

# **Programa de teoría**

## **Parte I. Estructuras de Datos.**

### **→ 1. Abstracciones y especificaciones.**

2. Conjuntos y diccionarios.
3. Representación de conjuntos mediante árboles.
4. Grafos.

## **Parte II. Algorítmica.**

1. Análisis de algoritmos.
2. Divide y vencerás.
3. Algoritmos voraces.
4. Programación dinámica.
5. Backtracking.
6. Ramificación y poda.

A.E.D.

Tema 1. Abstracciones y especificaciones.

1

# **PARTE I: ESTRUCTURAS DE DATOS**

## **Tema 1. Abstracciones y Especificaciones.**

- 1.1. Abstracciones, tipos y mecanismos.
- 1.2. Especificaciones informales.
- 1.3. Especificaciones formales.
  - 1.3.1. Método axiomático (o algebraico).
  - 1.3.2. Método constructivo (u operacional).

A.E.D.

Tema 1. Abstracciones y especificaciones.

2

# 1.1. Abstracciones, tipos y mecanismos.

```
290 DIM b$(22,2): FOR m=1 TO 14: FOR n=1 TO 2
300 LET s=INT (RND*22)+1
310 IF b$(s,1)=" " THEN LET b$(s,1)=g$(n,1): LET b$(s,2)=g$(n,2):
NEXT m: NEXT n: GO TO 330
320 GO TO 330
330 *INT(TZ$): LET d1=0: LET i1=0: LET us=.001
340 PRINT AT 20,2;d1: IF d1=275000 THEN LET d1=350000: PRINT AT
20,2: FLASH 1;d1:"CONSEGUIDO EL PLENO EN ":i1n:" veces": PRINT
#: "Pulse una tecla para empezar": GO SUB 440: GO SUB 440: GO SUB
440: PAUSE 0: GO TO 330
350 INKEY n: IF n=22 OR n<1 THEN GO TO 350
360 IF i(n)=1 THEN GO TO 350
370 LET k=n: GO SUB 700
380 INKEY m: IF m=22 OR m<1 OR m=n THEN GO TO 380
390 IF i(m)=1 THEN GO TO 380
400 LET k=n: GO SUB 700
410 LET k=i(n)+1: IF b$(n)=b$(m) THEN LET d1=d1+25000: PAPER 3: LET
k=n: GO SUB 720: PAPER 3: LET k=n: GO SUB 720: LET k(n)=1: LET
i(m)=1: GO SUB 440: GO SUB 450: GO TO 340
420 BRIGHT 1: PAUSE #4: PAUSE #4: LET c1(n): LET c2(n): PRINT AT
i,c1$(n,8):AT i+1,c1$(n,14):AT i+2,c1$(n,20): PRINT AT
i,c1$(n,7 TO 8):AT i+1,c1$(n,13 TO 14):AT i+2,c1$(n,19 TO 20):
BEEP/.01,-10: PRINT @$(n): BEEP -.02,0
430 LET i1(n): LET c2(n): PRINT AT i,c2$(n,8):AT
i+1,c2$(n,14):AT i+2,c2$(n,20): PRINT AT i,c2$(n,7 TO 8):AT
i+1,c2$(n,13 TO 14):AT i+2,c2$(n,19 TO 20): BEEP .03,-10: PRINT
@$(n): BEEP .02,0: BRIGHT 0: GO TO 350
```

procedure ordenar(a: array;  
var b: array);

type persona;

class pila;

**Abstraer:** Eliminar lo irrelevante y quedarnos con lo realmente importante.

¿Qué es lo importante?

A.E.D.

3

# 1.1. Abstracciones, tipos y mecanismos.

## MECANISMOS DE ABSTRACCIÓN

**Abstracción por especificación:** Sólo necesitamos conocer qué va a hacer el procedimiento y no cómo funciona.  
(Encapsulación)

**Abstracción por parametrización:** Un algoritmo, un tipo, o una variable se definen en base a unos parámetros.  
(Genericidad)

A.E.D.

4

## **1.1. Abstracciones, tipos y mecanismos.**

### **TIPOS DE ABSTRACCIONES**

- Abstracciones funcionales → Rutinas, funciones, procedimientos
- Abstracciones de datos → Tipos Abstractos de Datos (TAD)
- Abstracciones de iteradores → Iteradores

A.E.D.

Tema 1. Abstracciones y especificaciones.

5

## **1.1. Abstracciones, tipos y mecanismos.**

### **TIPO ABSTRACTO DE DATOS:**

**TIPO DE DATOS:**

**ESTRUCTURA DE DATOS:**

A.E.D.

Tema 1. Abstracciones y especificaciones.

6

## 1.1. Abstracciones, tipos y mecanismos.

### Ejemplos

- **TAD:** Enteros
- **Tipo de datos:** Tipo **integer** de Pascal, o tipo **int** de C
- **Estructura de datos:** Representación mediante enteros de 16 bits, 32 bits, listas de dígitos (enteros largos), etc.

