Tecnología de la programación
Sesión 18

Objetivos de la sesión

1. Recordatorio de que no tenemos la asignación en TADs
2. Corregir el ejercicio propuesto de sumar una secuencia de complejos
3. Operaciones de lectura y escritura en TADs
4. Implementar el TAD Números Complejos usando módulo y argumento
5. Acabar las diapositivas del tema

Guion

RECORDATORIO:

No podemos hacer asignaciones cuando estemos trabajando con TADs, porque no tenemos la garantía de cómo están implementados. Si un TAD estuviese implementado por vectores o punteros, podríamos tener comportamientos inesperados si realizamos asignaciones (estaríamos intentando hacer copias de vectores o teniendo dos punteros apuntando a lo mismo).

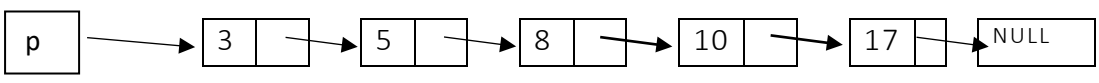
Por tanto, no vamos a tener funciones que devuelvan algo de tipo complejo, sino que para "devolver" algo, lo haremos a través de parámetros de salida en acciones.

¿Y CÓMO COPIAMOS TADs?

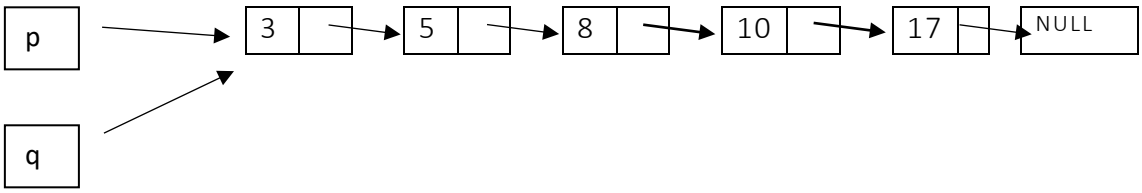
Muchas veces, necesitamos hacer "copias" de las variables de nuestros TADs. Por ejemplo, imaginemos que hemos creado un TAD Lista, donde hemos definido listas "infinitas" (en el sentido de ilimitadas, pero tendrán que tener fin, es decir, que cada lista tenga un número finito de elementos). Podemos implementar estas listas mediante una lista enlazada de nodos (usando punteros y nodos, de hecho es exactamente la representación que utilizábamos en el tema anterior).

Por tanto, podríamos tener una variable `p` que sea de tipo `lista`, donde `p` en el fondo no es más que un puntero que apunta al primer Nodo.

```
Lista p
```



Si nos creásemos otra variable q de tipo Lista e hiciésemos la asignación q = p, tendríamos lo siguiente:

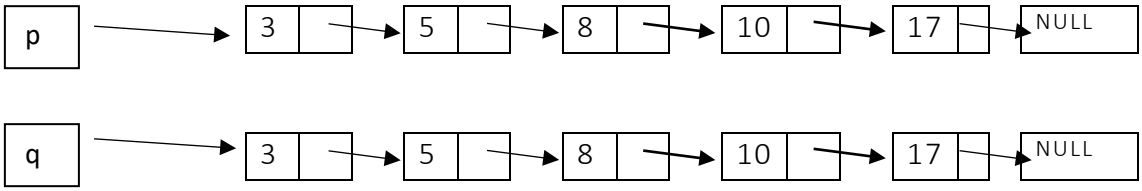


Como veis, no hemos hecho una copia, si no que tendríamos realmente la misma lista apuntada desde dos sitios. Por eso NO tenemos que hacer asignaciones con TADs, y siempre usaremos acciones.

