

Tecnología de la programación

Sesión 17

Objetivos de la sesión

1. Comenzar el tema de TADs (hasta la diapositiva 14)
2. Hacer la especificación, ejemplos de uso y la implementación típica del TAD complejo.
3. Deberes: sumar una secuencia de complejos acabada en $0 + 0i$. Para ello simplemente usaremos la especificación del TAD Números Complejos.

Guion

Sobre este nuevo tema

Es un tema **mucho más sencillo** que los anteriores. El objetivo es definir nuevas estructuras de datos que nos sirvan para modelar cosas de la realidad.

Es interesante hacerlo porque **con los tipos básicos** de los lenguajes de programación (enteros, reales, booleanos, etc) **no tenemos suficiente**. Por ejemplo, ¿cómo hacemos un programa para trabajar con complejos, si no tenemos el tipo adecuado en C++?

Lo que vamos a hacer ahora es **crear nuevos tipos (junto con sus operaciones)**, que vamos a llamar “tipos abstractos de datos” (TAD). Con los TADs, vamos a hacer 3 cosas:

- Especificarlos: decimos qué operaciones va a tener el TAD y sus cabeceras.
- Usarlos: usaremos el TAD basándonos únicamente en la especificación, es decir, sin tener ni idea de la implementación.
- Implementarlos: decidimos una representación del TAD (podría ser basada en punteros, o en vectores, o en registros, o combinaciones de varias cosas, etc) e implementamos las operaciones cumpliendo la especificación.

El segundo punto es muy importante: si os dais cuenta ya estáis usando los tipos de datos básicos (int, float, double, boolean) y sus operaciones básicas (sumar, restar, multiplicar, etc) sin tener ni idea de cómo están implementados por debajo. **Eso es lo importante, que como usuario no necesites saber cómo está implementado para usarlo en tus programas, solo necesitas la especificación y las cabeceras.** De hecho, el objetivo es que un programa que use un TAD, debería seguir funcionando aunque se cambiase la implementación.

Muy importante: es casi seguro que en el examen final se os va a pedir especificar, implementar y usar algún TAD. Así que tenéis que saber qué se hace en cada caso.

Ejemplo: crear un tipo abstracto de datos para modelar los números complejos.

ESPECIFICACIÓN DEL TAD NÚMEROS COMPLEJOS

especificación

TAD Números Complejos
género
complejo

Aquí ponemos el nombre que le vamos a dar al tipo. Es decir, lo que pondremos en las cabeceras para referirnos que es un número complejo.

operaciones

acción crear (Ent/ real a, Ent/ real b, Sal/ complejo z)
{PRE:}
{POST: z es el complejo con parte real a y parte imaginaria b}

función preal(complejo z) devuelve real
{PRE:}
{POST: devuelve la parte real de z}

En **operaciones** ponemos los métodos básicos que tendría el TAD. Hay que poner el nombre del método, su cabecera y su especificación.

función pimag(complejo z) devuelve real
{PRE:}
{POST: devuelve la parte imaginaria de z}

acción sumar(Ent/ complejo z1, Ent/ complejo z2, Sal/ complejo z)
{PRE:}
{POST: z es el complejo z1 + z2}

fin_especificacion

Hay dos detalles importantes:

1. La única dificultad es decidir qué operaciones forman parte de la especificación. Hay que tener cuidado y elegir bien.
2. No devolvemos el tipo de dato complejo por medio de funciones, sino como acciones. Esto se debe a que NO tenemos ninguna garantía de lo que haría la asignación, porque no sabemos **cómo** está implementado. Imaginemos que la operación **sumar** fuese una función que recibiese dos parámetros y devolviese un complejo.

Es decir, que tendríamos que hacer esto para conseguir el resultado:

```
z = sumar(z1,z2)
```

Entonces, si el tipo de dato complejo estuviese implementado usando vectores de dos componentes, tendríamos problemas (no se pueden igualar vectores). Para evitar esto, vamos a poner siempre acciones (salvo cuando el tipo de retorno sea un tipo de dato básico, como enteros, reales o booleanos). Porque nosotros como usuarios NO sabemos cómo está implementado y por tanto no nos podemos basar en la implementación.