A.E.D.

Tema 1. Abstracciones y especificaciones.

7

## 1.1. Abstracciones, tipos y mecanismos

- La evolución de los lenguajes de programación tiende a introducir cada vez más abstracciones.

Lenguajes  
de bajo nivel

Lenguajes  
estructurados

Lenguajes  
orientados a objetos



(Basic, Fortran,  
Ensamblador, ...)

(Pascal, C,  
Modula, ADA, ...)

(Smalltalk, C++,  
Java, Eiffel, ...)

A.E.D.

Tema 1. Abstracciones y especificaciones.

8

## 1.1. Abstracciones, tipos y mecanismos

- La evolución de los lenguajes de programación tiende a introducir cada vez más abstracciones.
- Soporte de TAD:
  - **Lenguajes estructurados (tipos definidos por el usuario):** Los datos y las operaciones van aparte.
  - **Lenguajes orientados a objetos (clases):** Los datos y las operaciones constituyen una unidad.

A.E.D.

Tema 1. Abstracciones y especificaciones.

9

## 1.1. Abstracciones, tipos y mecanismos.

### Pascal

---

**type**

```
Pila = register
  tope: integer;
  datos: array [1..10] of integer;
end;
```

```
proc push (i: integer; var p:Pila);
```

```
proc pop (var p: Pila);
```

```
func top (p: Pila): integer;
```

### ObjectPascal

---

**type**

```
Pila = class
  private:
    tope: integer;
    datos: array [1..10] of integer;
```

```
public:
```

```
  proc push (i: integer);
```

```
  proc pop;
```

```
  func top: integer;
```

```
end;
```

A.E.D.

Tema 1. Abstracciones y especificaciones.

10

## 1.1. Abstracciones, tipos y mecanismos

### Pascal

```
var
  p1, p2: Pila;
  i: integer;

begin
  push(10, p1);
  push(20, p1);
  pop(p1);
  i:= top(p1);
  p1.tope:= 243;
  i:= top(p1);
  ...
```

### ObjectPascal

```
var
  p1, p2: Pila;
  i: integer;

begin
  p1.push(10);
  p1.push(20);
  p1.pop;
  i:= p1.top;
  p1.tope:= 243;
  i:= p1.top;
  ...
```

→ **Error de compilación:**  
"tope" es privado

A.E.D.

Tema 1. Abstracciones y especificaciones.

11

## 1.1. Abstracciones, tipos y mecanismos.

### Especificaciones: Tipos de notaciones

- Notaciones informales.
- Notaciones formales.
  - Algebraicas (o axiomáticas).
  - Operacionales (o constructivas).

A.E.D.

Tema 1. Abstracciones y especificaciones.

12

## 1.2. Especificaciones informales.

### 1.2.1. Abstracciones funcionales.

#### Notación

**Operación** <nombre> (**ent** <id>: <tipo>; <id>: <tipo>, ..., **sal** <tipo>)

**Requiere:** Establece restricciones de uso.

**Modifica:** Identifica los datos de entrada que se modifican (si existe alguno).

**Calcula:** Descripción textual del comportamiento de la operación.

## 1.2. Especificaciones informales.

### 1.2.1. Abstracciones funcionales.

**Ejemplo 1:** Eliminar la repetición en los elementos de un array.

**Operación** QuitarDuplic (**ent**  $a$ : array [entero])

**Modifica:**  $a$

**Calcula:** Quita los elementos repetidos de  $a$ . El límite inferior del array no varía, pero sí lo puede hacer el superior.

**Ejemplo 2:** Buscar un elemento en un array de enteros.

**Operación** Buscar (**ent**  $a$ : array [entero];  $x$ : entero; **sal**  $i$ : entero)

**Requiere:**  $a$  debe estar ordenado de forma ascendente.

**Calcula:** Si  $x$  está en  $a$ , entonces  $i$  debe contener el valor del índice de  $x$  tal que  $a[i] = x$ . Si  $x$  no está en  $a$ , entonces  $i = \text{sup} + 1$ , donde  $\text{sup}$  es el índice superior del array  $a$ .

## 1.2.1. Abstracciones funcionales.

**Generalización:** una operación está definida independientemente de cual sea el tipo de sus parámetros.

**Ejemplo 3:** Eliminar la repetición en los elementos de un array.

**Operación** QuitarDuplic [T: tipo](ent a: array [T])

**Requiere:** T debe tener una operación de comparación MenorQue(ent T, T; sal booleano).

**Modifica:** a

**Calcula:** Quita los elementos repetidos de a. El límite inferior del array no varía, pero sí lo puede hacer el superior.

**Ejemplo 4:** Buscar un elemento en un array de enteros.

**Operación** Buscar [T: tipo](ent a: array [T]; x: T; sal i: entero)

**Requiere:** T debe tener dos operación de comparación MenorQue(ent T, T; sal bool), Igual(ent T, T; sal bool). a debe estar ordenado de forma ascendente.

