# E.T.S.I. Técnicos en Informática de Sistemas
# Laboratorio de Programación II. 1º B
## Práctica 3: Módulos Curso 2000/2001

En esta práctica se pide implementar el concepto de número racional mediante el uso de módulos. Para ello, se definirá el tipo RACIONAL y una serie de operaciones a poder realizar sobre dicho tipo. La práctica se grupos de tres personas de manera que cada una de ellas realizará una parte de la práctica. Sea el siguiente módulo de definición (o sea, la especificación) para el tipo racional.

```
DEFINITION MODULE Racional;
    TYPE RACIONAL;              (* Tipo Opaco *)

    PROCEDURE Asignar(VAR r:RACIONAL;n,d:INTEGER);
       (* Asigna a r un numerador y un denominador que provienen de  la normalización de
          n/d y donde el signo se incluirá en el numerador. OJO! Si la implementación es con punteros
          -tipo opaco- debe  "asignar" la memoria  necesaria a r. Si ya la tenía, simplemente machaca
          el numerador y el denominador *)
    PROCEDURE Numerador(r:RACIONAL):INTEGER;
       (* Devuelve el valor del numerador de r *)
    PROCEDURE Denominador(r:RACIONAL):INTEGER;
       (* Devuelve el valor del denominador de r *)
    PROCEDURE LiberarMemoria(VAR r:RACIONAL);
       (* Este procedimiento libera la memoria, si es posible, asociada al racional *)
END Racional.
```

**(PERSONA 1)** Debe codificar **DOS** módulos de implementación para el módulo de definición anterior donde el tipo RACIONAL es opaco en ambos casos. En un módulo se declara como

```
    TYPE RACIONAL = POINTER TO NUMERO;
        NUMERO = RECORD
                    Numerador,Denominador:INTEGER;
        END;
```

Y en el otro como

```
    TYPE RACIONAL = POINTER TO NUMERO;
       NUMERO = ARRAY [1..2] OF INTEGER;   (* Numerador en pos.1 y denominador en la 2*)
```

Si hubo algún error el procedimiento correspondiente deberá visualizar el mensaje adecuado por pantalla. Por ejemplo, pedir el numerador de un número racional que nunca fue asignado o por ejemplo asignar un 0 al denominador de un racional.

**(PERSONA 2)** Considera la siguiente "especificación" (o sea, módulo de definición) para la manipulación de los racionales:

```
    DEFINITION MODULE BibRacional;
        FROM Racional IMPORT RACIONAL;
        PROCEDURE Copiar(r1:RACIONAL;VAR r:RACIONAL);
        PROCEDURE Sumar(r1,r2:RACIONAL;VAR r:RACIONAL);
        PROCEDURE Restar(r1,r2:RACIONAL;VAR r:RACIONAL);
        PROCEDURE Multiplicar(r1,r2:RACIONAL;VAR r:RACIONAL);
        PROCEDURE Dividir(r1,r2:RACIONAL;VAR r:RACIONAL);
        PROCEDURE EsIgual(r1,r2:RACIONAL):BOOLEAN;
    END BibRacional.
```

La persona 2 debe implementar el módulo de Implementación de esta biblioteca. Observa que no se tiene acceso al tipo opaco y que sólo se pueden usar las operaciones que nos proporciona el tipo opaco.

**(Persona 3)** Debe construir un menú para ejecutar todas las operaciones, tanto las realizadas directamente sobre el racional como aquellas que involucran operaciones de su biblioteca. Debe haber también una opción para visualizar los resultados adecuadamente. Observa que habrá que importar el tipo RACIONAL y todas las operaciones necesarias de los módulos Racional y BibRacional.

Una vez codificados todos los módulos (y compilados separadamente) se debe ejecutar el menú (por lo que pondremos todos los módulos en una misma máquina) y ver si se ejecuta todo bien. Una vez que funcione para la primera implementación que hizo la persona 1, cambiaremos el módulo de implementación (OJO! sólo el módulo de implementación) por la segunda de las implementaciones y se debe comprobar que todo

**Practicas Complementarias:**

**Segunda parte de la práctica**: Modificar los módulos anteriores para que el error se propague y no se realicen las operaciones de E/S dentro de los módulos Racional y BibRacional.
Nota: Esto se hace añadiendo variables de error a las cabeceras de los módulos de definición, luego la personas 1 y 2 deben modificar estos módulos en este sentido y la persona 3 debe modificar el programa principal en este sentido.

**Tercera parte de la práctica**: Hacer el tipo Racional Transparente y ver cómo afecta a las ejecuciones.

## Otras nociones teóricas

La llamada

Available(SIZE(**Tipo**))

devuelve TRUE si hay espacio en memoria para almacenar una variable de tipo Tipo y en otro caso devuelve FALSE.
La función Available hay que importarla de la librería Storage.

Es útil a la hora de asignar memoria mediante NEW pues evita errores. Por ejemplo:

```
IF NOT Available(SIZE(NUMERO )) THEN  (* Si no hay espacio para una variable de tipo NUMERO *)
        Error:=TRUE;    (* Error es una variable Booleana *)
ELSE
         NEW(r);                          (* r es de tipo RACIONAL *)
         r^.Numerador:=.....;             (* r^ existe y puedo referenciarlo *)
END;
```

Nota: Tipo no de ser una variiable tipo puntero sino el tipo de la variable apuntada por una variable de tipo puntero o en su caso la variable anónima referenciada por el puntero, o sea, r^. Por ello el código anterior y el siguiente son el "mismo"

```
IF NOT Available(SIZE(r^)) THEN  (* Si no hay espacio para lo que "apunta" la variable r *)
        Error:=TRUE;
ELSE
         NEW(r);                          (* r es de tipo RACIONAL *)
         r^.Numerador:=.....;             (* r^ existe y puedo referenciarlo *)
END;
```