

## **Memoria MAUDE**

Jorge Urbelz Alonso-Cortés

Grado Ingeniería Informática, Universidad de Murcia

Algoritmos y Estructuras de Datos I

Jesús Sánchez Cuadrado

10 de octubre, 2023

# Índice

## Contenido

Datos del estudiante .....	3
Tabla resultados Mooshak.....	4
Envío ejercicios .....	5
Sección 1 .....	5
Ejercicio 101 .....	5
Ejercicio 102 .....	6
Ejercicio 103 .....	6
Ejercicio 104 .....	9
Sección 2 .....	12
Ejercicio 110 .....	12
Ejercicio 111 .....	13
Ejercicio 112 .....	13
Ejercicio 113 .....	14
Ejercicio 114 .....	15
Ejercicio 115 .....	17
Sección 3 .....	19
Ejercicio 120 .....	19
Ejercicio 121 .....	20
Ejercicio 125 .....	21
Sección 4 .....	24
Ejercicio 140 .....	24
Conclusiones y valoración del trabajo .....	28

## Datos del estudiante

Dueño de la cuenta: Jorge Urbelz Alonso-Cortés.

Grupo: 3.3.

Matrícula: 2ª matrícula. Cursó su primera matrícula en el curso 21/22.


Horas estimadas: 1 hora para subir ejercicios de nuevo a Maude y realización memoria.

Datos envío año 21/22:


















- Grupo: 2.

- Compañero de prácticas: Pedro López (dueño de la cuenta).

## Tabla resultados Mooshak

# Pais Equipo	101	102	103	104	105	106	107	Problemas
1  GX URBELZ ALONSO CORTES, JORGE	1 (1)	2 (1)	2 (1)	2 (1)				1 (1)

Problemas	110	111	112	113	114	115	116	117	118	120	121	122	125	126	140	141	142	143	144	Total Puntos
	1 (1)	1 (1)	2 (1)	2 (1)	3 (1)	1 (1)	0 (2)			1 (1)	2 (1)		2 (1)		2 (2)					14 24

#	Tiempo de Concurso	Pais	Equipo	Problema	Lenguaje	Resultado	Estado
2677	145:21:33		GX URBELZ ALONSO CORTES, JORGE	116	Maude	0 Runtime Error	final
2676	145:20:36		GX URBELZ ALONSO CORTES, JORGE	116	Maude	0 Runtime Error	final
2656	144:57:08		GX URBELZ ALONSO CORTES, JORGE	140	Maude	2 Accepted	final
2651	144:52:38		GX URBELZ ALONSO CORTES, JORGE	140	Maude	0 Runtime Error	final
2648	144:50:50		GX URBELZ ALONSO CORTES, JORGE	125	Maude	2 Accepted	final
2647	144:50:05		GX URBELZ ALONSO CORTES, JORGE	121	Maude	2 Accepted	final
2645	144:49:15		GX URBELZ ALONSO CORTES, JORGE	120	Maude	1 Accepted	final
2644	144:48:18		GX URBELZ ALONSO CORTES, JORGE	115	Maude	1 Accepted	final
2643	144:47:19		GX URBELZ ALONSO CORTES, JORGE	113	Maude	3 Accepted	final
2642	144:46:06		GX URBELZ ALONSO CORTES, JORGE	113	Maude	2 Accepted	final
2641	144:45:13		GX URBELZ ALONSO CORTES, JORGE	112	Maude	2 Accepted	final
2640	144:44:14		GX URBELZ ALONSO CORTES, JORGE	111	Maude	1 Accepted	final
2639	144:43:23		GX URBELZ ALONSO CORTES, JORGE	110	Maude	1 Accepted	final
2637	144:42:24		GX URBELZ ALONSO CORTES, JORGE	104	Maude	2 Accepted	final
2635	144:41:10		GX URBELZ ALONSO CORTES, JORGE	103	Maude	2 Accepted	final
590	55:11:11		GX URBELZ ALONSO CORTES, JORGE	102	Maude	2 Accepted	final
581	55:09:17		GX URBELZ ALONSO CORTES, JORGE	101	Maude	1 Accepted	final

## Envío ejercicios

A continuación, daré una pequeña explicación a cada ejercicio realizado. No ahondaré en la dificultad de estos debido a que ya los hice en su momento y no me acuerdo de si me fueron difíciles o no.

Voy a dividir los ejercicios con respecto a las distintas secciones a entregar. Por ejemplo, los ejercicios del 101 al 104 corresponden a la sección 1. Del 110 al 115 a la sección 2. Y así sucesivamente.

### Sección 1

En esta sección me centraré en los ejercicios 101, 102, 103 y 104.

#### Ejercicio 101

Envíos realizados: 1

Envío válido: 581

```
***** NOMBRE *****
fmod NATURAL is

***** CONJUNTOS *****
  protecting BOOL .
  sort N .
  sort NoN .
  subsort NoN < N .

***** SINTAXIS *****
  op cero : -> N .
  op sucesor : N -> N .
  op suma : N N -> N .
  op esCero : N -> Bool .
  op esIgual : N N -> Bool .
  op esDistinto : N N -> Bool .

  op NODEFINIDO : -> NoN .
  op INFINITO : -> NoN .
  op NEGATIVO : -> NoN .

***** SEMANTICA *****
  var n m : N .

  eq suma(cero, n) = n .
  eq suma(sucesor(m), n) = sucesor(suma(m, n)) .
  eq esCero(cero) = true .
  eq esCero(sucesor(n)) = false .
  eq esIgual(cero, n) = esCero(n) .
  eq esIgual(sucesor(n), cero) = false .
  eq esIgual(sucesor(n), sucesor(m)) = esIgual(n, m) .
  eq esDistinto(n, m) = not esIgual(n, m) .
endfm
```

#### Breve descripción

En esta sección trabajaremos a partir de los números naturales. Al ser este el primer ejercicio a realizar, debemos definir lo que es un número natural. Además, introducimos las operaciones más básicas para hacer con estos: la suma, la igualdad, y su contraparte, de naturales.

## Ejercicio 102

Envíos realizados: 1

Envío válido: 590

```

***** SINTAXIS *****
op cero : -> N .
op sucesor : N -> N .
op suma : N N -> N .
op esCero : N -> Bool .
op esIgual : N N -> Bool .
op esDistinto : N N -> Bool .
op producto : N N -> N .
op potencia : N N -> N .
op cuadrado : N -> N .
op factorial : N -> N .

op NODEFINIDO : -> NoN .
op INFINITO : -> NoN .
op NEGATIVO : -> NoN .

***** SEMANTICA *****
..

eq producto(cero, cero) = cero .
eq producto(cero, n) = cero .
eq producto(cero, sucesor(n)) = cero .
eq producto(sucesor(cero), sucesor(cero)) = sucesor(cero) .
eq producto(sucesor(cero), n) = n .
eq producto(sucesor(cero), sucesor(n)) = sucesor(n) .
eq producto(sucesor(m), n) = suma(producto(m, n), n) .

eq potencia(cero, cero) = NODEFINIDO .
eq potencia(n, cero) = sucesor(cero) .
eq potencia(cero, n) = cero .
eq potencia(sucesor(cero), n) = sucesor(cero) .
eq potencia(n, sucesor(cero)) = n .
eq potencia(n, sucesor(n)) = producto(potencia(n, n), n) .
eq potencia(n, sucesor(m)) = producto(potencia(n, m), n) .

eq cuadrado(cero) = cero .
eq cuadrado(sucesor(cero)) = sucesor(cero) .
eq cuadrado(n) = potencia(n, sucesor(sucesor(cero))) .

eq factorial(cero) = sucesor(cero) .
eq factorial(sucesor(cero)) = sucesor(cero) .
eq factorial(sucesor(sucesor(cero))) = sucesor(sucesor(cero)) .
eq factorial(sucesor(n)) = producto(factorial(n), sucesor(n)) .