**Calcula:** Si x está en a, entonces i debe contener ...

A.E.D.

15

Tema 1. Abstracciones y especificaciones.

## 1.2.1. Abstracciones funcionales.

- Ejemplo de especificación informal de funciones:

### Especificación OpenCV

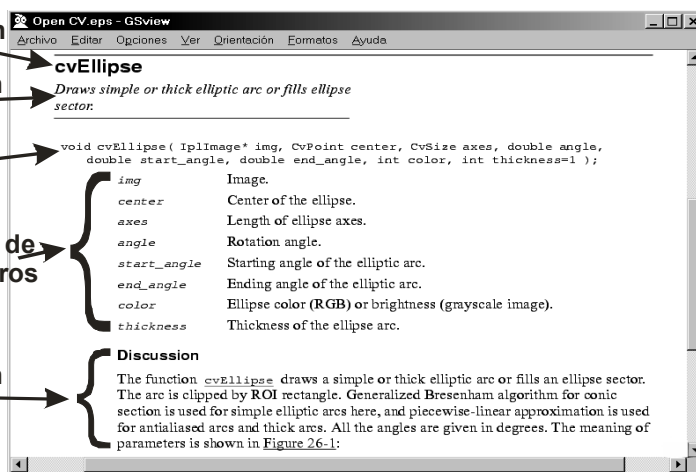
Nombre de la operación

Descripción corta

Sintaxis

Explicación de los parámetros

Descripción detallada



A.E.D.

16

Tema 1. Abstracciones y especificaciones.



## 1.2. Especificaciones informales.

### 1.2.2. Abstracciones de datos.

#### Notación

**TAD** <nombre\_tipo> **es** <lista\_operaciones>

#### Descripción

Descripción textual del tipo

#### Operaciones

Especificación informal de las operaciones de la lista anterior

**Fin** <nombre\_tipo>.

A.E.D.

Tema 1. Abstracciones y especificaciones.

17

### 1.2.2. Abstracciones de datos.

**TAD** CjtoEnteros **es** Vacío, Insertar, Suprimir, Miembro, EsVacío, Unión, Intersección, Cardinalidad

#### Descripción

Los CjtoEnteros son conjuntos matemáticos modificables, que almacenan valores enteros.

#### Operaciones

**Operación** Vacío (**sal** CjtoEnteros)

**Calcula:** Devuelve un conjunto de enteros vacío.

**Operación** Insertar (**ent** c: CjtoEnteros; x: entero)

**Modifica:** c.

**Calcula:** Añade x a los elementos de c. Después de la inserción, el nuevo conjunto es  $c \cup \{x\}$ .

...

**Fin** CjtoEnteros.

A.E.D.

Tema 1. Abstracciones y especificaciones.

18

## 1.2.2. Abstracciones de datos.

**TAD** ListaEnteros **es** Crear, Insertar, Primero, Ultimo, Cabeza, Cola, EsVacío, Igual

### Descripción

Las ListaEnteros son listas de enteros modificables. Las listas se crean con las operaciones Crear e Insertar...

### Operaciones

**Operación** Crear (**sal** ListaEnteros)

**Calcula:** Devuelve una lista de enteros vacía.

**Operación** Insertar (**ent** /: ListaEnteros; x: entero)

**Modifica:** /.

**Calcula:** Añade x a la lista / en la primera posición.

...

**Fin** ListaEnteros.

A.E.D.

19

Tema 1. Abstracciones y especificaciones.

## 1.2.2. Abstracciones de datos.

- **Generalización (parametrización de tipo):** El tipo se define en función de otro tipo pasado como parámetro.
- Útil para definir tipos **contenedores** o **colecciones**. P. ej. Listas, pilas, colas, arrays, conjuntos, etc.
- En lugar de:  
ListaEnteros  
ListaCadenas  
ListaReales  
....  
• Tenemos:  
Lista[T]
- Instanciación: Lista[entero], Lista[cadena],...

A.E.D.

20

Tema 1. Abstracciones y especificaciones.

## 1.2.2. Abstracciones de datos.

**TAD** Conjunto[T: tipo] es Vacío, Insertar, Suprimir, Miembro, EsVacío, Unión, Intersección, Cardinalidad

### Descripción

Los Conjunto[T] son conjuntos matemáticos modificables, que almacenan valores de tipo T.

### Operaciones

**Operación** Vacío (**sal** Conjunto[T])

...

**Operación** Insertar (**ent** c: Conjunto[T]; x: T)

...

**Operación** Suprimir (**ent** c: Conjunto[T]; x: T)

...

**Operación** Miembro (**ent** c: Conjunto[T]; x: T; **sal** booleano)

...

**Fin** Conjunto.

A.E.D.

21

Tema 1. Abstracciones y especificaciones.

