TEMA 3

Diseño de algoritmos.

T
E
M
A

3

Contenido del Tema

3.1.- Programación Modular y desarrollo de

3.2.- Diseño orientado a la abstracción de datos.

Programación Modular

No usar una metodología de diseño conlleva:

- Rigidez e inflexibilidad en los programas
- Pérdida excesiva de tiempo en corrección de errores
- Documentación insuficiente o nula
-

Diseño descendente.

- **Módulo** es un algoritmo autocontenido, que puede ser diseñado independientemente del contexto en el que va a ser usado

Programación Modular

Ventajas:

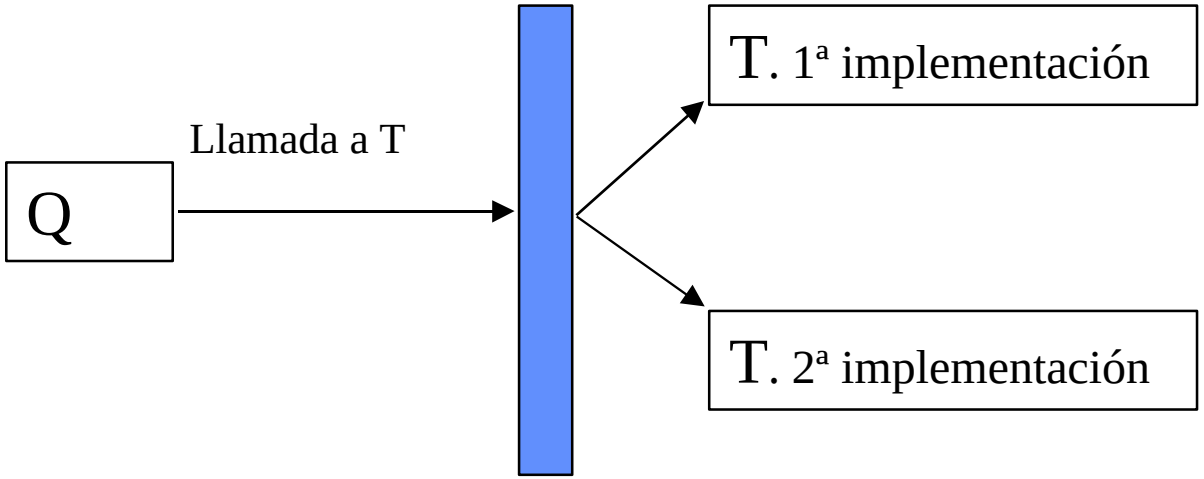
- Simplificación del diseño.
- Programación aislada (encapsulamiento).
- Conocimiento de lo **que** hace el subprograma, no **cómo** lo hace (Abstracción procedimental o de Operaciones).
- Reutilización del módulo en otro contexto.
- Simplificación de la comprensión del algoritmo

- Compilación separada.

Programación Modular

Encapsulamiento:

- Si el método para solucionar una tarea T cambia, cualquier otra tarea Q no se ve afectada.



- Ejemplo: Cambio del algoritmo de ordenación.

Programación Modular

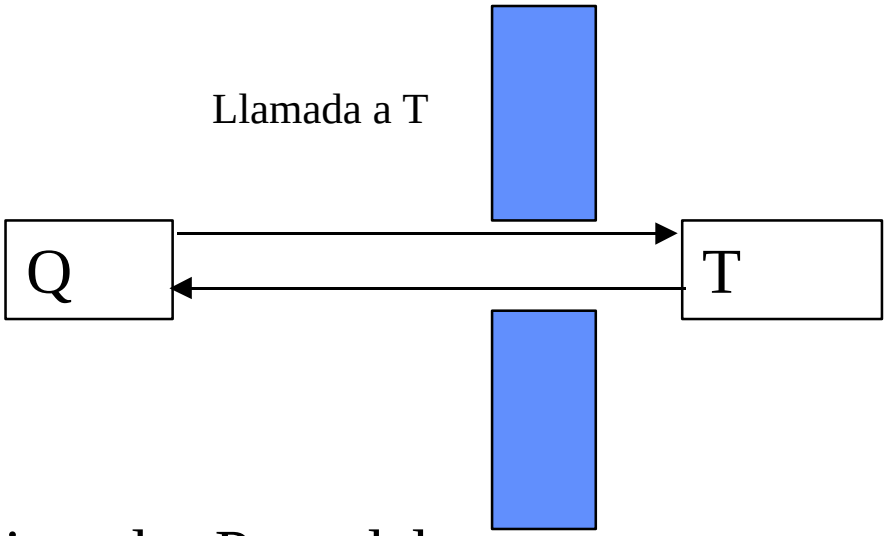
Especificaciones:

- Conjunto de condiciones que deben cumplirse para que se ejecute un módulo correctamente.