```

*Breve descripción*

Continuamos con los naturales. En la foto no he incluido todo el código, tan sólo donde he escrito la nueva semántica. El producto, potencia, cuadrado y factorial son parte de las nuevas operaciones sobre los naturales.

## Ejercicio 103

Envíos realizados: 1

Envío válido: 2635

## \*\*\*\*\* SINTAXIS \*\*\*\*\*

```
op cero : -> N .
op sucesor : N -> N .
op suma : N N -> N .
op esCero : N -> Bool .
op esIgual : N N -> Bool .
op esDistinto : N N -> Bool .
op producto : N N -> N .
op potencia : N N -> N .
op cuadrado : N -> N .
op factorial : N -> N .
op predecesor : N -> N .
op resta : N N -> N .
op diferencia : N N -> N .
op difUno : N N -> Bool .

op NODEFINIDO : -> NoN .
op INFINITO : -> NoN .
op NEGATIVO : -> NoN .
```

## \*\*\*\*\* SEMANTICA \*\*\*\*\*

```

eq predecesor(cero) = NEGATIVO .
eq predecesor(sucesor(cero)) = cero .
eq predecesor(sucesor(sucesor(cero))) = sucesor(cero) .
eq predecesor(sucesor(n)) = n .

eq resta(cero, cero) = cero .
eq resta(sucesor(cero), cero) = sucesor(cero) .
eq resta(cero, sucesor(cero)) = NEGATIVO .
eq resta(n, cero) = n .
eq resta(cero, n) = NEGATIVO .
eq resta(n, sucesor(cero)) = predecesor(n) .
eq resta(n, n) = cero .
eq resta(n, sucesor(sucesor(cero))) = predecesor(predecesor(n)) .
eq resta(sucesor(n), n) = sucesor(cero) .
eq resta(n, sucesor(n)) = NEGATIVO .
eq resta(n, sucesor(sucesor(n))) = NEGATIVO .
eq resta(n, sucesor(m)) = resta(n, suma(sucesor(cero), m)) .
eq resta(sucesor(n), m) = resta(suma(sucesor(cero), n), m) .

eq diferencia(cero, cero) = cero .
eq diferencia(cero, sucesor(cero)) = sucesor(cero) .
eq diferencia(sucesor(cero), sucesor(sucesor(cero))) = sucesor(cero) .
eq diferencia(sucesor(cero), cero) = sucesor(cero) .
eq diferencia(cero, sucesor(cero)) = sucesor(cero) .
eq diferencia(sucesor(cero), sucesor(cero)) = cero .
eq diferencia(n, cero) = n .
eq diferencia(cero, n) = n .
eq diferencia(sucesor(n), sucesor(cero)) = n .
eq diferencia(n, n) = cero .
eq diferencia(n, sucesor(sucesor(cero))) = predecesor(predecesor(n)) .
eq diferencia(sucesor(cero), n) = predecesor(n) .
eq diferencia(sucesor(n), n) = sucesor(cero) .
eq diferencia(n, sucesor(n)) = sucesor(cero) .
eq diferencia(n, sucesor(sucesor(n))) = sucesor(sucesor(cero)) .
eq diferencia(n, sucesor(m)) = diferencia(n, suma(sucesor(cero), m)) .

eq difUno(cero, cero) = true .
eq difUno(cero, sucesor(cero)) = true .
eq difUno(sucesor(cero), cero) = true .
eq difUno(sucesor(sucesor(cero)), sucesor(cero)) = true .
eq difUno(sucesor(cero), sucesor(cero)) = true .
eq difUno(cero, sucesor(sucesor(n))) = false .
eq difUno(sucesor(sucesor(n)), cero) = false .
eq difUno(n, n) = true .
eq difUno(n, sucesor(cero)) = false .
eq difUno(n, sucesor(n)) = true .
eq difUno(sucesor(n), n) = true .
eq difUno(n, sucesor(sucesor(n))) = false .
eq difUno(sucesor(sucesor(n)), n) = false .
eq difUno(n, sucesor(m)) = difUno(n, suma(sucesor(cero), m)) .
eq difUno(sucesor(n), m) = difUno(suma(sucesor(cero), n), m) .

```

### Breve descripción

Continuamos con los naturales. En la foto no he incluido todo el código, tan sólo donde he escrito la nueva semántica. El predecesor, la resta y la diferencia son parte de las nuevas operaciones sobre los naturales.



## Ejercicio 104

Envíos realizados: 1

Envío válido: 2637

\*\*\*\*\* SINTAXIS \*\*\*\*\*

```
op cero : -> N .
op sucesor : N -> N .
op suma : N N -> N .
op esCero : N -> Bool .
op esIgual : N N -> Bool .
op esDistinto : N N -> Bool .
op producto : N N -> N .
op potencia : N N -> N .
op cuadrado : N -> N .
op factorial : N -> N .
op esMenor : N N -> Bool .
op esMenorIgual : N N -> Bool .
op esMayor : N N -> Bool .
op esMayorIgual : N N -> Bool .
op maximo : N N -> N .
op minimo : N N -> N .

op NODEFINIDO : -> NoN .
op INFINITO : -> NoN .
op NEGATIVO : -> NoN .
```

\*\*\*\*\* SEMANTICA \*\*\*\*\*

```

eq esMenor(cero, cero) = false .
eq esMenor(sucesor(n), cero) = false .
eq esMenor(sucesor(sucesor(n)), n) = false .
eq esMenor(n, sucesor(sucesor(n))) = true .
eq esMenor(sucesor(cero), sucesor(sucesor(n))) = true .
eq esMenor(sucesor(n), sucesor(cero)) = false .
eq esMenor(cero, sucesor(n)) = true .
eq esMenor(sucesor(cero), cero) = false .
eq esMenor(cero, sucesor(cero)) = true .
eq esMenor(sucesor(cero), sucesor(cero)) = false .
eq esMenor(n, n) = false .
eq esMenor(n, sucesor(n)) = true .
eq esMenor(sucesor(n), n) = false .
eq esMenor(sucesor(n), m) = esMenor(suma(sucesor(cero), n), m) .
eq esMenor(n, sucesor(m)) = esMenor(n, suma(sucesor(cero), m)) .

eq esMenorIgual(cero, cero) = true .
eq esMenorIgual(sucesor(sucesor(n)), n) = false .
eq esMenorIgual(n, sucesor(sucesor(n))) = true .
eq esMenorIgual(sucesor(cero), sucesor(sucesor(n))) = true .
eq esMenorIgual(sucesor(sucesor(n)), sucesor(cero)) = false .
eq esMenorIgual(sucesor(n), cero) = false .
eq esMenorIgual(cero, sucesor(n)) = true .
eq esMenorIgual(sucesor(cero), cero) = esMenor(sucesor(cero), cero) .
eq esMenorIgual(cero, sucesor(cero)) = esMenor(cero, sucesor(cero)) .
eq esMenorIgual(sucesor(cero), sucesor(cero)) = esMenorIgual(cero, cero) .
eq esMenorIgual(n, n) = esMenorIgual(cero, cero) .
eq esMenorIgual(n, sucesor(n)) = esMenorIgual(cero, sucesor(cero)) .
eq esMenorIgual(sucesor(n), n) = esMenorIgual(sucesor(cero), cero) .
eq esMenorIgual(sucesor(n), m) = esMenorIgual(suma(sucesor(cero), n), m) .
eq esMenorIgual(n, sucesor(m)) = esMenorIgual(n, suma(sucesor(cero), m)) .

eq esMayor(cero, cero) = false .
eq esMayor(sucesor(cero), sucesor(sucesor(n))) = false .
eq esMayor(n, sucesor(sucesor(n))) = false .
eq esMayor(sucesor(sucesor(n)), n) = true .
eq esMayor(sucesor(sucesor(n)), sucesor(cero)) = true .
eq esMayor(sucesor(n), cero) = true .
eq esMayor(cero, sucesor(n)) = false .
eq esMayor(sucesor(cero), cero) = true .
eq esMayor(cero, sucesor(cero)) = false .
eq esMayor(sucesor(cero), sucesor(cero)) = esMayor(cero, cero) .
eq esMayor(n, n) = esMayor(cero, cero) .
eq esMayor(n, sucesor(n)) = esMayor(cero, sucesor(cero)) .
eq esMayor(sucesor(n), n) = esMayor(sucesor(cero), cero) .
eq esMayor(sucesor(n), m) = esMayor(suma(sucesor(cero), n), m) .
eq esMayor(n, sucesor(m)) = esMayor(n, suma(sucesor(cero), m)) .