## 1.2.2. Abstracciones de datos.

**TAD** Lista[T] es Crear, Insertar, Primero, Ultimo, Cabeza, Cola, EsVacío, Igual

### Descripción

Las Lista[T] son listas modificables de valores de tipo T. Las listas se crean con las operaciones Crear e Insertar...

### Operaciones

**Operación** Crear (**sal** Lista[T])

...

**Operación** Insertar (**ent** l: Lista[T]; x: entero)

...

**Operación** Primero (**ent** l: Lista[T]; **sal** Lista[T])

...

**Fin** Lista.

A.E.D.

22

Tema 1. Abstracciones y especificaciones.

## 1.2.2. Abstracciones de datos.

- En C++ es posible definir tipos parametrizados.
- Plantillas **template**:

```
template <class T>
class Pila {
    private:
        int tope;
        int maxDatos;
        T *datos;
    public:
        Pila (int maximo);
        Push (T valor);
        T Pop ();
}
```

A.E.D.

Tema 1. Abstracciones y especificaciones.

23

## 1.2. Especificaciones informales.

### 1.2.3. Abstracciones de iteradores.

- **Ejemplo:** Sobre el TAD CjtoEnteros queremos añadir operaciones para calcular la suma, el producto, ...

**Operación** suma\_conj (**ent** c: CjtoEnteros; **sal** entero)

**Calcula:** Devuelve la suma de los elementos de c.

....

**Operación** producto\_conj (**ent** c: CjtoEnteros; **sal** entero)

....

**Operación** varianza\_conj (**ent** c: CjtoEnteros; **sal** real)

....

A.E.D.

Tema 1. Abstracciones y especificaciones.

24

### 1.2.3. Abstracciones de iteradores.

- Necesitamos abstracciones de la forma:  
**para cada** elemento  $i$  del conjunto  $A$   
**hacer** acción sobre  $i$   
**para cada** elemento  $i$  de la lista  $L$   
**hacer** acción sobre  $i$   
**para cada**  $i$  de la cola  $C$  **tal que**  $P(i)$   
**hacer** acción sobre  $i$   
 $D :=$  **Seleccionar** todos los  $i$  de  $C$  **tal que**  $P(i)$
- Abstracción de **iteradores**: permiten definir un recorrido abstracto sobre los elementos de una colección.

### 1.2.3. Abstracciones de iteradores.

- La abstracción de iteradores no es soportada por los lenguajes de programación.
- **Posibles definiciones:**
  - Como una abstracción funcional:

**Iterador** ParaTodoHacer [T: tipo] (**ent** C: Conjunto[T]; *accion*: Operacion)

**Requiere:** *accion* debe ser una operación que recibe un parámetro de tipo T y no devuelve nada, *accion(ent T)*.

**Calcula:** Recorre todos los elementos  $c$  del conjunto  $C$ , aplicando sobre ellos la operación *accion(c)*.

### 1.2.3. Abstracciones de iteradores.

– Como una abstracción de datos:

**Tipoliterador** IteradorPreorden [T: tipo] **es** Iniciar, Actual, Avanzar, EsUltimo

#### Descripción

Los valores de tipo IteradorPreorden[T] son iteradores definidos sobre árboles binarios de cualquier tipo T. Los elementos del árbol son devueltos en preorden. El iterador se debe inicializar con Iniciar.

#### Operaciones

**Operación** Iniciar (**ent** A: ArbolBinario[T]; **sal** IteradorPreorden)

**Calcula:** Devuelve un iterador nuevo, colocado sobre la raíz de A.

**Operación** Actual (**ent** *iter*: IteradorPreorden; **sal** T)

...

**Fin** IteradorPreorden.

### 1.2.3. Abstracciones de iteradores.

```
var
  A: ArbolBinario[T];
  i: IteradorPreorden[T];
begin
  ...
  i:= Iniciar(A);
  while Not EsUltimo(i) do begin
    Acción sobre Actual(i);
    i:= Avanzar(i);
  end;
  ...
```

### 1.3. Especificaciones formales.

- Las especificaciones en lenguaje natural son ambiguas e imprecisas.
- **Especificaciones formales:** definen un TAD o una operación de manera precisa, utilizando un lenguaje matemático.
- Ventajas de una especificación formal:
  - **Prototipado.** Las especificaciones formales pueden llegar a ser ejecutables.
  - **Corrección del programa.** Verificación automática y formal de que el programa funciona correctamente.
  - **Reusabilidad.** Posibilidad de usar la especificación formal en distintos ámbitos.

### 1.3. Especificaciones formales.

#### Notación

La descripción formal constará de cuatro partes:

- **NOMBRE.** Nombre genérico del TAD.
- **CONJUNTOS.** Conjuntos de datos que intervienen en la definición.
- **SINTAXIS.** Signatura de las operaciones definidas.
- **SEMÁNTICA.** Indica el significado de las operaciones, cuál es su resultado.

