

```
int main(void)
{
   CComplejo c1,c4;     // Constructor por defecto
   CComplejo c2(3,2), c3(5,5); // Constructor con parámetros
   CComplejo c4(c1);                    //Constructor copia

   c1.Visualizar();
   c2.Visualizar();
   c3.Visualizar();

   c1 = c2 + c3;
   c1.Visualizar();
}
0 + 0i
3 + 2i
5 + 5i

8 + 7i
```



7 Funciones y clases amigas (`friend`)

- Funciones Amigas:
  - Permiten que una función *global* pueda acceder a los miembros de una clase como si fuera un método.
  - Ejemplo: función que compara dos complejos
    - Debe poder acceder a los miembros de la clase pero éstos son privados.
    - Para solucionarlo se puede declarar amiga de la clase CComplejo.

```
int ComplejosIguales(CComplejo &a, CComplejo &b)
{
   if(a.real==b.real && a.imag==b.imag)
      return 1;
   else return 0;
}
```

7 Funciones y clases amigas (`friend`) (2)

- Funciones Amigas (continuación):

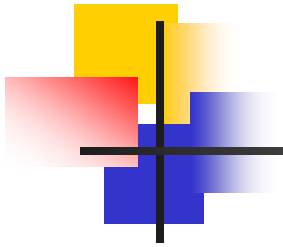
```
class CComplejo {

private:
    float real;        // parte real
    float imag;        // parte imaginaria

public:
    CComplejo(void);                        // constructor por defecto
    CComplejo(float num_real, float num_imag); // constructor a partir de dos reales
    CComplejo(CComplejo &compl_orig);                    // constructor copia
    ~CComplejo(void);                                    // destructor

    void AsignarReal(float num_real);     // Asigna la parte real
    void AsignarImag(float num_imag);  // Asigna la parte imaginaria
    void Visualizar(void);                // Muestra en pantalla el número complejo
    CComplejo& operator = (CComplejo & compl_orig);      // operador asignación
    CComplejo operator + (CComplejo &a);          // operador suma

    friend int ComplejosIguales(CComplejo &a, CComplejo &b);
};  // fin de la clase CComplejo
```

7.2 Funciones y clases amigas (`friend`) (3)

- Clases Amigas:
  - permite que una clase pueda acceder a todos los miembros de otra

```
Class item
{
private:
    int data;
public:

    friend class ConjuntoItems;
}
class ConjuntoItems
{
    ......................
    // puede acceder al miembro data de la clase item
}
```

Entrada / Salida por Consola



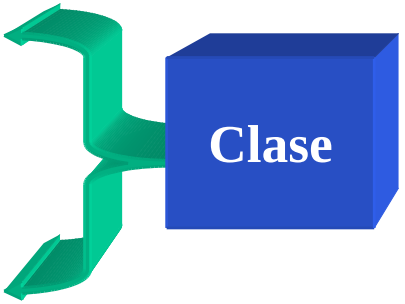
- Librería: <iostream.h>

- Variables (objetos) predefinidos:
  - cout : salida por consola (Objeto de la clase ostream)
  - cin : entrada por consola (Objeto de la clase istream)

- Métodos:
  - operador `<<` (escribir tipos predefinidos en un objeto ostream)
  - operador `>>` (leer tipos predefinidos de un objeto istream)
- Ejemplos:
  - `cout << “Hola, mundo.\n”;`
  - `cin >> valor;`

3. Características de C++

- Encapsulamiento — **Agrupamiento de los datos y funciones que operan sobre ellos. (CAJA NEGRA)**
- Abstracción — **Capacidad para generalizar (propiedades comunes de diferentes objetos)**



- Herencia — **Mecanismo para compartir métodos (funciones) y datos entre clases y objetos**

- Polimorfismo — **Diferentes formas de un mismo método dependiendo de la clase en la que se implemente**

- Sobrecarga de funciones y operadores. — **Mecanismo por el cual una misma función puede comportarse de diferente forma según los parámetros de llamada.**

Ejemplo 2, clase CMatriz

```
class CMatriz
{
    private:
       int fil;
       int col;
       double **mat;
    public:
       CMatriz(int fil=TAM_DEF, int col=TAM_DEF); // constructor con parámetro por defecto
       CMatriz(int fil, int col, double **datos); // constructor con filas, columnas y valores
       CMatriz(CMatriz &orig);//constructor copia
       ~CMatriz(void);

       CMatriz operator + (CMatriz &der);
       CMatriz operator - (CMatriz &der);
       CMatriz operator * (CMatriz &der);

       CMatriz& operator = (const CMatriz &der);

       void Visualizar(void); // visualiza los datos de la matriz
       void IntroducirValores(void);
};
```

Ejemplo 2, clase CMatriz

```
CMatriz::CMatriz(int filas, int columnas)
{
        fil = filas;
        col = columnas;
       //filas
        mat = new double*[fil];

        //columnas
        for(int i=0; i < fil; i++)
                mat[i]= new double[col];

}
```

```
CMatriz::CMatriz(int filas, int columnas, double **datos)
{
        int i, j;
        fil = filas;
        col = columnas;

        //filas
        mat = new double*[fil];

        //columnas
        for(i=0; i < fil; i++)
                mat[i]= new double[col];

        for(i = 0; i < fil; i++)
                for(j=0; j < col; j++)
                        mat[i][j]=datos[i][j];

}
```

Ejemplo 2, clase CMatriz

```
CMatriz::CMatriz(CMatriz& orig)
{
    int i, j;

    fil = orig.fil;
    col = orig.col;
    //filas
    mat = new double*[orig.fil];

    //columnas
    for(i=0; i < orig.fil; i++)
                    mat[i]= new double[orig.col];

    for(i = 0; i < orig.fil; i++)
        for(j=0; j < orig.col; j++)
                    mat[i][j]=orig.mat[i][j];
}
```

```
CMatriz::~CMatriz()
{

        for(int i=0; i < fil; i++)
                delete []mat[i];

        delete [] mat;
}
```

Ejemplo 2, clase CMatriz

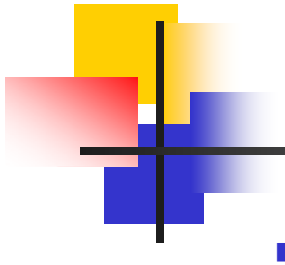
```
void CMatriz::Visualizar()
{
    int i, j;

    cout << endl;

    for(i = 0; i < fil; i++)
    {
        cout << endl;
        for(j=0; j < col; j++)
            cout << mat[i][j]<<"\t";
    }
}
```

Ejemplo 2, clase CMatriz

```
void CMatriz::IntroducirValores()
{
        int i, j;

        cout << endl<< "Introduzca los elementos de la matriz:"<<endl;

        for(i = 0; i < fil; i++)
        {
                for(j=0; j < col; j++){
                        cout << "Introduzca el elemento" << i << "," << j << endl;
                        cin >> mat[i][j];
                }
                cout << endl;
        }
}
```

Ejemplo 2, clase CMatriz

```
CMatriz CMatriz::operator + (CMatriz &der)
{
        CMatriz suma(*this);
        int i, j;

        if((fil!=der.fil)||(col !=der.col))
                cout << endl << "No se pueden sumar las matrices";

        else{

                for(i = 0; i < der.fil; i++)
                        for(j=0; j < der.col; j++)
                                suma.mat[i][j]=mat[i][j]+der.mat[i][j];

        }

        return suma;
}
```



Lectura recomendada

- http://mat21.etsii.upm.es/ayudainf/aprendainf/Cpp/manualcpp.pdf