```

```

eq esMayorIgual(cero, cero) = true .
eq esMayorIgual(sucesor(sucesor(n)), n) = true .
eq esMayorIgual(n, sucesor(sucesor(n))) = false .
eq esMayorIgual(sucesor(cero), sucesor(sucesor(n))) = false .
eq esMayorIgual(sucesor(sucesor(n)), sucesor(cero)) = true .
eq esMayorIgual(n, sucesor(sucesor(n))) = false .
eq esMayorIgual(sucesor(n), cero) = true .
eq esMayorIgual(cero, sucesor(n)) = false .
eq esMayorIgual(sucesor(cero), cero) = true .
eq esMayorIgual(cero, sucesor(cero)) = false .
eq esMayorIgual(sucesor(cero), sucesor(cero)) = esMayorIgual(cero, cero) .
eq esMayorIgual(n, n) = esMayorIgual(cero, cero) .
eq esMayorIgual(n, sucesor(n)) = esMayorIgual(cero, sucesor(cero)) .
eq esMayorIgual(sucesor(n), n) = esMayorIgual(sucesor(cero), cero) .
eq esMayorIgual(sucesor(n), m) = esMayorIgual(suma(sucesor(cero), n), m) .
eq esMayorIgual(n, sucesor(m)) = esMayorIgual(n, suma(sucesor(cero), m)) .

eq maximo(cero, cero) = cero .
eq maximo(n, n) = n .
eq maximo(n, sucesor(sucesor(n))) = sucesor(sucesor(n)) .
eq maximo(sucesor(sucesor(n)), n) = sucesor(sucesor(n)) .
eq maximo(sucesor(n), sucesor(cero)) = sucesor(n) .
eq maximo(sucesor(cero), sucesor(sucesor(n))) = sucesor(sucesor(n)) .
eq maximo(cero, sucesor(cero)) = sucesor(cero) .
eq maximo(cero, sucesor(n)) = sucesor(n) .
eq maximo(sucesor(cero), sucesor(cero)) = sucesor(cero) .
eq maximo(sucesor(sucesor(cero)), sucesor(sucesor(cero))) = sucesor(sucesor(cero)) .
eq maximo(sucesor(cero), sucesor(sucesor(cero))) = sucesor(sucesor(cero)) .
eq maximo(sucesor(n), cero) = sucesor(n) .
eq maximo(n, sucesor(n)) = sucesor(n) .
eq maximo(sucesor(n), n) = sucesor(n) .
eq maximo(sucesor(n), m) = maximo(suma(sucesor(cero), n), m) .
eq maximo(n, sucesor(m)) = maximo(n, suma(sucesor(cero), m)) .

eq minimo(cero, cero) = cero .
eq minimo(sucesor(sucesor(cero)), sucesor(sucesor(cero))) = sucesor(sucesor(cero)) .
eq minimo(sucesor(n), sucesor(n)) = sucesor(n) .
eq minimo(sucesor(n), sucesor(cero)) = sucesor(cero) .
eq minimo(sucesor(cero), sucesor(n)) = sucesor(cero) .
eq minimo(cero, sucesor(cero)) = cero .
eq minimo(cero, sucesor(n)) = cero .
eq minimo(sucesor(cero), sucesor(cero)) = sucesor(cero) .
eq minimo(sucesor(cero), sucesor(sucesor(cero))) = sucesor(cero) .
eq minimo(sucesor(n), cero) = cero .
eq minimo(n, sucesor(n)) = n .
eq minimo(sucesor(n), n) = n .
eq minimo(sucesor(sucesor(n)), n) = n .
eq minimo(n, sucesor(sucesor(n))) = n .
eq minimo(sucesor(n), m) = minimo(suma(sucesor(cero), n), m) .
eq minimo(n, sucesor(m)) = minimo(n, suma(sucesor(cero), m)) .
eq minimo(n, n) = n .

```

### Breve descripción

Último ejercicio sobre los naturales. En la foto no he incluido todo el código, tan sólo donde he escrito la nueva semántica. En este caso nos centramos en las comparaciones entre los naturales. Ver si un natural es menor, mayor o si es el máximo o mínimo son parte de esta semántica.

## Sección 2

En esta sección me centraré en los ejercicios 110, 111, 112, 113, 114 y 115.

### Ejercicio 110

Envíos realizados: 1

Envío válido: 2639

```
***** NOMBRE *****
fmod VOCAL is

***** CONJUNTOS *****
  protecting BOOL .
  sort V .

***** SINTAXIS *****
  ops A E I O U : -> V .
  op esIgual : V V -> Bool .
  op esDistinta : V V -> Bool .
  op esMenor : V V -> Bool .

***** SEMANTICA *****
  var v w : V .

  eq esIgual(v, v) = true .
  eq esIgual(v, w) = false .

  eq esDistinta(v, w) = not esIgual(v, w) .

  eq esMenor(v, v) = false .
  eq esMenor(A, E) = true .
  eq esMenor(A, I) = true .
  eq esMenor(A, O) = true .
  eq esMenor(A, U) = true .
  eq esMenor(E, I) = true .
  eq esMenor(E, O) = true .
  eq esMenor(E, U) = true .
  eq esMenor(I, O) = true .
  eq esMenor(I, U) = true .
  eq esMenor(O, U) = true .
  eq esMenor(U, O) = false .
  eq esMenor(U, I) = false .
  eq esMenor(U, E) = false .
  eq esMenor(U, A) = false .
  eq esMenor(O, I) = false .
  eq esMenor(O, E) = false .
  eq esMenor(O, A) = false .
  eq esMenor(I, E) = false .
  eq esMenor(I, A) = false .
  eq esMenor(E, A) = false .
endfm
```

### Breve descripción

Comenzamos una nueva sección centradas en las vocales. Lo primero, como en el ejercicio 101, será definir lo que es una vocal. En la parte de la semántica, usamos la operación esIgual y esDistinta para saber si dos vocales son iguales o distintos. Además, añadimos esMenor, para crear un orden entre vocales. De esta manera, Maude sabrá que debe respetar el orden 'A-E-I-O-U'.