## 1.3. Especificaciones formales.

- **Sintaxis:**

<nombre\_operación> : <conj\_dominio> → <conj\_resultado>

- Los distintas notaciones formales difieren en la forma de definir la semántica:
  - **Método axiomático o algebraico.** Se establece el significado de las operaciones a través de relaciones entre operaciones (*axiomas*). Significado implícito de las operaciones.
  - **Método constructivo u operacional.** Se define cada operación por sí misma, independientemente de las otras, basándose en un modelo subyacente. Significado explícito de las operaciones.

### 1.3.1. Método axiomático (o algebraico).

- La semántica de las operaciones se define a través de un conjunto de **axiomas**.
- Un axioma es una regla que establece la igualdad de dos expresiones:

<Expresión 1> = <Expresión 2>

- Por ejemplo:

Suma (dos, dos) = Producto (dos, dos)

Tope (Push (x, pila)) = x

And (verdadero, b) = verdadero



### 1.3.1. Método axiomático (o algebraico).

- ¿Qué axiomas introducir en la semántica?
- Los axiomas deben ser los necesarios para satisfacer dos propiedades:
  - **Compleitud:** Los axiomas deben ser los suficientes para poder deducir el significado de cualquier expresión.
  - **Corrección:** A partir de una expresión sólo se puede obtener un resultado.

A.E.D.

Tema 1. Abstracciones y especificaciones.

33

### 1.3.1. Método axiomático (o algebraico).

- **Ejemplo:** TAD Natural de los números naturales.

#### NOMBRE

Natural

#### CONJUNTOS

N Conjunto de naturales

B Conjunto de booleanos {true, false}

#### SINTAXIS

cero:  $\rightarrow N$

sucesor:  $N \rightarrow N$

esCero:  $N \rightarrow B$

igual:  $N \times N \rightarrow B$

suma:  $N \times N \rightarrow N$

A.E.D.

Tema 1. Abstracciones y especificaciones.

34

### 1.3.1. Método axiomático (o algebraico).

#### SEMANTICA

$\forall m, n \in \mathbb{N}$

1. esCero (cero) = true
2. esCero (sucesor (n)) = false
3. igual (cero, n) = esCero (n)
4. igual (sucesor (n), cero) = false
5. igual (sucesor (n), sucesor (m)) = igual (n, m)
6. suma (cero, n) = n
7. suma (sucesor (m), n) = sucesor (suma (m, n))

A.E.D.

Tema 1. Abstracciones y especificaciones.

35

### 1.3.1. Método axiomático (o algebraico).

- **Ejecución de una especificación algebraica:**  
aplicar sucesivamente las reglas de la semántica hasta que no se puedan aplicar más.
- **Ejemplo.** ¿Cuánto valen las siguientes expresiones?
  - a) suma (suma(sucesor(cero), cero), sucesor (cero) )
  - b) igual (sucesor (sucesor (cero)), suma (suma (sucesor (cero), cero), sucesor (cero) ) )

A.E.D.

Tema 1. Abstracciones y especificaciones.

36

### 1.3.1. Método axiomático (o algebraico).

- Supongamos un TAD,  $T$ .
- **Tipos de operaciones:**
  - **Constructores.** Conjunto mínimo de operaciones del TAD, a partir del cual se puede obtener cualquier valor del tipo  $T$ .  
 $c1: \rightarrow T, c2: V \rightarrow T, c3: V_1x...xV_n \rightarrow T$
  - **Modificación.** A partir de un valor del tipo obtienen otro valor del tipo  $T$ , y no son constructores.  
 $m1: T \rightarrow T, m2: TxV \rightarrow T, m3: V_1x...xV_n \rightarrow T$
  - **Consulta.** Devuelven un valor que no es del tipo  $T$ .  
 $o1: T \rightarrow V, o2: TxV \rightarrow V', o3: V_1x...xV_n \rightarrow V_{n+1}$

### 1.3.1. Método axiomático (o algebraico).

- La ejecución de una expresión acaba al expresarla en función de los constructores.
- ¿Cómo garantizar que una especificación es completa y correcta?
- Definir los axiomas suficientes para relacionar las operaciones de modificación y consulta con los constructores.
- No incluir axiomas que se puedan deducir de otros existentes.

### 1.3.1. Método axiomático (o algebraico).

- **Ejemplo:** Especificación del TAD genérico pila.

#### NOMBRE

Pila [T]

#### CONJUNTOS

S Conjunto de pilas

T Conjunto de elementos que pueden ser almacenados

B Conjunto de booleanos {true, false}

#### SINTAXIS

crearPila:  $\rightarrow$  S

esVacía: S  $\rightarrow$  B

pop: S  $\rightarrow$  S

tope: S  $\rightarrow$  T

push: T x S  $\rightarrow$  S

A.E.D.

Tema 1. Abstracciones y especificaciones.