Q debe saber:

-Lo que hace T.

-En que condiciones lo ejecuta.



- Ejemplo: Paso del

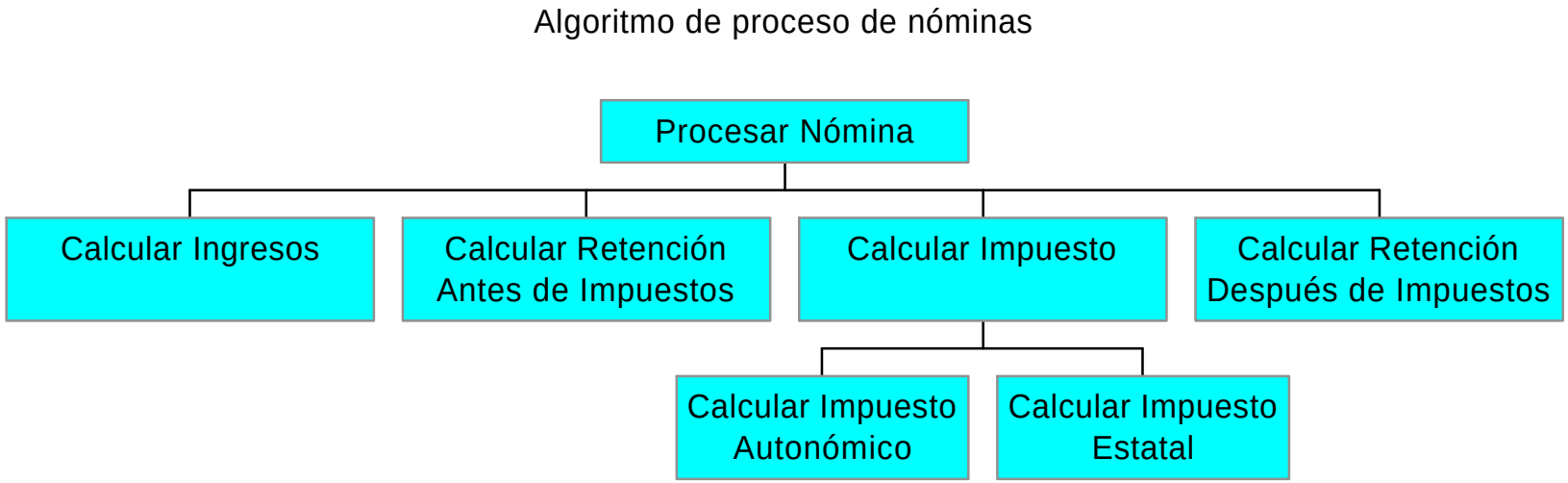
Programación Modular

Criterios de Modularización.

- No existe un algoritmo formal para descomponer un problema en módulos.
- Criterios a seguir:
  - Minimizar el acoplamiento (los módulos deben ser lo más independientes posible).
  - Maximizar la cohesión (relación entre las diferentes partes internas de un módulo).

Programación Modular

Ejemplo:



Programación Modular

Módulos.

- Comenzarán con la palabra **`Módulo`** seguida del nombre del módulo.
- Tanto el programa principal como los módulos podrán importar de otros módulos.
- Parte de Definición. Visible a los algoritmos que la

- Parte de Implementación. Local al módulo.
- En la parte de Implementación ***puede*** haber un cuerpo principal que se ejecutará antes del cuerpo principal del módulo o algoritmo que lo importe.


```
Módulo Coversión  (* Módulo para pasar de grados a radianes y viceversa *)
Definición
    Algoritmo GradosARadianes (grados : R)  :  R  (* Devuelve el
      ángulo en radianes*)
    Algoritmo RadianesAGrados (radianes : R )  :  R (* Devuelve el
      ángulo en grados *)

Implementación
    Constantes
      Pi = 3.1416
    Algoritmo GradosARadianes (grados : R)  : R  (* Devuelve el
      ángulo en radianes*)
    Inicio
      DEVOLVER grados * 2.0 * Pi / 360.0
    Fin
    Algoritmo RadianesAGrados (radianes : R ) : R  (* Devuelve el
      ángulo en grados *)
    Inicio
      DEVOLVER 360.0 * radianes / (2.0 * Pi)
    Fin
Fin   (* Conversión *)
```


```
Algoritmo TercerLado (* Calcula el tercer lado de un triángulo
   a partir de los otros dos y del ángulo entre ellos *)
Desde Conversión Importa GradosARadianes
Variables
    lado1, lado2, lado3, ánguloGrad, ánguloRad : R
Inicio
   Escribir (“Introduce la longitud de un lado”)

   Escribir (“Introduce la longitud del otro lado”)

   Escribir (“Introduce el ángulo (en grados) “)
                    )
            :=                  (          )
                ( lado1* lado1 + lado2 * lado2 - 2.0 * lado1 *
                          (
   coseno del ángulo *)
   Escribir (“La longitud del tercer lado es “, lado3)
Fin   (* TercerLado *)
```

Programación Modular

En un módulo se puede declarar:

- **Constantes**. Se pueden declarar en la parte de:
  - Definición. Cualquier Módulo o programa la podrá importar y utilizar.
  - Implementación. Local al Módulo. No es importable. (Las constantes declaradas en la parte de Definición son visibles en la parte de

  - **Variables**. Se pueden declarar en la parte de:
  - Definición. Cualquier Módulo o programa la podrá importar y utilizar. Solo es recomendable en casos fuera del ámbito de la asignatura.
  - Implementación. Local al Módulo. No es importable. Actúa como una variable global dentro del módulo. No es aconsejable. (Las variables declaradas en la parte de Definición son visibles en la parte de Implementación como variables globales).

Programación Modular

En un módulo se puede declarar:

- **Algoritmos**
  - Definición. En esta parte solo se declara la cabecera del algoritmo. Estos algoritmos podrán ser importados y llamados por cualquier otro módulo o programa. Se deben especificar todos los parámetros y, en el caso de una

  - Implementación. Por cada cabecera declarada en la parte de definición, se correspondiente. Su cabecera debe coincidir en el número y tipo de los parámetros. Además, se pueden declarar otros algoritmos, pero estos no podrán ser importados por otros módulos ni Estos algoritmos pueden servir para descomponer los algoritmos exportables).

En un módulo se puede declarar:

- **Tipos**. Se pueden declarar de dos formas distintas:
  - *Tipos Transparentes:*
    - Definición. Se define el tipo de la misma forma que se ha visto hasta ahora. Cualquier Módulo puede importarlos y utilizarlos como si

    - Implementación. Los tipos declarados en la parte de definición son visibles en la de implementación, por tanto no se pueden volver a

    - ***Tipos Opacos****:*
    - Definición. **No** se describe como están implementados. Simplemente se enumeran los tipos que se quieren exportar.
    - Implementación. Cada tipo que se quiere exportar tiene que ser declarado del tipo PUNTERO. El tipo de la variable puede ser cualquiera.
    - Los programas y módulos que importen un tipo opaco, solo pueden declarar variables de ese tipo, pero no acceder a su estructura. Para ello tendrán que usar los procedimientos que ofrezca el módulo que

    - Son especialmente útiles a la hora de implementar la abstracción de

Abstracción de Datos

PROGRAMA = ALGORITMO + ESTRUCTURAS

. Qué ejecuta y no cómo.

Abstracción de Datos. Qué se hace sobre una estructura y no cómo esta construida.

Abstracción de Datos

Para definir un Tipo Abstracto de Datos habrá que dar los siguientes conceptos:

Definición. Idea (intuitiva o formal) del dato a tratar.

