Bloque práctico 0: *C++*

1: Programación modular y métodos básicos para la creación de clases.

En esta práctica utilizaremos las capacidades de programación orientada a objetos básicas de *C++*. El objetivo de la práctica es programar un ejemplo sencillo de librería y un ejemplo de prueba que lo use. Las clases serán programadas siguiendo una filosofía modular y se compilará una biblioteca de clases estática que luego podremos enlazar a programas nuevos.

0) Recordatorio: la programación modular en *C++*

Para programar una clase como un módulo, debemos generar un archivo de cabecera .hpp con las declaraciones de las clases y un archivo .cpp con las definiciones de métodos e inicialización de variables de clase.

Se utiliza la técnica de definir una constante con el preprocesador en el archivo de cabecera que nos dirá si el archivo ha sido ya incluido con objeto de no volver a incluir las definiciones, ya que darían un error de compilación. Se suele elegir una constante que se parezca al nombre del archivo de cabecera o a la clase que se define.

En un módulo se definirá una única clase o bien clases muy relacionadas. No se debe intentar minimizar el número de archivos fuente sino maximizar la organización y legibilidad de las fuentes.

Ejemplo de clase compilada en un módulo:

Clase1.hpp

```
#ifndef CLASE1HPP
#define CLASE1HPP
 using namespace std;
 class Clase1{
 public:
        void pintar();
  };
#endif
```

Clase1.cpp

```
#include “Clase1.hpp”
#include <iostream>

void Clase1::pintar() {

        std::cout << “Clase1\n”;
}
```

Ejemplo.cpp

```
#include “Clase1.hpp”

int main() {

        Clase1 objeto;
        objeto.pintar();
}
```

Normalmente, diferentes clases relacionadas se empaquetan en bibliotecas (o librerías). La generación de bibliotecas o librerías es muy sencilla:

1. **Con *DevC++*** podemos llevar a cabo esta tarea creando un proyecto de librería estática e incluyendo en el proyecto los archivos de todos los módulos que queremos empaquetar en la biblioteca. La compilación generará un archivo con extensión *.a* en lugar de un ejecutable.

   Una vez generada la libreria *.a* podemos crear un proyecto de consola y linkar la librería en lugar de incluir el código fuente .cpp.

   Para ello, creamos un proyecto de consola, e incluimos en la carpeta del proyecto únicamente los ficheros cabeceras .h, el main.cpp y la librería .a, que habrá que enlazarla al proyecto

Enlace de archivos objeto (*.o*) y bibliotecas (*.a*): Se pueden enlazar archivos con código objeto y bibliotecas a nuestros programas a través de la pestaña *proyecto->opciones de proyecto* (o botón derecho sobre el nombre del proyecto en el navegador y, allí, *opciones de proyecto*), en la solapa de *parámetros*, en el apartado del enlazador (*linker*). Todos los archivos de código objeto (*.o*) y bibliotecas (*.a*) que introduzcamos en *linker* se enlazarán a nuestro código.

2. **Con *Code:Blocks*** podemos llevar a cabo esta tarea creando un proyecto de librería estática (Static library) e incluyendo en el proyecto los archivos de todos los módulos que queremos empaquetar en la biblioteca. La compilación generará un archivo con extensión *.a* en lugar de un ejecutable.

   Una vez generada la libreria *.a* podemos crear un proyecto de consola (Console application) y linkar la librería en lugar de incluir el código fuente .cpp.

   Para ello, creamos un proyecto de consola, e incluimos en la carpeta del proyecto únicamente los ficheros cabeceras .h, el main.cpp y la librería .a (IMPORTANTE: el fichero .a debe copiarse dentro de la carpeta **bin** del proyecto, no en el raíz). La librería .a habrá que enlazarla al proyecto

Enlace de archivos objeto (*.o*) y bibliotecas (*.a*): Se pueden enlazar archivos con código objeto y bibliotecas a nuestros programas a través de la pestaña *Project->Build options* (o botón derecho sobre el nombre del proyecto en el navegador y *Build options*), en la solapa de *Linker settings*, en el apartado del enlazador de librerías (*Link libraries*). Todos los archivos de código objeto (*.o*) y bibliotecas (*.a*) que introduzcamos en *Link libraries* se enlazarán a nuestro código.

Utilización de la biblioteca:

1) Añadir el/los archivos de cabecera (.h o .hpp) necesarios.
2) Añadir el *.a* correspondiente a la lista de archivos a enlazar.

PARTE 1:

Diseñe una clase **complejo** que permita manejar números complejos. Un número complejo (en forma binómica) está formado por 2 enteros a + bi, donde a es la parte real del número complejo y b la parte imaginaria.

Proporcione un único constructor para crear objetos de dicha clase que admita dos valores enteros (**real** e **imaginario**).

Proporcione dos métodos **getr( )** y **geti( )** que permitan consultar los valores de la parte real y de la parte imaginaria, métodos **set( )** que permitan modificar dichos valores (uno que lo haga pidiendo los valores por teclado y otro que acepte 2 valores pasados como parámetro), así como un método **ver( )** que permita ver el número complejo en pantalla en forma binómica.

A modo de ejemplo, los siguientes complejos serán así visualizados:
$z = 2 \Rightarrow 2 + 0i \;\; | z = -2 \Rightarrow -2 + 0i \;\; | z = -2i \Rightarrow 0 - 2i \;\; | z = 1 + 3i \Rightarrow 1 + 3i$

Sobrecargue los operadores + y – unario de forma que se puede operar con complejos directamente, así como sumar un entero con un complejo y un complejo con un entero.

(a + bi) + (c + di) = (a + c) + (b + d)i
r + (a + bi) = (r+a + bi), (a + bi) + r = (r+a + bi)
− (a + bi) = (−a − bi)

Sobrecargue el operador << (operador de inserción de flujo)

Hacer un programa de prueba que permita verificar que todos los métodos y operadores sobrecargados funcionan correctamente.

Restricciones:

a) La sobrecarga de los operadores + y – debemos hacerla utilizando funciones miembros (métodos). Cuando no sea posible usaremos funciones no miembros.

b) El programa debe tener como máximo 1 función amiga (el resto de funciones no miembros que haya que implementar deberán ser no amigas).

c) El programa debe tener un único constructor con 2 parámetros obligatorios (sin argumentos implícitos), es decir, no debe ser posible crear estos objetos complejo:

```
complejo c1, c2(2), c3(2,1); // c1 y c2 deben dar error al no tener 2 parámetros
```

d) Los nombres de la clase y métodos deben ser los indicados en negrita en el enunciado.

PARTE 2:

Salva la clase en una carpeta llamada complejo1 y crea una copia en otra llamada complejo2. Comprueba con el siguiente programa que la clase complejo es correcta (las modificaciones y correcciones que haya que hacer hazla en complejo2):

Prueba1.cpp

```
#include <iostream>   // std::cout, std::fixed
#include <iomanip>    // std::setprecision
#include <cstdlib>    // system
#include <fstream>    // para trabajar con ficheros
#include "complejo.h" // definicion de la clase complejo

using namespace std;

int main(int argc, char *argv[])
{
  complejo a(1,2), b(3,4) ,c(1,-3), d(-1,-2), e(6,2);
  //cout << fixed << setprecision(2); //mostrar 2 (setprecision) decimales (fixed)

  a.set(a.getr()+1,-1*a.geti());
            a.ver(); cout << endl; //a = 2-2i
  cout << "a.real vale " << a.getr() << "\n";  //a.real vale 2
  b=c+d;     b.ver(); cout << endl; //b = 0-5i
  b=5+c+a+2; b.ver(); cout << endl; //b =10-5i
  b=3+c+a;   b.ver(); cout << endl; //b = 6-5i
  b=-3*2+b;  b.ver(); cout << endl; //b = 0-5i
  c=5+c+a+2; c.ver(); cout << endl; //c = 10-5i
  a=-d;      a.ver(); cout << endl; //a = 1+2i
             d.ver(); cout << endl; //d =-1-2i
  c=-c;      c.ver(); cout << endl; //c =-10+5i
  a.set();                          //insertar por teclado el complejo 7-2i
  a.ver();                          //a = 7-2i
  cout << "a vale " << a << "\n";     //a vale 7-2i

  ofstream salida("prueba.txt");
  if (!salida.fail()) {
    salida << "a vale " << a << "\n";
    salida << "b vale " << b << "\n";
    salida << "c vale " << c << "\n";
  }
  salida.close();

  complejo z(1,3), s(1,1);
  cout << s << ", " << z << endl; //1+1i, 1+3i

  z=-z;       cout << "z = " << z << endl; //z = -1-3i
  z=2+z;      cout << "z = " << z << endl; //z = 1-3i
  z=-z;       cout << "z = " << z << endl; //z = -1+3i
  z.set(2,0); cout << "z = " << z << endl; //z = 2+0i
  z=-z;       cout << "z = " << z << endl; //z = -2+0i
  z.set(0,2); cout << "z = " << z << endl; //z = 0+2i
  z=-z;       cout << "z = " << z << endl; //z = 0-2i
  system("PAUSE");  return EXIT_SUCCESS;
}
```

