Introducción a la Algorítmica y Programación (3300)

Prof. Ariel Ferreira Szpiniak - aferreira@exa.unrc.edu.ar
Departamento de Computación
Facultad de Cs. Exactas, Fco-Qcas y Naturales
Universidad Nacional de Río Cuarto

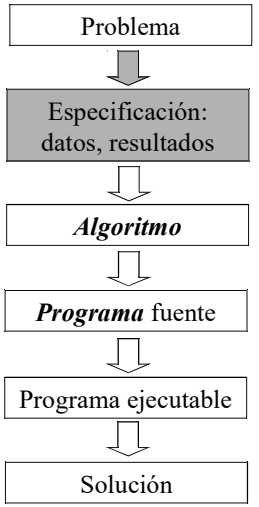
Teoría 19

Especificación

Secuencias. Notación

Especificación
Pasos para solucionar un problema

- ***Análisis***
  El problema debe ser claramente especificado y entendido.
- ***Diseño***
  Construcción de una solución general del problema.
- **Implementación**
  Traducción del algoritmo a un lenguaje de programación de alto nivel.
- **Compilación**
  Escritura del programa Pascal, usando en nuestro caso el Compilador Turbo Pascal 7.0.
- **Ejecución y Prueba**
  Corrida y prueba de funcionamiento del programa en la computadora.



Especificación
Introducción

Hasta ahora hemos visto que:

- *El problema debe estar bien definido.*
- *Los datos de entrada y los resultados (salida) deben ser precisamente descriptos.*

1. *Preguntas concernientes a la entrada:*
   - *¿Cuáles y cuántos son los valores de entrada?*
   - *¿Cuáles son valores válidos de entrada?*
2. *Preguntas concernientes a los resultados (salida):*
   - *¿Cuáles y cuántos son los valores del resultado?*
   - *¿Cuáles son valores válidos del resultado?*
   - *¿Cómo se llega a esos resultados (fórmulas, ecuaciones, etc.)?*
3. *Conocimiento o cuestiones Adicionales necesarias para la solución del problema (qué debo saber)*

Especificación
Introducción

Existen diferentes visiones acerca de como hacer para obtener **programas** que resuelvan **correctamente** los **problemas**. Un punto central del proceso es lograr **describir** el **problema** de la **manera** más **clara** posible, **sin ambigüedades**.

Actualmente la búsqueda de métodos de especificación de problemas es un importante campo de investigación dentro de las Ciencias de la Computación.

La visión que utilizaremos en este curso es la que plantea que un programa corresponde a una *transformación de datos del entorno*. Esta transformación se produce a partir de un contexto determinado por las *precondiciones*, el programa transforma los datos de entrada y debiera llegar al resultado esperado (salida) produciendo un nuevo contexto, caracterizado por las *poscondiciones*.

Especificación

Precondición / Poscondición / Definición

Especificar consiste en detallar cuidadosamente el problema, es decir, contestar a la pregunta *¿qué debo hacer?,* de manera clara y precisa.

Para lograr claridad y precisión es necesario un lenguaje formal, es decir, que la sintaxis (forma de escritura) y la semántica (su significado) del lenguaje estén perfectamente definidas.

Para especificar de manera formal utilizaremos la técnica de precondición/poscondición o precondición/definición, basada en la lógica de primer orden.

Especificación

Precondición / Poscondición / Definición

Definición: Especificar un problema significa explicitar de una manera formal, semiformal o informal, mediante alguna notación, la ***precondición*** del mismo y la ***poscondición*** *o su* ***definición***.

Especificación

Precondición / Poscondición / Definición

La ***precondición*** define cuales son las **restricciones** que deben cumplir los datos. Esto no implica que los valores de los mismos deban conocerse de antemano.

Tanto la ***precondición*** como la ***poscondición*** *o la* ***definición*** son predicados lógicos que deben satisfacerse si o si.