39

### 1.3.1. Método axiomático (o algebraico).

- En el caso de **tope: S  $\rightarrow$  T**, ¿qué pasa si la pila está vacía?
- Se puede añadir un conjunto de mensajes en **CONJUNTOS**, de la forma:

M Conjunto de mensajes {"Error. La pila está vacía"}

- Y cambiar en la parte de **SINTAXIS** la operación **tope:**

tope: S  $\rightarrow$  T U M

A.E.D.

Tema 1. Abstracciones y especificaciones.

40

### 1.3.1. Método axiomático (o algebraico).

	crearPila	push (t, s)
esVacía ( )	esVacía(crearPila) =	esVacía(push(t, s)) =
pop ( )	pop(crearPila) =	pop(push(t, s)) =
tope ( )	tope(crearPila) =	tope(push(t, s)) =

### 1.3.1. Método axiomático (o algebraico).

#### SEMANTICA

$\forall t \in T; \forall s \in S$

1. esVacía (crearPila) = true
2. esVacía (push (t, s)) = false
3. pop (crearPila) = crearPila
4. pop (push (t, s)) = s
5. tope (crearPila) = "Error. La pila está vacía"
6. tope (push (t, s)) = t

### 1.3.1. Método axiomático (o algebraico).

- Calcular:
  - a) `pop(push(3, push(2, pop(crearPila))))`
  - b) `tope(pop(push(1, push(2, crearPila))))`
- Añadir una operación **Igual** para comparar dos pilas.
- Modificar la operación **pop**, para que devuelva un error si la pila está vacía.
- ¿Cómo hacer que la operación **pop** devuelva el tope de la pila y al mismo tiempo lo saque de la pila?

A.E.D.

Tema 1. Abstracciones y especificaciones.

43

### 1.3.1. Método axiomático (o algebraico).

- Para facilitar la escritura de la expresión del resultado en la semántica, se pueden emplear condicionales de la forma:  
**SI** <condición>  $\Rightarrow$  <valor si cierto> | <valor si falso>
- **Ejemplo:** Especificación algebraica del TAD bolsa.

#### **NOMBRE**

Bolsa[!]

#### **CONJUNTOS**

S Conjunto de bolsas

I Conjunto de elementos

B Conjunto de booleanos {true, false}

N Conjunto de naturales

#### **SINTAXIS**

bolsaVacía:  $\rightarrow$  S

poner: S x I  $\rightarrow$  S

esVacía: S  $\rightarrow$  B

cuántos: S x I  $\rightarrow$  N

A.E.D.

Tema 1. Abstracciones y especificaciones.

44

### **1.3.1. Método axiomático (o algebraico).**

- Incluir una operación quitar, que saque un elemento dado de la bolsa.
- ¿Y si queremos que los saque todos?
- Incluir una operación Igual, de comparación de bolsas.

### **1.3.1. Método axiomático (o algebraico).**

#### **Conclusiones:**

- Las operaciones no se describen de manera explícita, sino **implícitamente** relacionando el resultado de unas con otras.
- La construcción de los axiomas se basa en un **razonamiento inductivo**.
- ¿Cómo se podría especificar, por ejemplo, un procedimiento de ordenación?

### 1.3.2. Método constructivo (operacional).

- Para cada operación, se establecen las precondiciones y las postcondiciones.
- **Precondición:** Relación que se debe cumplir con los datos de entrada para que la operación se pueda aplicar.
- **Postcondición:** Relaciones que se cumplen después de ejecutar la operación.

### 1.3.2. Método constructivo (operacional). Notación

En la semántica, para cada operación <nombre>:

**pre-<nombre>**(<param\_entrada>):= <condición\_lógica>

**post-<nombre>**(<param\_entrada>;<param\_salida>):=  
<condición\_lógica>

- **Ejemplo:** operación **máximo**, que tiene como entrada dos números reales y da como salida el mayor de los dos.

**máximo:**  $\mathbf{R \times R \rightarrow R}$  (SINTAXIS)

**pre-máximo**( $x, y$ ) ::= true (SEMANTICA)

**post-máximo**( $x, y; r$ ) ::=  $(r \geq x) \wedge (r \geq y) \wedge (r=x \vee r=y)$



### 1.3.2. Método constructivo (operacional).

- **Ejemplo:** operación **máximo** sobre números reales, pero restringida a números positivos.

**máximop:**  $\mathbb{R} \times \mathbb{R} \rightarrow \mathbb{R}$

**pre-máximop**( $x, y$ ) ::=  $(x \geq 0) \wedge (y \geq 0)$

**post-máximop**( $x, y; r$ ) ::=  $(r \geq x) \wedge (r \geq y) \wedge (r=x \vee r=y)$

- ¿Qué sucedería si  $x$  o  $y$  no son mayores que 0?
- No se cumple la precondición  $\rightarrow$  No podemos asegurar que se cumpla la postcondición.

A.E.D.

Tema 1. Abstracciones y especificaciones.

49

### 1.3.2. Método constructivo (operacional).

- **Implementación** en C++ de pre- y post-condiciones.

**máximop:**  $\mathbb{R} \times \mathbb{R} \rightarrow \mathbb{R}$

**pre-máximop**( $x, y$ ) ::=  $(x \geq 0) \wedge (y \geq 0)$

**post-máximop**( $x, y; r$ ) ::=  $(r \geq x) \wedge (r \geq y) \wedge (r=x \vee r=y)$

```
float maximop (float x, float y)
{
    assert(x >= 0 && y >= 0); // Precondición
    double r;
    if (x > y) r = x; else r = y;
    assert(r >= x && r >= y && (r == x || r == y)); // Postcond
    return r;
}
```

A.E.D.

Tema 1. Abstracciones y especificaciones.

50

### 1.3.2. Método constructivo (operacional).

- **Otra posibilidad:** Definir un conjunto **M** (de mensajes de error) y cambiar la imagen. Modificar la sintaxis y la semántica:

**máximop2:  $R \times R \rightarrow R \cup M$**

**pre-máximop2**(x, y) ::= true

**post-máximop2**(x, y; r) ::=  $SI (x \geq 0) \wedge (y \geq 0)$

$\Rightarrow (r \geq x) \wedge (r \geq y) \wedge (r=x \vee r=y)$

| r = "Fuera de rango"

- ¿Cuál es la mejor opción?

A.E.D.

Tema 1. Abstracciones y especificaciones.

51

### 1.3.2. Método constructivo (operacional).

- ¿Cuál es la mejor solución?
- **La especificación como un contrato.**
- **Contrato de una operación:**  
Si se cumplen unas condiciones en los parámetros de entrada, entonces garantiza una obtención correcta del resultado.



A.E.D.

Tema 1. Abstracciones y especificaciones.

52

### 1.3.2. Método constructivo (operacional).

- Dos puntos de vista del contrato (especificación):
  - **Implementador. Obligación:** Cumplir la postcondición.  
**Derechos:** Sabe que se cumple la precondition.
  - **Usuario. Obligación:** Cumplir la precondition.  
**Derechos:** Sabe que se cumple la postcondición.
- **Idea:**
  - La operación no trata todos los casos de error, sino que hace uso de las precondiciones.
  - La responsabilidad de comprobar la precondition es del que usa la operación.

### 1.3.2. Método constructivo (operacional).

- ¿Cómo se pueden definir las pre/post-condiciones cuando el TAD es más complejo? Por ejemplo, para TAD colecciones.
- Necesitamos un **modelo subyacente**, en el cual se base la definición del TAD.
- No siempre se encuentra uno adecuado...
- **Ejemplo:** Para definir el TAD **Pila[T]**, definiremos el TAD **Lista[T]** por el método axiomático, y luego lo usaremos para definir el TAD pila con el método constructivo.

## 1.3.2. Método constructivo (operacional).

### NOMBRE

Lista[l]

### CONJUNTOS

- L Conjunto de listas
- I Conjunto de elementos
- B Conjunto de booleanos {true, false}
- N Conjunto de naturales
- M Conjunto de mensajes {"La lista está vacía"}

### SINTAXIS

crearLista:		→	L
formarLista:	I	→	L
concatenar:	L x L	→	L
último:	L	→	I U M
cabecera:	L	→	L
primero:	L	→	I U M
cola:	L	→	L
longitud:	L	→	N
esListaVacía:	L	→	B

A.E.D.

55

Tema 1. Abstracciones y especificaciones.

## 1.3.2. Método constructivo (operacional).

### SEMANTICA

$\forall i \in I; \forall a, b \in L$

1. último (crearLista) = "La lista está vacía"
2. último (formarLista (i)) = i
3. último (concatenar (a, b)) = SI esListaVacía (b)  $\Rightarrow$   
último (a) | último (b)
4. cabecera (crearLista) = crearLista
5. cabecera (formarLista (i)) = crearLista
6. cabecera (concatenar (a, b)) = SI esListaVacía (b)  $\Rightarrow$   
cabecera (a) | concatenar (a, cabecera (b))
7. primero (crearLista) = "La lista está vacía"
8. primero (formarLista (i)) = i
9. primero (concatenar (a, b)) = SI esListaVacía (a)  $\Rightarrow$   
primero (b) | primero (a)

A.E.D.

56

Tema 1. Abstracciones y especificaciones.

### 1.3.2. Método constructivo (operacional).

10. cola (crearLista) = crearLista
11. cola (formarLista (i)) = crearLista
12. cola (concatenar (a, b)) = SI esListaVacía (a)  $\Rightarrow$   
cola (b) | concatenar (cola (a), b)
13. longitud (crearLista) = cero
14. longitud (formarLista (i)) = sucesor (cero)
15. longitud (concatenar (a, b)) = suma (longitud (a),  
longitud (b))
16. esListaVacía (crearLista) = true
17. esListavacía (formarLista (i)) = false
18. esListaVacía (concatenar (a, b)) = esListaVacía (a)  
AND esListaVacía(b)

**Aserto invariante:** siempre que aparezca un mensaje a la entrada de una operación, la salida será el mismo mensaje.

A.E.D.