## Ejercicio 111

Envíos realizados: 1

Envío válido: 2640

```

***** NOMBRE *****
fmod PILA is

***** CONJUNTOS *****
    protecting BOOL .
    protecting VOCAL .
    sort MensajePilas .
    sort P .
    subsorts MensajePilas < V .

***** SINTAXIS *****
    op pilaVacía : -> P .
    op esVacía : P -> Bool .
    op push : V P -> P .
    op pop : P -> P .
    op tope : P -> V .

    op ERRORPILAVACIA : -> MensajePilas .

***** SEMANTICA *****
    var p : P .
    var v : V .

    eq esVacía(pilaVacía) = true .
    eq esVacía(push(v, p)) = false .
    eq pop(pilaVacía) = pilaVacía .
    eq pop(push(v, p)) = p .
    eq tope(pilaVacía) = ERRORPILAVACIA .
    eq tope(push(v, p)) = v .
endfm

```

*Breve descripción*

Añadimos al código, que describimos en el anterior ejercicio, la definición de la pila. La pila viene con las siguientes operaciones: esVacía (para saber si se encuentra alguna vocal dentro de la pila), pop (saca de la pila la vocal que se encuentra en el tope de esta), tope (devuelve la vocal que se encuentra en el tope de la pila) y push (añade al principio de la pila una vocal).

## Ejercicio 112

Envíos realizados: 1

Envío válido: 2641

```

***** SINTAXIS *****
op pilaVacía : -> P .
op esVacía : P -> Bool .
op push : V P -> P .
op pop : P -> P .
op tope : P -> V .
op esIgual : P P -> Bool .
op primero : P -> V .
op tamaño : P -> N .
op cuentaVocal : V P -> N .

op ERRORPILAVACIA : -> MensajePilas .

***** SEMANTICA *****
var p q : P .
var v w t r : V .

eq esVacía(pilaVacía) = true .
eq esVacía(push(v, p)) = false .
eq pop(pilaVacía) = pilaVacía .
eq pop(push(v, p)) = p .
eq tope(pilaVacía) = ERRORPILAVACIA .
eq tope(push(v, p)) = v .

eq esIgual(pilaVacía, pilaVacía) = true .
eq esIgual(p, pilaVacía) = false .
eq esIgual(pilaVacía, p) = false .
eq esIgual(pilaVacía, pop(push(v, p))) = true .
eq esIgual(push(v, p), pilaVacía) = false .
eq esIgual(push(v, p), push(v, p)) = true .
eq esIgual(push(v, p), pop(push(v, p))) = false .
eq esIgual(push(v, p), pop(push(v, push(v, p)))) = true .
eq esIgual(push(A, p), push(E, p)) = false .
eq esIgual(push(A, p), push(I, p)) = false .
eq esIgual(push(A, p), push(O, p)) = false .
eq esIgual(push(A, p), push(U, p)) = false .
eq esIgual(push(E, p), push(I, p)) = false .
eq esIgual(push(E, p), push(O, p)) = false .
eq esIgual(push(E, p), push(U, p)) = false .
eq esIgual(push(I, p), push(O, p)) = false .
eq esIgual(push(I, p), push(U, p)) = false .
eq esIgual(push(v, p), push(t, q)) = if esIgual(v, t) == true then esIgual(p, q) else false fi .

eq primero(pilaVacía) = ERRORPILAVACIA .
eq primero(push(v, pilaVacía)) = v .
eq primero(push(v, p)) = primero(p) .
eq primero(push(v, push(A, p))) = A .
eq primero(pop(push(v, push(A, p)))) = A .

eq tamaño(pilaVacía) = cero .
eq tamaño(push(v, p)) = sucesor(tamaño(p)) .
eq tamaño(push(v, push(v, p))) = sucesor(sucesor(cero)) .
eq tamaño(push(v, pop(push(v, p)))) = sucesor(cero) .

eq cuentaVocal(v, pilaVacía) = cero .
eq cuentaVocal(v, push(t, p)) = if esIgual(v, t) == true then sucesor(cuentaVocal(v, p)) else cuentaVocal(v, p) fi .

```

### Breve descripción

Añadimos a la pila la operación esIgual (ya la usamos con los naturales, pero en este caso son vocales), primero (devuelve el primer elemento de pila), tamaño (devuelve el tamaño de pila) y cuentaVocal, el cual cuenta cuantas veces tenemos almacenada una vocal en pila. Esta operación hará uso de un if.

### Ejercicio 113

Envíos realizados: 1

Envío válido: 2642

```

***** NOMBRE *****
fmod COLA is

***** CONJUNTOS *****
  protecting BOOL .
  protecting VOCAL .
  sort C .
  sort MensajeColas .
  subsorts MensajeColas < V .

***** SINTAXIS *****
  op colaVacía : -> C .
  op esVacía : C -> Bool .
  op meter : V C -> C .
  op sacar : C -> C .
  op cabecera : C -> V .

  op ERRORCOLAVACIA : -> MensajeColas .

***** SEMANTICA *****
  var c : C .
  var v w q r t : V .

  eq esVacía(colaVacía) = true .
  eq esVacía(meter(v, c)) = false .
  eq esVacía(sacar(meter(v, colaVacía))) = true .
  eq esVacía(meter(q, meter(t, meter(w, meter(v, colaVacía))))) = false .

  eq sacar(colaVacía) = colaVacía .
  eq sacar(meter(v, colaVacía)) = colaVacía .
  eq sacar(meter(v, meter(w, colaVacía))) = meter(v, colaVacía) .
  eq sacar(meter(v, c)) = meter(v, sacar(c)) .

  eq cabecera(colaVacía) = ERRORCOLAVACIA .
  eq cabecera(sacar(meter(v, colaVacía))) = ERRORCOLAVACIA .
  eq cabecera(sacar(meter(v, meter(w, c)))) = v .
  eq cabecera(meter(v, colaVacía)) = v .
  eq cabecera(meter(w, meter(v, colaVacía))) = v .
  eq cabecera(meter(v,c)) = cabecera(c) .

endfm

```

#### Breve descripción

¡Nueva definición! En este caso se trata de una cola. Las colas son completamente opuestas a las pilas. Mientras las pilas son estructuras *LIFO* (*Last In, First Out*), las colas son un exponente de la estrategia *FIFO* (*First In, First Out*). En la imagen podemos apreciar su sintaxis y su semántica. Volvemos a jugar con las vocales. *EsVacía* es un booleano que devuelve true o false dependiendo de si la cola se encuentra vacía o no. *Meter* introduce un nuevo elemento en la cola, al contrario de *sacar*, el cual lo elimina (en este caso elimina el primero que se encuentra en la cola). Por último, *cabecera* muestra la primera vocal.