La ***precondición*** la expresaremos como: **{Pre-condición: Q}**

La ***poscondición*** la expresaremos como: **{Pos-condición: R}**

La ***definición*** la expresaremos como: **{Definición: D}**

La poscondición se utiliza cuando el módulo que se pretende desarrollar es una **acción** (modifica el entorno), y la **definición** cuando es una **función** (no debe modificar el entorno).

Especificación

Precondición / Poscondición / Definición

Supongamos que tenemos que desarrollar un módulo para resolver cierto problema, y ya hemos determinado los datos de entrada, de salida, y el conocimiento.

La precondición incluye a los objetos que son datos de entrada (parámetros de entrada) y describe los estados válidos de dichos objetos en los cuales el módulo puede realizarse.

La poscondición incluye a los objetos que son datos de entrada (parámetros de entrada) y a los de salida (parámetros de salida), y describe los estados finales al culminar el módulo, dando la relación entre la entrada y la salida.

Especificación
Notación

Para que la especificación pueda ser entendida por todos debemos ponernos de acuerdo en la notación a utilizar, al igual que lo hicimos con la notación para escribir algoritmos.

A continuación daremos la notación que sería deseable utilizar, aunque como recién estamos comenzando, entendemos que será un proceso en el cual iremos perfeccionando nuestras especificaciones, tratando de lograr el mayor nivel de formalidad posible, aunque en algunos casos tengamos que recurrir al lenguaje natural.

Lo importante de la especificación es que cuanto más trabajemos en ella, y mayor formalismo utilicemos, mejor entenderemos el problema a resolver y por lo tanto más fácil será construir algoritmos y razonar sobre los mismos.

Especificación
Notación básica

- Objetos sin restricciones: **{True}**
- Objetos con un valor inicial pero desconocido: **{a=A_0}**

(por ejemplo que fue obtenido anteriormente)

- Objeto con un valor inicial conocido: **{Iva=21}** (poco usual)
- Objeto con alguna restricción: **{a<0}**, **{num>=0}**, **{long>0}**, **{divisor≠0}**
- Varias restricciones sobre los datos: **{a=A_0 ∧ b>=0 ∧ long=$Long_0$}**

Nota: Los valores utilizados en las precondiciones no pueden ser utilizados en el programa ya que viven dentro del dominio de las especificaciones. Como tales, deberían ser usados para especificar resultados a partir de los valores iniciales.

Especificación
Ejemplos

Problema: Desarrollar un módulo que **dado** el lado de una baldosa, calcule e informe el área ocupada por la misma.

Especificación:

{**Pre-condición:** lado =$Lado_0$ ∧ lado>0}

{**Pos-condición:** area=$Area_0$ ∧ $Area_0$=$Lado_0$*$Lado_0$}

objeto — valor dado — objeto — valor que debería adquirir

Más resumido pero equivalente

{Pos-condición: area=$Lado_0$*$Lado_0$**}**

Problema: Desarrollar un módulo que **dado** el diámetro de un círculo, calcule e informe la superficie ocupada por el mismo.

{**Pre-condición:** }

{**Pos-condición:** }

Especificación
Ejemplos

Problema: Desarrollar un módulo que a partir del **precio bruto** de un producto, calcule el precio final del producto, es decir, le agregue el 21% de IVA.

Especificación:

{**Pre-condición:** Iva=21}

Primer valor del objeto

{**Pos-condición:** precioBruto= $precioBruto_0$ ∧ precioFinal=$precioBruto_0$+($precioBruto_0$*Iva)/100}

O más resumido, si no interesa el valor final del objeto precioBruto

{**Pos-condición:** precioFinal=$precioBruto_0$+($precioBruto_0$*Iva)/100}

Especificación

Ejemplos

Problema: Desarrollar un módulo que detecte si un número ingresado es positivo o negativo y almacene el resultado en una variable.

Especificación:
{**Pre-condición: True**}

Uno estaría tentado a decir que:

Primer valor del objeto

{**Pos-condición**: num=Num_0 ∧ (si num es menor que cero entonces esPositivo=False o si num el mayor o igual que cero entonces esPositivo=True) }

Incorrecto!

Especificación

Implicación lógica: Discusión

Formalizando....