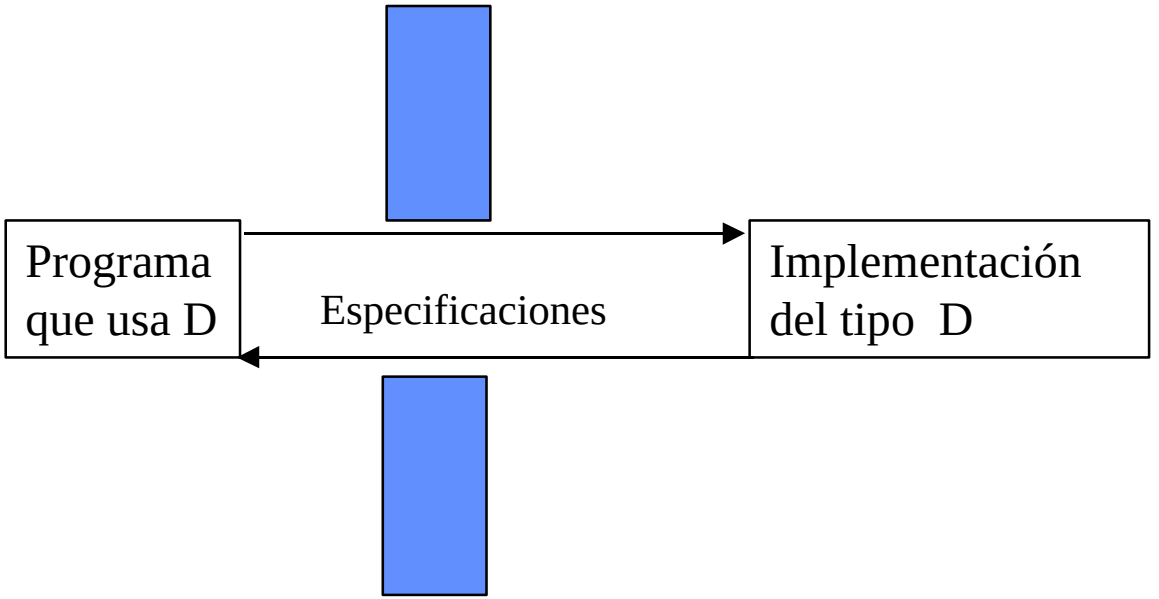
Operaciones. Conjunto de operaciones que se pueden realizar sobre el tipo.

Implementación. Implementación del tipo y de sus

Con la definición y las operaciones sobre el tipo se podrán declarar variables del tipo, las cuales se utilizarán mediante sus operaciones.

Abstracción de Datos

- Con esta técnica se separa el Nivel de utilización del Nivel de Implementación.
- Al igual que en la Abstracción necesarias las especificaciones



Abstracción de Datos

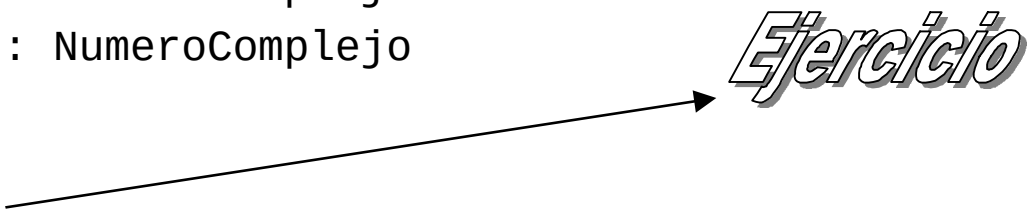
- Ejemplo: Tipo de Datos Complejo.
- **Definición:** Tipo de datos capaz de guardar la información de un numero complejo. ***NumeroComplejo***.
- **Operaciones:**
  - **Asignación:** Toma la parte real y la imaginaria y devuelve un número complejo.
    `Algoritmo Asignación(r,i:R) : NumeroComplejo`
  - **Real:** Toma un número complejo y devuelve su parte real.
    `Algoritmo Real(c : NumeroComplejo) : R`
  - **Imaginaria:** Toma un Numero Complejo y devuelve su parte imaginaria.
    `Algoritmo Imaginaria(c : NumeroComplejo) : R`
  - **Suma:** Toma dos Números Complejos y devuelve su suma.
    `Algoritmo Suma(c1, c2 : NumeroComplejo) : NumeroComplejo`
  - **Producto:** Toma dos Números Complejos y devuelve su producto.
    `Algoritmo Producto(c1, c2 : NumeroComplejo): NumeroComplejo`