#### Ejercicio 114

Envíos realizados: 1

Envío válido: 2643



```

***** NOMBRE *****
fmod COLA is

***** CONJUNTOS *****
protecting BOOL .
protecting VOCAL .
protecting NATURAL .
sort C .
sort MensajeColas .
subsorts MensajeColas < V .

***** SINTAXIS *****
op colaVacia : -> C .
op esVacia : C -> Bool .
op meter : V C -> C .
op sacar : C -> C .
op cabecera : C -> V .
op meterVarias : V N C -> C .
op sacarVarias : N C -> C .
op esIgual : C C -> Bool .
op tamano : C -> N .

op ERRORCOLAVACIA : -> MensajeColas .

***** SEMANTICA *****
var c d : C .
var v w q r t : V .
var n : N .

eq esVacia(colaVacia) = true .
eq esVacia(meter(v, c)) = false .
eq esVacia(sacar(meter(v, colaVacia))) = true .
eq esVacia(meter(q, meter(t, meter(w, meter(v, colaVacia))))) = false .

eq sacar(colaVacia) = colaVacia .
eq sacar(meter(v, colaVacia)) = colaVacia .
eq sacar(meter(v, meter(w, colaVacia))) = meter(v, colaVacia) .
eq sacar(meter(v, c)) = meter(v, sacar(c)) .

eq cabecera(colaVacia) = ERRORCOLAVACIA .
eq cabecera(sacar(meter(v, colaVacia))) = ERRORCOLAVACIA .
eq cabecera(sacar(meter(v, meter(w, c)))) = v .
eq cabecera(meter(v, colaVacia)) = v .
eq cabecera(meter(w, meter(v, colaVacia))) = v .
eq cabecera(meter(v, c)) = cabecera(c) .

eq meterVarias(v, cero, c) = c .
eq meterVarias(w, cero, colaVacia) = colaVacia .
eq meterVarias(v, sucesor(n), c) = meterVarias(v, n, meter(v, c)) .

eq sacarVarias(cero, c) = c .
eq sacarVarias(sucesor(n), c) = sacarVarias(n, sacar(c)) .

eq esIgual(colaVacia, colaVacia) = true .
eq esIgual(c, colaVacia) = false .
eq esIgual(colaVacia, c) = false .
eq esIgual(meter(v, c), meter(w, d)) = if esIgual(v, w) == true then esIgual(c, d) else false fi .

eq tamano(colaVacia) = cero .
eq tamano(meter(v, c)) = sucesor(tamano(c)) .
eq tamano(meterVarias(v, sucesor(n), c)) = suma(n, tamano(c)) .

endfm

```

### Breve descripción

En el anterior ejercicio nos centrábamos exclusivamente en las vocales como elementos. Ahora, tendremos en cuenta también los naturales. Añadimos nuevas operaciones a la semántica de cola. Estas operaciones son: meterVarias (introduce varios elementos a la cola. Tanto vocales como naturales), sacarVarias (la contraparte de meterVarias), esIgual (Muestra mediante resultado booleano si dos elementos son iguales. Hace uso de un if) y tamaño.



## Ejercicio 115

Envíos realizados: 1

Envío válido: 2644

```

***** NOMBRE *****
fmod NATURAL is

***** CONJUNTOS *****
  protecting BOOL .
  sort N .
  sort NoN .
  subsort NoN < N .

***** SINTAXIS *****
  op cero : -> N .
  op sucesor : N -> N .
  op suma : N N -> N .
  op esCero : N -> Bool .
  op esIgual : N N -> Bool .
  op esDistinto : N N -> Bool .

  op NODEFINIDO : -> NoN .
  op INFINITO : -> NoN .
  op NEGATIVO : -> NoN .

***** SEMANTICA *****
  var n m : N .

  eq suma(cero, n) = n .
  eq suma(sucesor(m), n) = sucesor(suma(m, n)) .
  eq esCero(cero) = true .
  eq esCero(sucesor(n)) = false .
  eq esIgual(cero, n) = esCero(n) .
  eq esIgual(sucesor(n), cero) = false .
  eq esIgual(sucesor(n), sucesor(m)) = esIgual(n, m) .
  eq esDistinto(n, m) = not esIgual(n, m) .
endfm

***** NOMBRE *****
fmod VOCAL is

***** CONJUNTOS *****
  protecting BOOL .
  sort V .

***** SINTAXIS *****
  ops A E I O U : -> V .
  op esIgual : V V -> Bool .
  op esDistinta : V V -> Bool .
  op esMenor : V V -> Bool .

***** SEMANTICA *****
  var v w : V .

  eq esIgual(v, v) = true .
  eq esIgual(v, w) = false .
  eq esDistinta(v, w) = not esIgual(v, w) .

```

```

eq esMenor(v, v) = false .
eq esMenor(A, E) = true .
eq esMenor(A, I) = true .
eq esMenor(A, O) = true .
eq esMenor(A, U) = true .
eq esMenor(E, I) = true .
eq esMenor(E, O) = true .
eq esMenor(E, U) = true .
eq esMenor(I, O) = true .
eq esMenor(I, U) = true .
eq esMenor(O, U) = true .
eq esMenor(U, O) = false .
eq esMenor(U, I) = false .
eq esMenor(U, E) = false .
eq esMenor(U, A) = false .
eq esMenor(O, I) = false .
eq esMenor(O, E) = false .
eq esMenor(O, A) = false .
eq esMenor(I, E) = false .
eq esMenor(I, A) = false .
eq esMenor(E, A) = false .
endfm

***** NOMBRE *****
fmod LISTA is

***** CONJUNTOS *****
protecting BOOL .
protecting NATURAL .
protecting VOCAL .
sort L .
sort MensajeListas .
subsorts MensajeListas < V .

***** SINTAXIS *****
op listaVacía : -> L .
op esVacía : L -> Bool .
op insertar : V L -> L .
op insertarFinal : V L -> L .
op tamaño : L -> N .

op ErrorListaVacía : -> MensajeListas .

***** SEMANTICA *****
var l : L .
var v w : V .

eq esVacía(listaVacía) = true .
eq esVacía(insertar(v, l)) = false .
eq insertarFinal(v, listaVacía) = insertar(v, listaVacía) .
eq insertarFinal(v, insertar(w, l)) = insertar(w, insertarFinal(v, l)) .
eq tamaño(listaVacía) = cero .
eq tamaño(insertar(v, l)) = sucesor(tamaño(l)) .
endfm

```

### Breve descripción

Dejamos la pila y la cola a un lado. Para este ejercicio trabajaremos con listas. Lo bueno de este tipo de datos es que podemos introducir y eliminar elementos de la lista por ambos extremos.

### Sección 3

En esta sección me centraré en los ejercicios 120, 121 y 125.

