6. Arreglos

En este capítulo definimos el tipo **arreglo**. La mayoría de los lenguajes de programación (JAVA, C, PASCAL, etc.) poseen el tipo arreglo. Como veremos, los arreglos permiten implementar, representar y manipular de una manera muy conveniente al tipo abstracto de dato **secuencia**, en particular permiten implementar de manera sencilla los cambios de valores que pueda tener una variable tipo secuencia.

6.1. Definición del tipo arreglo

Un conjunto finito de enteros consecutivos se denomina un ***segmento***. Por ejemplo {0, 1, 2} es un segmento y lo denotamos por **[0..3)**. En general, dados dos enteros p, q con $p \leq q$, el segmento de los enteros i que satisfacen $p \leq i < q$, lo denotamos por [p..q). Note que p = q implica que el segmento [p..q) es el conjunto vacío. También note que el número de elementos del segmento [p..q) es q-p. Note además que [2..1) no es un segmento.

Un ***arreglo*** es una función parcial de los enteros en cualquiera de los tipos básicos ya vistos (entero, real, carácter, booleano) o el mismo tipo arreglo, y cuyo dominio es un segmento [p..q), para p, q dados. Note que si p=0 entonces el arreglo no es más que una secuencia. Por lo que usaremos la misma notación de secuencias para denotar la aplicación de la función en un elemento del segmento, por ejemplo si b es un arreglo con dominio [0..3) entonces b es una secuencia de tres elementos y b[2] representa la imagen de 2 según b, es decir, el último elemento de la secuencia. Decimos que b[0] es el primer elemento del arreglo b, b[1] es el segundo elemento del arreglo b, etc. Un arreglo con dominio [p..p) diremos que es un arreglo sin elementos, es vacío.

Si b es un arreglo con dominio [0..2) y rango los números enteros, y cuyos elementos tienen los valores b[0]=-4, b[1]=-5, entonces el valor del arreglo lo denotamos por b = <-4, -5>.

Decimos que el tipo arreglo es un tipo ***estructurado***, lo cual significa que los valores del tipo vienen definidos por una estructura sobre valores de otros tipos (en este caso una función de enteros a otro conjunto).

Tradicionalmente los arreglos han sido vistos como un conjunto de variables indexadas e independientes que comparten un mismo nombre: el arreglo b con dominio [0..3) corresponde a las variables b[0], b[1] y b[2] (ver figura 1). Sin embargo, ver a los arreglos como funciones nos ayudará a manejar la formalidad cuando demostremos correctitud de programas.

En nuestro pseudolenguaje declararemos un arreglo b de la forma siguiente:

arreglo [p..q) de <tipo>

donde <tipo> puede ser cualquier tipo de nuestro pseudolenguaje, inclusive el tipo arreglo mismo. Tanto p como q representan expresiones de tipo entero cuyos valores satisfacen $p \leq q$.

Ejemplo de declaraciones de arreglos:

1) var b: arreglo [0..N) de entero; (con N ≥ 0)
2) var f: arreglo [2..3) de entero;
3) var a: arreglo [1..6) de booleano;

En la figura 7 vemos la representación gráfica de un arreglo, donde cada casilla corresponde a una localidad de memoria, es decir, b[i] es una variable.

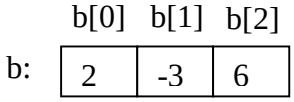

Figura 7

Si b es un arreglo con dominio [p..q) y p ≤ i ≤ j ≤ q, denotamos por b[i..j) los elementos del arreglo b correspondientes a los índices en el segmento [i..j) y decimos que es el segmento de b entre i y j. Note que, por ejemplo, b[p,p) (la secuencia vacía) es un segmento de b.

El tipo de los elementos de un arreglo puede ser un arreglo:

b: arreglo [p1..q1) de arreglo [p2..q2) de <tipo>;

Para abreviar la notación colocamos:

b: arreglo [p1..q1)×[p2..q2) de <tipo>;

Decimos que b es un arreglo de dimensión 2 de elementos de tipo <tipo>. Note también que b es un arreglo de dimensión 1 de elementos de tipo "arreglo [p2..q2) de <tipo>". Y vemos que b será una función del producto cartesiano [p1..q1)×[p2..q2) en el conjunto de valores de <tipo>. Vemos que b[i], p1 ≤ i < q1, es un arreglo cuyos elementos son de tipo arreglo [p2..q2) de <tipo>. Y b[i][j] será el elemento j del arreglo b[i]

Por ejemplo si b es un arreglo con dominio [0..2)×[1..4) en los enteros, y b=<<-4, -5, 2>, <5, 0, -3>>, entonces b[0][2] = -5.

Ejemplo de uso de arreglos:

El problema ya visto "sumar los elementos de una secuencia de enteros de largo N" puede ser implementado con el tipo arreglo, obteniéndose el programa siguiente:

```
[  const  N: entero;
   const f: arreglo [0..N) de entero;
   var  x,i: entero;
   {  N  ≥  0 }
```

```
  x := 0;
  i := 0;
  { Invariante I: ( x = ( ∑j: 0 ≤ j < i : f [j] ) ) ