



Duración	1 sesión.
Fechas de realización	Del 6 al 10 de Marzo.
Fechas de entrega	En el comienzo de la siguiente práctica, medio indicado por el profesor de prácticas.
Objetivos	<ul style="list-style-type: none">■ Poner en práctica la utilización de asertos en la implementación de programas.■ Realizar diversas inmersiones de un programa recursivo.
Trabajo a realizar	Implementar: i) una clase que permita manejar números complejos en coordenadas polares, ii) un programa para el cálculo del producto escalar de dos vectores de complejos de cuatro formas. Contestar las preguntas planteadas.

Índice

1. Clase Complejo	2
2. Cálculo del producto escalar de vectores de complejos	2
3. Cuestiones	3

Referencias

- [1] Plantilla del documento de especificaciones. [[documentacion.doc](#)]
- [2] Aprenda C++ como si estuviera en primero, Javier Garca de Jaln, Jos Ignacio Rodriguez, Jos Mara Sarriegui, y Alfonso Brazlez, Escuela Superior de Ingenieros Industriales de San Sebastin (Universidad de Navarra), 1998. [[manualcpp.pdf](#)]
- [3] Tema 1. [EFICIENCIA DE LOS ALGORITMOS](#). Fernando Barber. 2005.
- [4] Tema 2. [ESPECIFICACIÓN Y CORRECCIÓN DE ALGORITMOS](#). Fernando Barber. 2005.
- [5] Tema 3. [VERIFICACIÓN Y DERIVACIÓN DE ALGORITMOS. SEMÁNTICA AXIOMÁTICA](#). Fernando Barber. 2005.
- [6] Tema 4. [DISEÑO DE ALGORITMOS RECURSIVOS](#). Fernando Barber. 2005.



1. Clase Complejo

En primer lugar se debe crear una clase **Complejo** que permita manejar números complejos en coordenadas polares. En esta notación, un número complejo se representa mediante su módulo y su argumento (la relación entre módulo-argumento y parte real-imaginaria se puede ver en la implementación parcial de la clase que se ofrece). El argumento lo representaremos siempre en radianes y estar comprendido entre 0 y 2π .

Al implementar esta clase se debe incluir también un **invariante de clase**. El invariante de clase es un aserto que se ha de cumplir siempre para cada objeto de la clase, si no se cumple significa que los valores del objeto son incorrectos. Normalmente se comprueba al principio y al final de todos los métodos de la clase que pueden modificar el objeto (métodos no const), aunque en esta práctica sólo lo pondremos al final de los métodos que modifican el objeto, incluidos los constructores.

Únicamente se deben implementar los métodos presentes en el fichero de cabecera (**Complejo.h**). El método para imprimir un complejo en pantalla (**print()**) se puede sustituir por la sobrecarga del operador \ll .

Se debe desarrollar un programa de prueba (**ComplejoTest.cpp**) que compruebe el correcto funcionamiento de todos y cada uno de los métodos de la clase desarrollada.

2. Cálculo del producto escalar de vectores de complejos

Una vez creada la clase y comprobada, se debe realizar un programa (**ProductoEscalar.cpp**) que calcule el producto escalar de dos vectores de complejos de cuatro formas diferentes:

- Mediante una función recursiva lineal no final.
- Mediante una función recursiva final.
- Mediante una función recursiva final con postcondición constante.
- Mediante una función iterativa.

En cada una de las funciones se debe escribir su precondition, su postcondición, un aserto que garantice que la función de cota es un natural y en el caso del bucle además el invariante. Para poder escribir estos asertos es necesario utilizar la función **inner_product** de los algoritmos de la STL. Como ejemplo de uso, la siguiente línea calcula el producto escalar de dos vectores *a* y *b* desde 0 hasta *i*. El cuarto parámetro contiene la inicialización del acumulador del sumatorio, que debe ser el elemento neutro de la suma.



```
inner_product(a.begin(), a.begin() + i, b.begin(), Complejo() )
```

Cada una de las funciones se calculará mediante una transformación a partir de su predecesora y todas las funciones deben incluir también un comentario que indique cual es la función de cota. **Los cálculos en papel se deberán entregar junto con el código de la práctica.**

3. Cuestiones

1. Di parámetros de entrada para cada una de las funciones recursivas que hagan falsas las precondiciones.

2. ¿ Es posible modificar alguna de las 4 funciones para que devuelva un resultado erróneo y que todos los asertos sigan siendo ciertos? ¿ Cómo?



3. ¿ Es posible modificar alguna de las 4 funciones para que entre en un bucle o recursividad infinita y que todos los asertos sigan siendo ciertos? ¿ Cómo?

4. Una vez depurado el programa, los asertos ya no son necesarios. ¿ Cómo se crea un ejecutable que no los incluya?