{**Pos-condición**: num=num_0 ∧ ((num<0 → esPositivo=False) ∨ (num>=0 → esPositivo=True))}

Dijimos que tanto la pre como la poscondición deben ser predicados verdaderos.

¿Qué ocurre si el antecedente en falso?

La implicación es...

¿Está bien un programa que lea en **num** el valor **- 5**, por ejemplo, y deje esPositivo con valor True? ¿Satisface la poscondición?

Especificación

Implicación lógica: Discusión

- En los enunciados informales en lenguaje natural es muy frecuente encontrar la palabra "entonces".
- Sin embargo, la implicación lógica "→" (de la lógica matemática) no es equivalente al "entonces“ del lenguaje natural.

Ejemplo

(num<0 → esPositivo=False) ∨ (num>=0 → esPositivo=True)

- Utilizando la equivalencia lógica entre (p→q) y (¬p ∨ q), tenemos que:

(¬num<0 ∨ esPositivo=False) ∨ (¬num>=0 ∨ esPositivo=True)

- Lo cual es equivalente a:

(num>=0 ∨ esPositivo=False) ∨ (num<0 ∨ esPositivo=True)

Especificación

Implicación lógica: Discusión

(num>=0 ∨ esPositivo=False) ∨ (num<0 ∨ esPositivo=True)

- Pero resulta que éste predicado se verifica, por ejemplo, con un estado en el que num sea negativo y esPositivo quede con el valor Verdadero.
- En tal caso los valores de verdad serían:

(F ∨ T) ∨ (T ∨ F)

- Aplicando las definiciones de disyunción, podemos ver que lo anterior es equivalente a T (verdadero).
- Puede comprobarse fácilmente que la traducción correcta sería:

(num<0 ∧ esPositivo=False) ∨ (num>=0 ∧ esPositivo=True)

Especificación
Ejemplos

Lo correcto sería:

1. {**Pos-condición**: num=Num_0 ∧ ((num<0 ∧ esPositivo=False) ∨ (num>=0 ∧ esPositivo=True))**}**

2. **{Pos-condición**: num=Num_0 ∧ ((num<0 ∧ ¬esPositivo) ∨ (num>=0 ∧ esPositivo))**}**

3. **{Pos-condición**: num=Num_0 ∧ (esPositivo=Num_0>=0)**}**

Es decir que la Especificación quedaría de la siguiente manera:

{Pre-condición: True}

{Pos-condición: num=Num_0 ∧ (esPositivo=Num_0>=0)**}**

Secuencias
Noción de Secuencias

Diremos que un conjunto de objetos está organizado en forma de secuencia si es posible definir las siguientes nociones:

1. **Primer elemento de la secuencia**: permite el acceso ulterior a todos los demás elementos de la secuencia.

2. **Relación de sucesión entre elementos**: todo elemento de la secuencia (excepto el último) precede a uno de los demás elementos (su siguiente).

3. **Caracterización del fin de la secuencia**: debe existir una manera de saber cuando se termina la secuencia.

Notación de Secuencias

$<>$ = secuencia vacía
$[e_1, e_2, ..., e_n]$ = secuencia de *n* elementos

En caso de secuencias de caracteres:
$['e_1', 'e_2', ..., 'e_n'] = "e_1e_2...e_n"$ = secuencia de *n* caracteres

Si Σ es un conjunto de elementos (N, Z, R, Caracteres, etc.):

Σ^+ = secuencias no vacías de elementos de Σ

Σ^* = secuencias de elementos de $\Sigma^* = \Sigma^+ \cup \{<>\}$

Nota: esta notación se utiliza para especificar con mayor claridad.