Abstracción de Datos

- Nivel de Utilización: Basta con la Definición del Tipo y de sus Operaciones para poder trabajar con él. (No necesitamos saber cómo esta implementado)

```
Algoritmo SumaComplejos (* Lee y suma dos números complejos *)
Desde Complejos Importa NumeroComplejo, Asignación, Suma, Real,
   Imaginaria
Variables
    c1, c2, resultado : NumeroComplejo
    real, imag : R
Inicio
   Escribir (“Introduzca la parte real y la imaginaria del

   Leer (real); Leer(     )
   c1 :=  Asignación (real,     )
   Escribir (“Introduzca la parte real y la imaginaria del

   Leer (real); Leer(     )
   c2 :=  Asignación (real,     )
   resultado := Suma (c1, c2)
   Escribir (“El resultado es: “)
   Escribir (Real (resultado), “+ “, Imaginaria (Resultado),

Fin
```

```
Algoritmo ArrayComplejos (*Multiplica una array de complejos *)
Desde Complejos Importa NumeroComplejo, Asignación, Producto, Real,

Constantes
   TamañoVector = 1000
Tipos
   VectorComplejos = ARRAY [1..TamañoVector] DE NumeroComplejo
Variables
   vComplejos: VectorComplejos
   resultado : NumeroComplejo
   índice : N
Inicio
   Leer_Vector (vComplejos) (* Lee el vector de Complejos *)
   resultado := Asignación (1.0, 0.0)
   PARA índice := 1 HASTA TamañoVector HACER
      resultado := Producto (resultado,
   FINPARA
   Escribir (“El resultado es: “)
   Escribir (Real (resultado), “+ “, Imaginaria (Resultado), “i”)
Fin
```



Abstracción de Datos

- Nivel de Implementación.
  - Definimos un módulo donde se implementa el tipo abstracto y sus operaciones.
  - El tipo podrá ser transparente u opaco.

```
Módulo Complejos (* Implementación Transparente del tipo NumeroComplejo *)
Definición
Tipos
        NumeroComplejo = REGISTRO
                parte_real, parte_imag : R
        FINNREGISTRO
        Algoritmo Asignación(r,i:R) : NumeroComplejo (* Toma la parte real y la imaginaria y
                                                        devuelve un número complejo. *)
        Algoritmo Real(c : NumeroComplejo): R (* Toma un número complejo y devuelve su parte real. *)
        Algoritmo Imaginaria(c : NumeroComplejo): R(* Toma un Numero Complejo y devuelve su
    parte imaginaria.*)
        Algoritmo Suma(c1, c2 : NumeroComplejo) : NumeroComplejo (* Toma dos Números
    Complejos y devuelve su suma *)
        Algoritmo Producto(c1, c2 : NumeroComplejo): NumeroComplejo (* Toma dos Números
    Complejos y devuelve su producto. *)

Implementación
        Algoritmo Asignación(r,i:R) : NumeroComplejo
        Variables
                c : NumeroComplejo
        Inicio
                c.parte_real := r
                c.parte_imag := i
        Fin

       Algoritmo Real(c : NumeroComplejo): R (* Toma un número complejo y devuelve su parte real. *)
        Inicio
                DEVOLVER c.parte_real
        Fin
```

```
        Imaginaria(c : NumeroComplejo): R (* Toma un número complejo y devuelve su
                                                parte imaginaria. *)
    Inicio
        DEVOLVER c.parte_imag
    Fin

    Algoritmo Suma(c1, c2 : NumeroComplejo) : NumeroComplejo (* Toma dos Números
                                                Complejos y devuelve su suma *)
    Variables
        resultado : NumeroComplejo
    Inicio
        resultado. parte_real := c1.parte_real + c2.parte_real
        resultado. parte_
        DEVOLVER resultado
    Fin

    Algoritmo Producto(c1, c2 : NumeroComplejo): NumeroComplejo (* Toma dos Números
                                                Complejos y devuelve su producto. *)
    Variables
        resultado : NumeroComplejo
    Inicio
        resultado. parte_real := c1.parte_real * c2.parte_real -

        resultado. parte_     := c1.parte_real * c2.parte_     +


        DEVOLVER resultado
    Fin

Fin (* Módulo Complejos *)
```