#### Ejercicio 120

Envíos realizados: 1

Envío válido: 2645

```
***** NOMBRE *****
fmod NATURAL is

***** CONJUNTOS *****
protecting BOOL .
sort N .
sort NoN .
subsort NoN < N .

***** SINTAXIS *****
op cero : -> N .
op sucesor : N -> N .
op suma : N N -> N .
op esCero : N -> Bool .
op esIgual : N N -> Bool .
op esDistinto : N N -> Bool .

op NODEFINIDO : -> NoN .
op INFINITO : -> NoN .
op NEGATIVO : -> NoN .

***** SEMANTICA *****
var n m : N .

eq suma(cero, n) = n .
eq suma(sucesor(m), n) = sucesor(suma(m, n)) .
eq esCero(cero) = true .
eq esCero(sucesor(n)) = false .
eq esIgual(cero, n) = esCero(n) .
eq esIgual(sucesor(n), cero) = false .
eq esIgual(sucesor(n), sucesor(m)) = esIgual(n, m) .
eq esDistinto(n, m) = not esIgual(n, m) .
endfm

***** NOMBRE *****
fmod VOCAL is

***** CONJUNTOS *****
protecting BOOL .
sort V .

***** SINTAXIS *****
ops A E I O U : -> V .
op esIgual : V V -> Bool .
op esDistinta : V V -> Bool .
op esMenor : V V -> Bool .

***** SEMANTICA *****
var v w : V .

eq esIgual(v, v) = true .
eq esIgual(v, w) = false .
eq esDistinta(v, w) = not esIgual(v, w) .
```

```

eq esMenor(v, v) = false .
eq esMenor(A, E) = true .
eq esMenor(A, I) = true .
eq esMenor(A, O) = true .
eq esMenor(A, U) = true .
eq esMenor(E, I) = true .
eq esMenor(E, O) = true .
eq esMenor(E, U) = true .
eq esMenor(I, O) = true .
eq esMenor(I, U) = true .
eq esMenor(O, U) = true .
eq esMenor(U, O) = false .
eq esMenor(U, I) = false .
eq esMenor(U, E) = false .
eq esMenor(U, A) = false .
eq esMenor(O, I) = false .
eq esMenor(O, E) = false .
eq esMenor(O, A) = false .
eq esMenor(I, E) = false .
eq esMenor(I, A) = false .
eq esMenor(E, A) = false .
endfm

***** NOMBRE *****
fmod CONJUNTO is

***** CONJUNTOS *****
protecting BOOL .
protecting NATURAL .
protecting VOCAL .
sort C .

***** SINTAXIS *****
op cjtoVacio : -> C .
op esVacio : C -> Bool .
op insertar : V C -> C .
op esMiembro : V C -> Bool .
op tamano : C -> N .

***** SEMANTICA *****
var v w : V .
var c : C .

eq esVacio(cjtoVacio) = true .
eq esVacio(insertar(v, c)) = false .
eq esMiembro(v, cjtoVacio) = false .
eq esMiembro(v, insertar(w, c)) = esIgual(v, w) or esMiembro(v, c) .
eq tamano(cjtoVacio) = cero .
eq tamano(insertar(v, c)) = suma(tamano(c), if esMiembro(v, c) then cero else sucesor(cero) fi ) .
endfm

```

### Breve descripción

Comenzamos la tercera sección con la primera aparición de los conjuntos. Aquí, el orden de inserción de los elementos es irrelevante. En este caso concreto, nos centramos en crear un conjunto de vocales. Las operaciones a usar son parecidas a otras anteriormente vistas.

### Ejercicio 121

Envíos realizados: 1

Envío válido: 2647

```

***** NOMBRE *****
fmod CONJUNTO is

***** CONJUNTOS *****
protecting BOOL .
protecting NATURAL .
protecting VOCAL .
sort C .

***** SINTAXIS *****
op cjtoVacio : -> C .
op esVacio : C -> Bool .
op insertar : V C -> C .
op esMiembro : V C -> Bool .
op tamano : C -> N .
op union : C C -> C .
op interseccion : C C -> C .
op diferencia : C C -> C .
op eliminar : V C -> C .

***** SEMANTICA *****
var v w t r : V .
var c d f : C .

eq esVacio(cjtoVacio) = true .
eq esVacio(insertar(v, c)) = false .

eq esMiembro(v, cjtoVacio) = false .
eq esMiembro(v, insertar(w, c)) = esIgual(v, w) or esMiembro(v, c) .

eq tamano(cjtoVacio) = cero .
eq tamano(insertar(v, c)) = suma(tamano(c), if esMiembro(v, c) then cero else sucesor(cero) fi ) .

eq union(cjtoVacio, cjtoVacio) = cjtoVacio .
eq union(cjtoVacio, c) = c .
eq union(insertar(v, c), d) = insertar(v, union(c, d)) .

eq interseccion(cjtoVacio, c) = cjtoVacio .
eq interseccion(insertar(v, c), d) = if esMiembro(v, d) == true then insertar(v, interseccion(c, d)) else interseccion (c, d) fi .

eq diferencia(cjtoVacio, cjtoVacio) = cjtoVacio .
eq diferencia(c, cjtoVacio) = c .
eq diferencia(c, insertar(v, d)) = diferencia(eliminar(v, c), d) .

eq eliminar(v, cjtoVacio) = cjtoVacio .
eq eliminar(v, insertar(w, c)) = if esIgual(v, w) then eliminar(v, c) else insertar(w, eliminar(v, c)) fi .

```

### Breve descripción

Añadimos a los conjuntos nuevas operaciones, como unión e intersección de dos conjuntos de vocales, la diferencia (todos los conjuntos que están en una variable que no están en la otra) y eliminar (elimina una vocal del conjunto).

### Ejercicio 125

Envíos realizados: 1

Envío válido: 2648

```

***** NOMBRE *****
fmod NATURAL is
***** CONJUNTOS *****
  protecting BOOL .
  sort N .
  sort NoN .
  subsort NoN < N .
***** SINTAXIS *****
  op cero : -> N .
  op sucesor : N -> N .
  op suma : N N -> N .
  op esCero : N -> Bool .
  op esIgual : N N -> Bool .
  op esDistinto : N N -> Bool .
  op NODEFINIDO : -> NoN .
  op INFINITO : -> NoN .
  op NEGATIVO : -> NoN .
***** SEMANTICA *****
  var n m : N .
  eq suma(cero, n) = n .
  eq suma(sucesor(m), n) = sucesor(suma(m, n)) .
  eq esCero(cero) = true .
  eq esCero(sucesor(n)) = false .
  eq esIgual(cero, n) = esCero(n) .
  eq esIgual(sucesor(n), cero) = false .
  eq esIgual(sucesor(n), sucesor(m)) = esIgual(n, m) .
  eq esDistinto(n, m) = not esIgual(n, m) .
endfm

```

```

***** NOMBRE *****
fmod VOCAL is
***** CONJUNTOS *****
  protecting BOOL .
  protecting NATURAL .
  sort V .
  sort B .
***** SINTAXIS *****
  ops A E I O U : -> V .
  op esIgual : V V -> Bool .
  op esDistinta : V V -> Bool .
  op esMenor : V V -> Bool .
  op bolsaVacía : -> B .
  op esVacía : B -> Bool .
  op insertar : V B -> B .
  op contar : V B -> N .
  op eliminar : V B -> B .
***** SEMANTICA *****
  var v w : V .
  var n : N .
  var b : B .
  eq esIgual(v, v) = true .
  eq esIgual(v, w) = false .
  eq esDistinta(v, w) = not esIgual(v, w) .
  eq esMenor(v, v) = false .
  eq esMenor(A, E) = true .
  eq esMenor(A, I) = true .
  eq esMenor(A, O) = true .
  eq esMenor(A, U) = true .
  eq esMenor(E, I) = true .
  eq esMenor(E, O) = true .
  eq esMenor(E, U) = true .
  eq esMenor(I, O) = true .
  eq esMenor(I, U) = true .
  eq esMenor(O, U) = true .
  eq esMenor(U, O) = false .
  eq esMenor(U, I) = false .
  eq esMenor(U, E) = false .
  eq esMenor(U, A) = false .
  eq esMenor(O, I) = false .
  eq esMenor(O, E) = false .
  eq esMenor(O, A) = false .
  eq esMenor(I, E) = false .
  eq esMenor(I, A) = false .
  eq esMenor(E, A) = false .
  eq esVacía(bolsaVacía) = true .
  eq esVacía(insertar(v, b)) = false .
  eq contar(v, bolsaVacía) = cero .
  eq contar(v, insertar(v, bolsaVacía)) = sucesor(cero) .
  eq contar(v, insertar(w, bolsaVacía)) = cero .
  eq contar(v, insertar(v, b)) = sucesor(contar(v, b)) .
  eq contar(v, insertar(w, b)) = if esIgual(v, w) then sucesor(contar(v, b)) else contar(v, b) fi .
  eq eliminar(v, bolsaVacía) = bolsaVacía .
  eq eliminar(v, insertar(w, b)) = if esIgual(v, w) then b else insertar(w, eliminar(v, b)) fi .
endfm

```

### Breve descripción

En el último ejercicio realizado de esta sección, mostraremos el uso de un tipo abstracto de conjunto: la bolsa. Esta añade una nueva distinción: la bolsa puede contener 0 o más repeticiones de un elemento específico. Para ello, se realizan las operaciones básicas para que funcione.