Operaciones sobre Secuencias

Concatenar dos secuencias: $\alpha \& \beta$

$\& : \Sigma^* \times \Sigma^* \rightarrow \Sigma^*$

Ej: [1,2,3]&[8,9] representa [1,2,3,8,9] - "abc"&"zy" representa "abczy"

Concatenar un elemento con una secuencia: $e \circ \alpha$

$\circ : \Sigma \times \Sigma^* \rightarrow \Sigma^+$

Ej: 'a'∘['l', 't', 'o'] representa ['a', 'l', 't', 'o'] - 6∘[3, 9] representa [6, 3, 9]

Concatenar una secuencia con un elemento: $\alpha \bullet e$

$\bullet : \Sigma^* \times \Sigma \rightarrow \Sigma^+$

Ej: "alt" • 'o' representa "alto" - [6, 3] • 9 representa [6, 3, 9]

Nota: estas operaciones se utilizan para especificar con mayor claridad.

Cuantificadores sobre Secuencias

Existencial

$(\exists x:\ rango:\ propiedad)$

$(\exists x:\ x \in \alpha:\ x=\text{‘i’})$

Ejemplos: $(\exists x:\ x \in [\text{‘H’,‘i’,‘j’,‘o’}]:\ x=\text{‘i’})$

$(\exists x:\ x \in [\text{‘H’,‘O’,‘j’,‘O’}]:\ x=\text{‘i’})$

$(\exists x:\ x \in [6,2,5,7,3,9,4]:\ x>=7)$

Donde la evaluación de *propiedad* resulta ser un valor lógico.
El cuantificador puede tener más de una variable ligada (x, y).

Cuantificadores sobre Secuencias

Universal

$(\forall x:\ rango:\ propiedad)$

$(\forall x:\ x \in \alpha:\ x=\text{‘a’})$

Ejemplos: $(\forall\ x:\ x \in [\text{‘H’,‘i’,‘j’,‘o’}]:\ x=\text{‘i’})$

$(\forall\ x:\ x \in [\text{‘H’,‘O’,‘j’,‘O’}]:\ x=\text{‘i’})$

$(\forall\ x:\ x \in [6,2,5,7,3,9,4]:\ x>=0)$

Donde la evaluación de *propiedad* resulta ser un valor lógico.
El cuantificador puede tener más de una variable ligada (x, y).

Operadores sobre Secuencias

Contatoria

$(\#x:\ rango:\ propiedad)$

Ejemplos: $(\#x:\ x \in \alpha:\ x>0)$

$(\#x:\ x \in [6,2,5,7,3,9,4]:\ 2<x<7)$

$(\#x:\ x \in [\text{‘H’,‘O’,‘j’,‘O’}]:\ x=\text{‘O’})$

Máximo

$(\Uparrow x:\ rango:\ término)$

Ejemplos: $(\Uparrow x:\ x \in \alpha \wedge x>0:\ x)$

$(\Uparrow x:\ x \in [6,2,5,7,3,9,4]:\ x)$

$(\Uparrow x:\ x \in [6,2,5,7,3,9,4] \wedge 2<x<7:\ x)$

Donde la evaluación de *propiedad* resulta ser un valor lógico.
El cuantificador puede tener más de una variable ligada (x, y).

Operadores sobre Secuencias

Mínimo

$(\Downarrow x:\ rango:\ término)$

Ejemplos: $(\Downarrow x:\ x \in \alpha \wedge x>0:\ x)$

$(\Downarrow x:\ x \in [6,2,5,7,3,9,4]:\ x)$

$(\Downarrow x:\ x \in [6,2,5,7,3,9,4] \wedge 2<x<7:\ x)$

Donde la evaluación de *propiedad* resulta ser un valor lógico y *término* un número.
El cuantificador puede tener más de una variable ligada (x,y).

Operadores sobre Secuencias

Sumatoria $(\sum x:$ *rango:* *término*$)$

Ejemplos: $(\sum x: x \in \alpha: x)$

$(\sum x: x \in [-2,5,-4,3,4,-8] \wedge x>0: x)$

$(\sum x: x \in [6,2,5,7,3,9,4] \wedge 2<x<7: x^2)$

$(\sum x: x \in \alpha \wedge 2<x<7 \wedge (x \bmod 2)=1: x)$

$(\sum x: x \in \alpha \wedge x>0 : 1) = (\#x: x \in \alpha: x>0)$

Productoria

$(\prod x:$ *rango:* *término*$)$

Ejemplos: $(\prod x: x \in \alpha: x)$

$(\prod x: x \in [-2,5,-4,3,4,-8] \wedge x>0: x)$

$(\prod x: x \in [6,2,5,7,3,9,4] \wedge 2<x<7: x^2)$

$(\prod x: x \in \alpha \wedge 2<x<7 \wedge (x \bmod 2)=1: x)$

Donde la evaluación de *término es* un número.
El cuantificador puede tener más de una variable ligada (x, y).