A partir de la especificación, podríamos usar ese tipo de dato, sin saber cómo han decidido implementarlo. Es más, es lo que estáis haciendo habitualmente. Por ejemplo, aquí está la especificación de las funciones de <cstring>:
<http://www.cplusplus.com/reference/cstring/>

Entonces, podríamos implementar ya un montón de funciones interesantes acerca del TAD Complejo, incluso una librería completa con algoritmos interesantes basándonos en la especificación para darle más funcionalidad:

EJEMPLOS DE USO DEL TAD NÚMEROS COMPLEJOS

```
acción conjugado(Ent/ complejo z, Sal/ complejo conjz)
{PRE:}
{POST: conjz es el conjugado de z}
principio
    crear(preal(z), -pimag(z), conjz)
fin
```

```
acción restar(Ent/ complejo z1, Ent/ complejo z2, Sal/ complejo z)
{PRE:}
{POST: z es z1-z2}
variables
    z2menos
principio
    crear(-preal(z2), -pimag(z2), z2menos)
    sumar(z1,z2menos,z)
fin
```

```
función módulo(complejo z)
{PRE:}
{POST: devuelve el módulo de z}
principio
    dev (sqrt(preal(z)*preal(z) + pimag(z)*pimag(z)))
fin
```

Repaso de matemáticas: ¿qué es el módulo de un complejo?
https://es.wikipedia.org/wiki/N%C3%BAmero_complejo#Valor_absoluto_o_m%C3%B3dulo_de_un_n%C3%BAmero_complejo

```
función argumento(complejo z)
{PRE:}
{POST: devuelve el argumento de z}
principio
    dev (atan2(pimag(z)/preal(z))
fin
```

Repaso de matemáticas: ¿qué es el argumento de un complejo?
[https://es.wikipedia.org/wiki/Argumento_\(an%C3%A1lisis_complejo\)](https://es.wikipedia.org/wiki/Argumento_(an%C3%A1lisis_complejo))

Aquí es necesario poner **atan2** para que haga el cálculo bien. Es la arcotangente definida para los 4 cuadrantes. **No entra en el examen.**

Pasamos ahora a la implementación. Tenemos que decidir dos cosas: cómo representar el tipo de dato complejo y luego programar las operaciones de la especificación.

Una posible representación del tipo de dato complejo es por medio de un registro con dos cambios de tipo real. Vamos a hacerlo:

IMPLEMENTACIÓN DEL TAD NÚMEROS COMPLEJOS

implementación

TAD Números Complejos

Tipo

```
complejo = registro
  real pr
  real pi
fin
```

Aquí es donde ponemos que lo implementamos usando un registro

Interpretación: pr es la parte real del complejo, pi es la parte imaginaria.

Aquí explicamos la representación. Es muy importante para que no haya confusión.

acción crear (Ent/ real a, Ent/ real b, Sal complejo z)

principio

```
z.pr = a
z.pi = b
```

fin

función preal(complejo z) devuelve real

principio

```
devuelve (z.pr)
```

fin

función pimag(complejo z) devuelve real

principio

```
devuelve (z.pi)
```

fin

acción sumar(Ent/ complejo z1, Ent/ complejo z2, Sal/ complejo z)

principio

```
z.pr = z1.pr + z2.pr
z.pi = z1.pi + z2.pi
```

fin

fin_implementacion

Implementamos las 4 operaciones especificadas usando la representación que hemos elegido

Tenemos que darnos cuenta, que hay otras posibles implementaciones. Por ejemplo, podemos basarnos en vectores de dos componentes:

implementación

TAD Números Complejos

Tipo

```
typedef real= complejo[2]
```

Aquí es donde ponemos que lo implementamos usando un vector de dos elementos.

Interpretación: el vector tendrá en su primera componente la parte real y en su segunda componente la parte imaginaria del complejo.

Aquí explicamos la representación. Es muy importante para que no haya confusión (por ejemplo, ¿dónde guardamos la parte real, en la primera o en la segunda?)

```
acción crear (Ent/ real a, Ent/ real b, Sal complejo z)
principio
    z[0] = a
    z[1] = b
fin
```

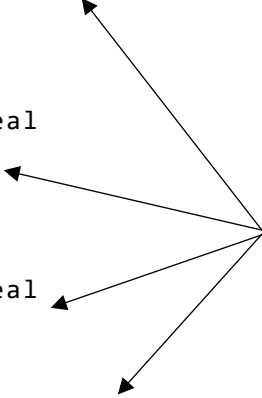
```
función preal(complejo z) devuelve real
principio
    devuelve (z[0])
fin
```

```
función pimag(complejo z) devuelve real
principio
    devuelve (z[1])
fin
```

```
acción sumar(Ent/ complejo z1, Ent/ complejo z2, Sal/ complejo z)
principio
    z[0] = z1[0] + z2[0]
    z[1] = z1[1] + z2[1]
fin
```

fin_implementacion

Implementamos las 4 operaciones especificadas usando la representación que hemos elegido



Si nos damos cuenta, las funciones y acciones que hemos definido como usuarios (modulo, restar, argumento, conjugado) seguirán funcionando, sea cual sea la implementación.

Deberes: sumar una secuencia de complejos acabada en $0 + 0i$. Para ello simplemente usaremos la especificación del TAD Números Complejos.