Para hacer lo que queremos, tendríamos que hacer una acción que se encargase de copiar la Lista:

```
acción copiarLista(Ent/ Lista p, Sal/ Lista q)
{PRE: La lista p está iniciada}
{POST: La lista q contiene exactamente los elementos de p en el mismo orden.
La lista p no se modifica.}
```

De forma que lo que consiguiésemos sea lo siguiente (ya veremos en el próximo tema cómo hacer esa acción en concreto para un TAD Lista):



Con esta acción hay dos posibilidades:

1. Incluirla en la especificación
2. No incluirla, pero entonces nos la implementaremos nosotros (como usuarios, sin basarnos en la representación).

Debido a que es una operación que usaremos con frecuencia, a partir de ahora nosotros la incluiremos en la especificación del TAD (ojo con los estudiantes de años anteriores: antes no la incluíamos y teníais que implementarla vosotros como usuarios).

Para el caso concreto del TAD Números Complejos, entonces tendríamos la siguiente acción (siguiendo la opción 1):

```
acción copiarComplejo(Ent/ complejo z1, Sal/ Complejo z2)
{PRE: }
{POST: z2 es una copia de z1}
Principio
    z2.pr = z1.pr
    z2.pi = z2.pi
fin
```

En caso de que no estuviese en la especificación y nos la tuviésemos que implementar como usuarios (opción 2), tendríamos lo siguiente:

```
acción copiarComplejo(Ent/ complejo z1, Sal/ Complejo z2)
{PRE: }
{POST: z2 es una copia de z1}
Principio
    crear(preal(z1), pimag(z1), z2)
fin
```

CORRECCIÓN DEL EJERCICIO DE SUMAR UNA SECUENCIA DE COMPLEJOS:

Una posible solución sería la siguiente. Le pedimos al usuario que introduzca los complejos, haciendo que vaya metiendo valores de dos en dos (parte real y parte imaginaria), para acabar al llegar al 0 0.

```
Algoritmo SumarComplejos
Variables
    complejo x, resultado
    real a, b
principio
    crear(0,0,resultado)
    escribir("Introduce una secuencia de complejos acabada en 0 0")
    leer(a)
    leer(b)
    mientras que a!=0 OR b !=0 hacer
        crear(a,b,x)
        sumar(resultado, x, resultado)
        leer(a)
        leer(b)
    fmq
    escribir(preal(resultado) + "+" + pimag(resultado) + "i")
fin
```

LECTURA Y ESCRITURA EN TADs

Si nos damos cuenta, no solo tenemos que tener cuidado al leer (no podemos leer algo de tipo complejo porque no tenemos esa operación en el TAD), sino que también tenemos que tener cuidado al escribir: no podemos hacer `escribir(resultado)` porque el TAD Números Complejos tampoco tiene esa operación.

Por tanto, nos hemos dado cuenta de que hay otras dos operaciones que generalmente serán interesantes en los TAD: la lectura y la escritura.

De nuevo estamos con el mismo caso que al copiar, tenemos dos posibilidades. Incluirlas en la especificación, o de no hacerlo, implementarlas como usuarios. Tenéis que estar preparados para encontraros con ambas posibilidades, porque su inclusión o no va a depender de la persona que diseñe el TAD y del propio TAD en sí. Nosotros intentaremos incluirlas siempre (pero cuidado, hay TADs donde no las incluiremos porque no tiene sentido o porque haya varias formas de leer o escribir, como en el TAD Árbol Binario).

Para el TAD Números complejos, tendríamos la siguiente implementación (en caso de que las hayamos incluido en la especificación):

```
acción leerComplejo(Sal/ complejo z)
principio
    escribir("Introduce la parte real")
    leer(z.pr)
    escribir("Introduce la parte imaginaria")
    leer(z.pi)
fin

/*Hacemos el condicional para evitar que muestre + - cuando la parte imaginaria
sea negativa*/

acción escribirComplejo(Ent/ complejo z).
principio
    si (z.pi > 0)
        escribir(z.pr + "+" + z.pi + "i")
    si_no
        escribir(z.pr + "-" + (-1 * z.pi) + "i")
    fsi
fin
```

Si no están incluidas en la especificación, las tendríamos que implementar como usuarios:

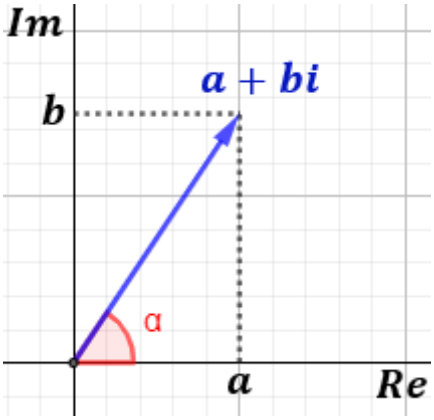
```
acción leerComplejo(Sal/ complejo z)
variables
    real a, b
principio
    escribir("Introduce la parte real")
    leer(a)
    escribir("Introduce la parte imaginaria")
    leer(b)
    crear(a,b,z)
fin

/*Hacemos el condicional para evitar que muestre + - cuando la parte imaginaria
sea negativa*/

acción escribirComplejo(Ent/ complejo z).
principio
    si (pimag(z) > 0)
        escribir(preal(z) + "+" + pimag(z) + "i")
    si_no
        escribir(preal(z) + "-" + (-1 * pimag(z)) + "i")
    fsi
fin
```

OTRA IMPLEMENTACIÓN DEL TAD NÚMEROS COMPLEJOS

Vamos a ver otra posible implementación del TAD, ahora usando módulo y argumento.



La longitud (la flecha azul) es el módulo y el ángulo (el α en rojo) el argumento.

Vamos a suponer que se han incluido las operaciones de copiar, lectura y escritura en la especificación, y por tanto las implementaremos también.

```
implementación
    TAD Números Complejos
Tipo
    complejo = registro
        real mod
        real arg
    fin
```

Aquí es donde ponemos que cómo lo implementamos (en este caso también usando un registro de dos campos)

```
Interpretación: mod es el módulo del complejo, arg el argumento del complejo
```

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.mod = sqrt(a*a+b*b)
    z.pi = atan2(b/a)
fin

función preal(complejo z) devuelve real
principio
    devuelve (z.mod * cos(z.arg))
fin

función pimag(complejo z) devuelve real
principio
    devuelve (z.mod * sin(z.arg))
fin

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

```
variables
    real auxr, auxi
principio
    auxr = z1.mod*cos(z1.arg) + z2.mod * cos(z2.arg)
    auxi = z1.mod*sin(z1.arg) + z2.mod * sin(z2.arg)
    z.mod = sqrt(auxr*auxr+auxi*auxi)
    z.arg = atan2 (auxi/auxr)
fin

acción leerComplejo(Sal/ complejo z)
variables
    real a, b
principio
    escribir("Introduce la parte real")
    leer(a)
    escribir("Introduce la parte imaginaria")
    leer(b)
    z.mod = sqrt(a*a+b*b)
    z.arg = atan2(b/a)
fin

acción escribirComplejo(Ent/ complejo z).
variables
    real pr, pi
principio
    pr = z.mod * cos(z.arg)
    pi = z.mod * sin (z.arg)
    si (pi > 0)
        escribir(pr + "+" + pi + "i")
    si_no
        escribir(pr + "-" + (-1 * pi) + "i")
    fsi
fin

acción copiarComplejo(Ent/ complejo z1, Sal/ complejo z2).
principio
    z2.mod = z1.mod
    z2.arg = z2.arg
fin

fin_implementacion
```

DIAPOSITIVAS HASTA EL FINAL DEL TEMA:

Son sencillas. Para entender el diseño modular, se puede ver con un ejemplo:

Ejemplo 1: Gestión de un banco.

- Con diseño descendente, tendríamos las operaciones listar, dar de alta, realizar movimiento…
- Con diseño modular, tendríamos los TADs: Cuenta, Cliente, Movimiento…

Ejemplo 2: partida de cartas

- Diseño descendente: jugar, barajar, apostar
- Diseño modular: TAD baraja, TAD Carta, TAD Partida,...