Funciones para especificar

- En ocasiones, es necesario especificar varias veces lo mismo pero en diferentes problemas.
- A efectos de hacer más potente y sencilla la especificación, definiremos nuevas funciones de especificación para reducir la complejidad de la notación.
- Estas funciones tienen una especificación formal en términos de los operadores analizados recientemente.
- Por tal motivo, para evitar escribir la especificación completa, usaremos la función en lugar de su especificación.

Nota: estas funciones de especificación se utilizan para especificar con mayor claridad. Si es necesario pueden definirse nuevas, para facilitar las especificaciones. Son descriptivas, se utilizan en las pre y pos condiciones o definiciones pero no están implementadas en los algoritmos.

Funciones para especificar

vacía : $\Sigma^* \rightarrow$ Lógico
Dada una secuencia cualquiera, devuelve Verdadero si y solo si dicha secuencia no contiene elementos.
Ejemplo: vacía ([]) es verdadero, vacía ([9,7,5]) es falso

prim : $\Sigma^+ \rightarrow \Sigma$
Dada una secuencia con al menos un elemento, devuelve el primer elemento de dicha secuencia.
Ejemplo: prim ([9,7,5]) = 9

fin : $\Sigma^+ \rightarrow \Sigma^*$
Dada una secuencia con al menos un elemento, devuelve la misma secuencia pero sin su primer elemento.
Ejemplo: fin ("alto") = "lto"

Funciones para especificar

ult : $\Sigma^+ \rightarrow \Sigma$
Dada una secuencia con al menos un elemento, devuelve el último elemento de dicha secuencia.
Ejemplo: ult ("alto") = "o"

com : $\Sigma^+ \rightarrow \Sigma^*$
Dada una secuencia con al menos un elemento, devuelve la misma secuencia pero sin su último elemento.
Ejemplo: com ([9,7,5]) = [9,7]

longitud : $\Sigma^* \rightarrow Z$
Dada una secuencia cualquiera, devuelve la longitud de la misma, es decir, la cantidad de elementos que posee.
Ejemplo: longitud([9,7,5])= 3

Funciones para especificar

suma: $Z^* \rightarrow Z$

Dada una secuencia cualquiera de números enteros, devuelve la suma de todos sus elementos, es decir, la sumatoria de los mismos.
Ejemplo: suma([9,7,5])= 21

producto: $Z^* \rightarrow Z$

Dada una secuencia cualquiera de números enteros, devuelve el producto de todos sus elementos, es decir, la productoria de los mismos.
Ejemplo: producto([9,7,5])= 315

div: $(Z, Z) \rightarrow Z$

Dados dos elementos enteros cualquiera, devuelve el cociente de la división entera.
Ejemplo: div(19,5)= 3 o (19 div 5)=3

Funciones para especificar

nroApariciones :$(\Sigma , \Sigma^*) \rightarrow Z$

Dado un elemento y una secuencia cualquiera del mismo tipo que el elemento, devuelve la cantidad de veces que el elementos está presente dentro de la secuencia.
Ejemplo: nroApariciones(´a´, "cama")= 2

nroSubSec:$(\Sigma^*, \Sigma^*) \rightarrow Z$

Dadas dos secuencias cualquiera, del mismo tipo, devuelve la cantidad de veces que la primer secuencia se encuentra contenida dentro de la segunda secuencia.
Ejemplo: nroSubSec("mi", "mimica")= 2

mod: $(Z, Z) \rightarrow Z$

Dados dos elementos enteros cualquiera, devuelve el resto de la división entera.
Ejemplo: mod(19,5)= 4 o (19 mod 5)=4