Abstracción de Datos

- La implementación se separa de la utilización.
- La implementación del tipo se puede hacer de muchas

  - `NumeroComplejo = ARRAY [1..2 ] DE R`
  - `NumeroComplejo = REGISTRO`

    ```
        Módulo, Argumento : R
    FINREGISTRO
    ```

- Un cambio en la implementación no debe suponer cambios en la utilización, sin embargo, con la implementación del ejemplo se puede realizar a nivel de utilización:
- `c.parte_real = 5.0`

```
Módulo Complejos  (* Implementación del tipo NumeroComplejo como tipo Opaco*)
Definición
Tipos
        NumeroComplejo (* No decimos como esta implementado *)
        Algoritmo Asignación(r,i:R) : NumeroComplejo (* Toma la parte real y la imaginaria y
                                                        devuelve un número complejo. *)
        Algoritmo Real(c : NumeroComplejo): R (* Toma un número complejo y devuelve su parte real. *)
        Algoritmo Imaginaria(c : NumeroComplejo): R  (* Toma un Numero Complejo y devuelve su
                                                        parte imaginaria.*)
        Algoritmo Suma(c1, c2 : NumeroComplejo) : NumeroComplejo (* Toma dos Números
                                                        Complejos y devuelve su suma *)
        Algoritmo Producto(c1, c2 : NumeroComplejo): NumeroComplejo (* Toma dos Números
                                                        Complejos y devuelve su producto. *)

Implementación
Tipos
        NumeroComplejo = PUNTERO A Reg   (* Siempre puntero *)
        Reg = REGISTRO
                parte_real, parte_imag : R
        FINREGISTRO

        Algoritmo Asignación(r,i:R) : NumeroComplejo
        Variables
                c : NumeroComplejo
        Inicio
                Asignar (c, Tamaño (Reg))
                c^.parte_real := r
                c^.parte_imag := i
        Fin
```

```
Algoritmo Real(c : NumeroComplejo): R
     (* Toma un número complejo y devuelve su parte real. *)
Inicio
     DEVOLVER c^.parte_real
Fin

Algoritmo Imaginaria(c:NumeroComplejo): R
     (* Toma un número complejo y devuelve su parte imaginaria. *)
Inicio
     DEVOLVER c^.parte_imag
Fin

Algoritmo Suma(c1, c2 : NumeroComplejo) : NumeroComplejo
     (*  Toma dos Números      Complejos y devuelve su suma *)
Variables
     resultado : NumeroComplejo
Inicio
     Asignar (resultado, Tamaño (Reg))
     resultado^. parte_real :=  c1^.parte_real + c2^.parte_real
     resultado^. parte_
     DEVOLVER resultado
Fin
```

```
Algoritmo Producto(c1, c2 : NumeroComplejo ): NumeroComplejo
(*  Toma dos Números Complejos y devuelve su producto. *)
 Variables
    resultado : NumeroComplejo
 Inicio
   Asignar (resultado, Tamaño (Reg))
   resultado^.parte_real := c1^.parte_real * c2^.parte_real -

   resultado^.parte_     := c1^.parte_real * c2^.parte_     +
                                          * c2^.parte_real

   DEVOLVER resultado
 Fin

Fin (* Módulo Complejos *)
```

Abstracción de Datos

- El cambio de la implementación no afecta al nivel

- Impide que a nivel de utilización se acceda a la estructura interna del tipo.
- *Ejercicio*: Realizar la implementación opaca de como un puntero a registro donde se guarda módulo y argumento. El algoritmo de multiplicación del vector de complejos será más eficiente sin haber sido

Bibliografía

- ***MODULA - 2. Desarrollo de Software***. C. Galán. Paraninfo. 1987
- ***Algoritmos + Estructuras de datos = Programas.*** Wirth N Ed. Ed del Castillo.
- ***Fundamentos de programación. Algoritmos y estructuras de datos***. L. Joyanes. McGraw-Hill. 1996
- ***PASCAL y Estructuras de datos***. N. Dale & S. Lilly. McGraw-Hill. 1992
- ***Walls and mirrors. Modula-2 edition.*** Hellman, Veroff.. Ed. Benjamin/Cummings Series. 1988.