Modifique el programa anterior y declare el complejo d constante:

```
const complejo d(-1,-2);
```

Si la clase complejo está bien diseñada el programa debe seguir funcionando correctamente. Si no es así ¿qué es lo que no habéis tenido en cuenta? ¿Cuántas veces he hablado de esto? Pensad y reflexionad. Corrige los errores encontrados.

Copia todo en una nueva carpeta llamada complejo3 y modifica en dicha carpeta la clase complejo para que el siguiente main( ) pueda ejecutarse:

Prueba2.cpp

```
#include <iostream>   // std::cout, std::fixed
#include <iomanip>    // std::setprecision
#include <cstdlib>    // system
#include "complejo.h" // definicion de la clase complejo

using namespace std;

int main(int argc, char *argv[])
{
  complejo a(1,2), b(3), c(a), e(6,2); //b debe ser  3+0i, c es 1+2i
  const complejo d(-1,-2);
  cout << fixed << setprecision(2); //mostrar 2 (setprecision) decimales (fixed)

  a.set(a.getr()+1,-1*a.geti());
            a.ver(); cout << endl; //a = 2-2i
  b=5+c+a;  b.ver(); cout << endl; //b = 8+0i
  c=5+c+a+2; c.ver(); cout << endl; //c = 10+0i
  c=-c;     c.ver(); cout << endl; //c =-10+0i
  c=d+1;    c.ver(); cout << endl; //c = 0-2i
  c=d+c;    c.ver(); cout << endl; //c =-1-4i


  ++a;      cout << a << endl;     //a = 3-2i
  a++;      cout << b << endl;     //a = 4-2i


  //int r = (int)a;             //r = (int) a → devuelve la parte real de a (4)

  e.set(8,0);                      //e = 8+0i
  if (e==b)
   cout << "e y b son iguales \n";
  else
   cout << "e y b son distintos \n";
  if (e==8)
   cout << e << " es igual a " << 8 << endl;
  system("PAUSE");  return EXIT_SUCCESS;
}
```

Una vez que pruebes que funciona correctamente, modifica el programa anterior añadiendo las siguientes 2 líneas al final (antes del `system("PAUSE");`):

```
  b=++a;
  c=b++;
  cout << a << ", " << b << ", " << c << endl; //a=5-2i, b=6-2i, c=6-2i, c=5-2i
```

¿Funciona el programa?

Las modificaciones de la carpeta complejo3 deben cumplir las siguientes restricciones:

a) Implementar el mínimo número de constructores posible (usa argumentos implícitos).
b) Los operadores + y = deben sobrecargarse el menor número de veces posible, es decir, cuando una sobrecarga no sea estrictamente necesaria (porque el compilador pueda hacer un casting implícito) no se hará la sobrecarga.

Si descomentamos la línea `//int t = (int)a;`

¿Es posible hacer lo indicado en b)? ¿Qué es lo que ocurre? Piensa y reflexiona por qué ahora aparecen errores y antes no.