Funciones para especificar

reversa: $\Sigma^* \rightarrow \Sigma^*$
Dada una secuencia cualquiera, devuelve la misma secuencia pero con sus elementos en orden inverso.
Ejemplo: reversa([9,7,5]) = [5,7,9]
reversa("pam") = "map"

esPermutación: $(\Sigma^*, \Sigma^*) \rightarrow$ Lógico
Dadas dos secuencia cualquiera, devuelve Verdadero si y solo si una secuencia es permutación de la otra, es decir, posee los mismos elementos y en idéntica cantidad.
Ejemplo: esPermutación([9,7,5], [7,5,9]) = Verdadero
esPermutación("abc", "bca") = Verdadero
esPermutación("pi", “pipi ") = Falso

Funciones para especificar

permutatoria: $\Sigma^* \rightarrow \{\Sigma^*\}$
Dada una secuencia cualquiera, devuelve el conjunto de todas las permutaciones posibles de dicha secuencia.
Ejemplos:
permutatoria([9,7,5]) = {[9,5,7], [7,5,9], [7,9,5], [5,7,9], [5,9,7], [9,7,5]}
permutatoria("abc") = {"acb", "bac", "bca“, "cab", "cba“, "abc"}
permutatoria("pi") = {"ip","pi"}

ordenar: $\Sigma^* \rightarrow \Sigma^*$
Dada una secuencia cualquiera, que posee relación de orden, devuelve la misma secuencia pero ordenada en forma creciente.
Ejemplos: ordenar([9,5,7]) = [5,7,9]
ordenar([8,3]) = [3,8]
ordenar([9,5,7]) = [5,7,9]

Funciones para especificar

ordenado: $\Sigma^* \rightarrow$ Lógico
Dada una secuencia cualquiera, que posee relación de orden, devuelve Verdadero si y solo si la secuencia está ordenada en forma creciente.

Ejemplos: ordenado([9,5,7]) = Falso
ordenado([3,8,11]) = Verdadero
ordenado("beso") = Verdadero
ordenado("amores") = Falso
ordenado([])=Verdadero

Ejemplos de especificaciones

- Dada una secuencia S cuyo primer elemento es una letra 'A', indicar en la variable lógica hayUnaLetraE si existe o no una letra 'E' en la secuencia.

  {Pre-condición: S='A'°α} o {prim(S)='A'}
  {Pos-condición: (($\exists$x: x$\in$S: x='E') = hayUnaLetraE) }
  o {Pos-condición: (($\forall$x: x$\in$S: x $\neq$'E') = $\neg$hayUnaLetraE) }

- Retornar la cantidad de elementos de una secuencia A que son mayores que 100 y menores o iguales que 1.000.

  {Pre-condición: True}
  {Definición: (#z: z$\in$A: 100<z$\leq$1.000) }

- Retornar la cantidad de vocales que posee una secuencia B no vacía.

  {Pre-condición: $\neg$vacia(B)} o {Pre-condición : long(B)>0}
  {Definición: nroApariciones(´a´, B) + nroApariciones(´e´, B) +
  nroApariciones(´i´, B) + nroApariciones(´o´, B) + nroApariciones(´u´, B)}

Ejemplos de especificaciones

- Calcular, en cant, la longitud de la secuencia α.

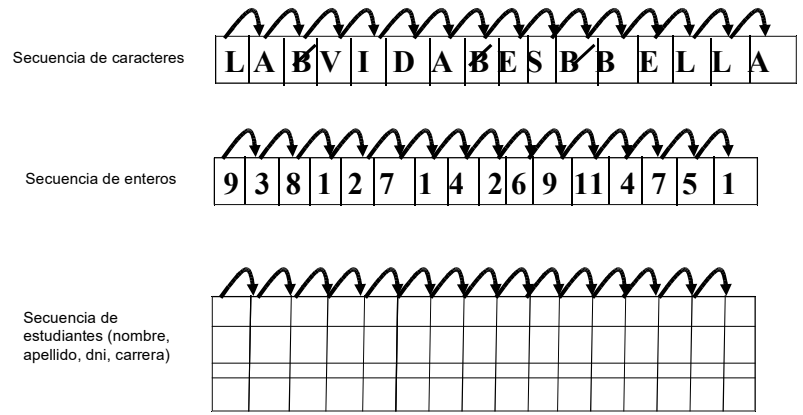
  {Pre-condición: True}
  {Pos-condición: cant=longitud(α)} o
  {Pos-condición: cant=(#*x: x*$\in\alpha$: True)}

- Dada una secuencia de enteros que contiene al menos un número 3, calcular, en sum, la sumatoria de todos los números pares de la secuencia, que están antes que el primer 3.

  {Pre-condición: S=(α•3)&$\beta \wedge \neg$($\exists$x: x$\in\alpha$: x=3)} o
  {Pre-condición: S=α&(3°β) $\wedge \neg$($\exists$x: x$\in\alpha$: x=3)}
  {Pos-condición: sum=($\sum$y: y$\in\alpha \wedge$ (y mod 2) = 0: y}

Arreglos como secuencias

Los arreglos son estructuras de datos que permiten representar secuencias.



Notación de Secuencias sobre arreglos
Ejemplos

En lugar de ligar un elemento al cuantificador, o al operador, se liga el índice y luego se hace referencia al arreglo en el índice deseado.

Existencial: $(\exists i: 1 \le i \le N: a[i]=0)$

$(\exists k: 1 \le k \le L: s[k]='G')$

$(\exists i,j: 1 \le i,j \le N \wedge i \ne j: a[i]=a[j])$

$(\exists i,j: 1 \le i,j \le N: a[i]=a[j] \wedge i \ne j)$

Universal: $(\forall k: 1 \le k \le M: a[k]>0)$

$(\forall i,j: 1 \le i,j \le N : s[i]=s[j] \wedge i \ne j)$

$(\forall i,j: 1 \le i \le Fila \wedge 1 \le j \le Col: 0<matrizAula79[i,j]<=100)$

Notación de Secuencias sobre arreglos
Ejemplos

Contatoria: $(\#j: 1 \le j \le N \text{ div } 2: -1<a[j]<10)$

$(\#i: 1 \le i < L: s[i]<>'D' \wedge s[i+1]<>'E')$

Máximo: $(\Uparrow i: 1 \le i \le N: (a[i] \bmod 2)=1)$

Mínimo: $(\Downarrow k: 1 \le k \le (N \text{ div } 2): a[k] \bmod 2)=1 \vee a[k] \bmod 2)=0)$

Sumatoria: $(\sum j: 1 \le j < M \wedge (b[j] \bmod 2)=0: b[j+1])$

Productoria: $(\prod i: 1 \le j \le N \wedge a[j]>10: a[j]^2)$

Notación de Secuencias sobre arreglos
Definición formal de algunas funciones

nroApariciones (x, a) = (#i : 0 < i <longitud(a) : x = a[i])

reversa(a) = b ↔ (∀i : 0 < i ≤ longitud(b): a[i] = b[longitud(b) - (i-1)])

ordenado (a) = (∀i : 0 < i ≤ longitud(a)-1 : a[i] <=a[i+1])

esPermutación (a,b) = longitud(a) = longitud(b) ∧ (∀i : 0 < i < longitud(b): nroApariciones(b[i], b) = nroApariciones(b[i], a))

ordenar (a) = b ↔ esPermutación(b,a) ∧ ordenado(b)

vacia(a) = (a = [])

Bibliografía

– Scholl, P. y Peyrin, J.-P. “Esquemas Algorítmicos Fundamentales: Secuencias e iteración”.

– Peyrin, J.-P. “Material del Seminario sobre la enseñanza de la programación: Todavía no aprendí a enseñar programación”. Escuela de Verano de Ciencias Informáticas, RIO 2006.

– Backhouse, R. “Program Construction. Calculating Implementations from Specifications”. Wiley, 2003

– Marti Oliet, N., Segura Díaz, C., Verdejo López, J.Especificación, derivación y análisis de algoritmos. Capítulo 1 (pags 1-19). 2006