57

Tema 1. Abstracciones y especificaciones.

### 1.3.2. Método constructivo (operacional).

- Seguimos el ejemplo y aplicamos el método constructivo a la definición de **Pila[I]**, teniendo como modelo subyacente el tipo **Lista[I]**.

#### NOMBRE

Pila[I]

#### CONJUNTOS

S Conjunto de pilas

I Conjunto de elementos

B Conjunto de valores booleanos {true, false}

M Conjunto de mensajes {"La pila está vacía"}

#### SINTAXIS

crearPila:  $\rightarrow$  S

tope: S  $\rightarrow$  I U M

pop: S  $\rightarrow$  S U M

push: I x S  $\rightarrow$  S

esVacíaPila: S  $\rightarrow$  B

A.E.D.

58

Tema 1. Abstracciones y especificaciones.

### 1.3.2. Método constructivo (operacional).

#### SEMANTICA

$\forall i \in I; \forall s \in S; b \in B; r \in S; t \in I \cup M; p \in S \cup M$

1. pre-crearPila () ::= true
2. post-crearPila (s) ::= s = crearLista
3. pre-tope (s) ::= true
4. post-tope (s; t) ::= Si esListaVacía (s)  
                                           $\Rightarrow t = \text{"La pila está vacía"}$   
                                          | t = primero (s)
5. pre-pop (s) ::= true
6. post-pop (s; p) ::= Si esListaVacía (s)  
                                           $\Rightarrow p = \text{"La pila está vacía"}$   
                                          | p = cola (s)
7. pre-push (i, s) ::= true
8. post-push (i, s; r) ::= r = concatenar (formarLista (i), s)
9. pre-esVacíaPila (s) ::= true
10. post-esVacíaPila (s; b) ::= b = esListaVacía (s)

A.E.D.

59

### 1.3.2. Método constructivo (operacional).

#### NOMBRE

Pila[I]

#### CONJUNTOS

S Conjunto de pilas

I Conjunto de elementos

B Conjunto de valores booleanos {true, false}

#### SINTAXIS

crearPila:                     $\rightarrow$     S

tope:                         S  $\rightarrow$   I

pop:                         S  $\rightarrow$     S

push:                        I x S  $\rightarrow$  S

esVacíaPila:                S  $\rightarrow$     B

A.E.D.

60

### 1.3.2. Método constructivo (operacional).

#### SEMANTICA

$\forall i, t \in I; \forall s, r, p \in S; \forall b \in B$

1. pre-crearPila () ::= true
2. post-crearPila (s) ::= s = crearLista
3. pre-tope (s) ::= NOT esListaVacía (s)
4. post-tope (s; t) ::= t = primero (s)
5. pre-pop (s) ::= NOT esListaVacía (s)
6. post-pop (s; p) ::= p = cola (s)
7. pre-push (i, s) ::= true
8. post-push (i, s; r) ::= r = concatenar (formarLista (i), s)
9. pre-esVacíaPila (s) ::= true
10. post-esVacíaPila (s; b) ::= b = esListaVacía (s)

A.E.D.

61

Tema 1. Abstracciones y especificaciones.

### 1.3.2. Método constructivo (operacional).

- Seguimos el ejemplo y aplicamos el método constructivo a la definición de **Cola[I]**, teniendo como modelo subyacente el tipo **Lista[I]**.

#### NOMBRE

Cola[I]

#### CONJUNTOS

C Conjunto de colas

I Conjunto de elementos

B Conjunto de valores booleanos {true, false}

M Conjunto de mensajes {"La cola está vacía"}

#### SINTAXIS

crearCola:  $\rightarrow$  C

frente: C  $\rightarrow$  I U M

inserta: C  $\rightarrow$  C U M

resto: I x C  $\rightarrow$  C

esVacíaCola: C  $\rightarrow$  B

A.E.D.

62

Tema 1. Abstracciones y especificaciones.

### 1.3.2. Método constructivo (operacional).

- **Ejecución de la especificación:** comprobar precondiciones y postcondiciones de todas las operaciones de la expresión.
- **Ejemplos:** Pila[Natural]
  - a) tope (push (4, pop (push (2, crearPila) ) ) )
  - b) esVacíaPila (push (2, pop (crearPila) ) )

### 1.3.2. Método constructivo (operacional).

#### Conclusiones:

- La especificación constructiva está limitada por la necesidad de **modelos subyacentes**.
- **No confundir** especificación con implementación.
- Es más fácil incluir especificaciones constructivas en los programas (p.ej., mediante asertos).