## Sección 4

En esta sección me centraré en el ejercicio 140.

### Ejercicio 140

Envíos realizados: 2

Envío válido: 2656

```
***** NOMBRE *****
fmod NATURAL is
***** CONJUNTOS *****
  protecting BOOL .
  sort N .
  sort NoN .
  subsort NoN < N .
***** SINTAXIS *****
  op cero : -> N .
  op sucesor : N -> N .
  op suma : N N -> N .
  op esCero : N -> Bool .
  op esIgual : N N -> Bool .
  op esDistinto : N N -> Bool .
  op producto : N N -> N .
  op potencia : N N -> N .
  op cuadrado : N -> N .
  op factorial : N -> N .
  op esMenor : N N -> Bool .
  op esMenorIgual : N N -> Bool .
  op esMayor : N N -> Bool .
  op esMayorIgual : N N -> Bool .
  op maximo : N N -> N .
  op minimo : N N -> N .
  op NODEFINIDO : -> NoN .
  op INFINITO : -> NoN .
  op NEGATIVO : -> NoN .
***** SEMANTICA *****
  var n m : N .
  eq suma(cero, n) = n .
  eq suma(sucesor(m), n) = sucesor(suma(m, n)) .
  eq esCero(cero) = true .
  eq esCero(sucesor(n)) = false .
  eq esIgual(cero, n) = esCero(n) .
  eq esIgual(sucesor(n), cero) = false .
  eq esIgual(sucesor(n), sucesor(m)) = esIgual(n, m) .
  eq esDistinto(n, m) = not esIgual(n, m) .
  eq producto(cero, cero) = cero .
  eq producto(cero, n) = cero .
  eq producto(cero, sucesor(n)) = cero .
  eq producto(sucesor(cero), sucesor(cero)) = sucesor(cero) .
  eq producto(sucesor(cero), n) = n .
  eq producto(sucesor(cero), sucesor(n)) = sucesor(n) .
  eq producto(sucesor(m), n) = suma(producto(m, n), n) .
  eq potencia(cero, cero) = NODEFINIDO .
  eq potencia(n, cero) = sucesor(cero) .
  eq potencia(cero, n) = cero .
  eq potencia(sucesor(cero), n) = sucesor(cero) .
  eq potencia(n, sucesor(cero)) = n .
  eq potencia(n, sucesor(n)) = producto(potencia(n, n), n) .
  eq potencia(n, sucesor(m)) = producto(potencia(n, m), n) .
  eq cuadrado(cero) = cero .
  eq cuadrado(sucesor(cero)) = sucesor(cero) .
  eq cuadrado(n) = potencia(n, sucesor(sucesor(cero))) .
  eq factorial(cero) = sucesor(cero) .
  eq factorial(sucesor(cero)) = sucesor(cero) .
  eq factorial(sucesor(sucesor(cero))) = sucesor(sucesor(cero)) .
  eq factorial(sucesor(n)) = producto(factorial(n), sucesor(n)) .
  eq esMenor(cero, cero) = false .
  eq esMenor(sucesor(n), cero) = false .
  eq esMenor(sucesor(sucesor(n)), n) = false .
  eq esMenor(n, sucesor(sucesor(n))) = true .
  eq esMenor(sucesor(cero), sucesor(sucesor(n))) = true .
```



```

eq esMenor(sucesor(n), sucesor(cero)) = false .
eq esMenor(cero, sucesor(n)) = true .
eq esMenor(sucesor(cero), cero) = false .
eq esMenor(cero, sucesor(cero)) = true .
eq esMenor(sucesor(cero), sucesor(cero)) = false .
eq esMenor(n, n) = false .
eq esMenor(n, sucesor(n)) = true .
eq esMenor(sucesor(n), n) = false .
eq esMenor(sucesor(n), m) = esMenor(suma(sucesor(cero), n), m) .
eq esMenor(n, sucesor(m)) = esMenor(n, suma(sucesor(cero), m)) .
eq esMenorIgual(cero, cero) = true .
eq esMenorIgual(sucesor(sucesor(n)), n) = false .
eq esMenorIgual(n, sucesor(sucesor(n))) = true .
eq esMenorIgual(sucesor(cero), sucesor(sucesor(n))) = true .
eq esMenorIgual(sucesor(sucesor(n)), sucesor(cero)) = false .
eq esMenorIgual(sucesor(n), cero) = false .
eq esMenorIgual(cero, sucesor(n)) = true .
eq esMenorIgual(sucesor(cero), cero) = esMenor(sucesor(cero), cero) .
eq esMenorIgual(cero, sucesor(cero)) = esMenor(cero, sucesor(cero)) .
eq esMenorIgual(sucesor(cero), sucesor(cero)) = esMenorIgual(cero, cero) .
eq esMenorIgual(n, n) = esMenorIgual(cero, cero) .
eq esMenorIgual(n, sucesor(n)) = esMenorIgual(cero, sucesor(cero)) .
eq esMenorIgual(sucesor(n), n) = esMenorIgual(sucesor(cero), cero) .
eq esMenorIgual(sucesor(n), m) = esMenorIgual(suma(sucesor(cero), n), m) .
eq esMenorIgual(n, sucesor(m)) = esMenorIgual(n, suma(sucesor(cero), m)) .
eq esMayor(cero, cero) = false .
eq esMayor(sucesor(cero), sucesor(sucesor(n))) = false .
eq esMayor(n, sucesor(sucesor(n))) = false .
eq esMayor(sucesor(sucesor(n)), n) = true .
eq esMayor(sucesor(sucesor(n)), sucesor(cero)) = true .
eq esMayor(sucesor(n), cero) = true .
eq esMayor(cero, sucesor(n)) = false .
eq esMayor(sucesor(cero), cero) = true .
eq esMayor(cero, sucesor(cero)) = false .
eq esMayor(sucesor(cero), sucesor(cero)) = esMayor(cero, cero) .
eq esMayor(n, n) = esMayor(cero, cero) .
eq esMayor(n, sucesor(n)) = esMayor(cero, sucesor(cero)) .
eq esMayor(sucesor(n), n) = esMayor(sucesor(cero), cero) .
eq esMayor(sucesor(n), m) = esMayor(suma(sucesor(cero), n), m) .
eq esMayor(n, sucesor(m)) = esMayor(n, suma(sucesor(cero), m)) .
eq esMayorIgual(cero, cero) = true .
eq esMayorIgual(sucesor(sucesor(n)), n) = true .
eq esMayorIgual(n, sucesor(sucesor(n))) = false .
eq esMayorIgual(sucesor(cero), sucesor(sucesor(n))) = false .
eq esMayorIgual(sucesor(sucesor(n)), sucesor(cero)) = true .
eq esMayorIgual(n, sucesor(sucesor(n))) = false .
eq esMayorIgual(sucesor(n), cero) = true .
eq esMayorIgual(cero, sucesor(n)) = false .
eq esMayorIgual(sucesor(cero), cero) = true .
eq esMayorIgual(cero, sucesor(cero)) = false .
eq esMayorIgual(sucesor(cero), sucesor(cero)) = esMayorIgual(cero, cero) .
eq esMayorIgual(n, n) = esMayorIgual(cero, cero) .
eq esMayorIgual(n, sucesor(n)) = esMayorIgual(cero, sucesor(cero)) .
eq esMayorIgual(sucesor(n), n) = esMayorIgual(sucesor(cero), cero) .
eq esMayorIgual(sucesor(n), m) = esMayorIgual(suma(sucesor(cero), n), m) .
eq esMayorIgual(n, sucesor(m)) = esMayorIgual(n, suma(sucesor(cero), m)) .
eq maximo(cero, cero) = cero .
eq maximo(n, n) = n .
eq maximo(n, sucesor(sucesor(n))) = sucesor(sucesor(n)) .
eq maximo(sucesor(sucesor(n)), n) = sucesor(sucesor(n)) .
eq maximo(sucesor(n), sucesor(cero)) = sucesor(n) .
eq maximo(sucesor(cero), sucesor(sucesor(n))) = sucesor(sucesor(n)) .
eq maximo(cero, sucesor(cero)) = sucesor(cero) .
eq maximo(cero, sucesor(n)) = sucesor(n) .

```

---

```

eq maximo(sucesor(cero), sucesor(cero)) = sucesor(cero) .
eq maximo(sucesor(sucesor(cero)), sucesor(sucesor(cero))) = sucesor(sucesor(cero)) .
eq maximo(sucesor(cero), sucesor(sucesor(cero))) = sucesor(sucesor(cero)) .
eq maximo(sucesor(n), cero) = sucesor(n) .
eq maximo(n, sucesor(n)) = sucesor(n) .
eq maximo(sucesor(n), n) = sucesor(n) .
eq maximo(sucesor(n), m) = maximo(suma(sucesor(cero), n), m) .
eq maximo(n, sucesor(m)) = maximo(n, suma(sucesor(cero), m)) .
eq minimo(cero, cero) = cero .
eq minimo(sucesor(sucesor(cero)), sucesor(sucesor(cero))) = sucesor(sucesor(cero)) .
eq minimo(sucesor(n), sucesor(n)) = sucesor(n) .
eq minimo(sucesor(n), sucesor(cero)) = sucesor(cero) .
eq minimo(sucesor(cero), sucesor(n)) = sucesor(cero) .
eq minimo(cero, sucesor(cero)) = cero .
eq minimo(cero, sucesor(n)) = cero .
eq minimo(sucesor(cero), sucesor(cero)) = sucesor(cero) .
eq minimo(sucesor(cero), sucesor(sucesor(cero))) = sucesor(cero) .
eq minimo(sucesor(n), cero) = cero .
eq minimo(n, sucesor(n)) = n .
eq minimo(sucesor(n), n) = n .
eq minimo(sucesor(sucesor(n)), n) = n .
eq minimo(n, sucesor(sucesor(n))) = n .
eq minimo(sucesor(n), m) = minimo(suma(sucesor(cero), n), m) .
eq minimo(n, sucesor(m)) = minimo(n, suma(sucesor(cero), m)) .
eq minimo(n, n) = n .
endfm
***** NOMBRE *****
fmod VOCAL is
***** CONJUNTOS *****
protecting BOOL .
sort V .
***** SINTAXIS *****
ops A E I O U : -> V .
op esIgual : V V -> Bool .
op esDistinta : V V -> Bool .
op esMenor : V V -> Bool .
***** SEMANTICA *****
var v w : V .
eq esIgual(v, v) = true .
eq esIgual(v, w) = false .
eq esDistinta(v, w) = not esIgual(v, w) .
eq esMenor(v, v) = false .
eq esMenor(A, E) = true .
eq esMenor(A, I) = true .
eq esMenor(A, O) = true .
eq esMenor(A, U) = true .
eq esMenor(E, I) = true .
eq esMenor(E, O) = true .
eq esMenor(E, U) = true .
eq esMenor(I, O) = true .
eq esMenor(I, U) = true .
eq esMenor(O, U) = true .
eq esMenor(U, O) = false .
eq esMenor(U, I) = false .
eq esMenor(U, E) = false .
eq esMenor(U, A) = false .
eq esMenor(O, I) = false .
eq esMenor(O, E) = false .
eq esMenor(O, A) = false .
eq esMenor(I, E) = false .
eq esMenor(I, A) = false .
eq esMenor(E, A) = false .
endfm
***** NOMBRE *****

```

```

fmod ARBOLBINARIODEVOCALES is
***** CONJUNTOS *****
    protecting BOOL .
    protecting NATURAL .
    protecting VOCAL .
    sort AB .
***** SINTAXIS *****
    op arbolVacio : -> AB .
    op esVacio : AB -> Bool .
    op construir : V AB AB -> AB .
    op construirRaiz : V -> AB .
    op altura : AB -> N .
    op numNodos : AB -> N .
***** SEMANTICA *****
    var v w t : V .
    var a1 a2 : AB .
    var n : N .
    eq esVacio(arbolVacio) = true .
    eq esVacio(construir(v, a1, a2)) = false .
    eq construirRaiz(v) = construir(v, arbolVacio, arbolVacio) .
    eq altura(arbolVacio) = cero .
    eq altura(construir(v, a1, a2)) = sucesor(maximo(altura(a1), altura(a2))) .
    eq numNodos(arbolVacio) = cero .
    eq numNodos(construir(v, a1, a2)) = sucesor(suma(numNodos(a1), numNodos(a2))) .
endfm

```

#### *Breve descripción*

Para terminar, el ejercicio más complejo no podía ser otro que uno que incluye árboles. Nos centramos en el TAD árbol binario de vocales. Las operaciones que forman parte de la semántica son: `esVacio` (booleano que devuelve si el árbol está vacío), `construirRaiz` (dada una vocal, crea un árbol binario con raíz `v`, el subárbol izquierdo es `a1` y el derecho `a2`, ambos vacíos), `construir` (dado una vocal y dos subárboles, `a1` y `a2`, coloca en el nodo raíz a `v`. El subárbol izquierdo es `a1` y el derecho `a2`), `altura` (calcula la altura del árbol, usando un número natural) y `numNodos` (calcula el número de nodos, otro natural).

## Conclusiones y valoración del trabajo

Como ya he expresado al principio de esta memoria, ya soy de segunda matrícula, por lo que para la realización del trabajo no he tenido que escribir nada de código. He rescatado lo que ya hicimos mi compañero de prácticas y yo hace dos años. Esta memoria está escrita desde cero, ya que mi compañero fue quien subió a la tarea la anterior memoria. Fallo mío fue el no pedirle que me enviase una copia. Pero no todo es negativo. La parte positiva de haber tenido que hacer la memoria otra vez es que he podido recordar como se programaba en Maude. Recuerdo que en su momento nos costó sacarla adelante. Sin embargo, mirándola desde un punto distinto y con mucha más experiencia educativa en mi haber, puedo asegurar que se trata de una actividad muy interesante.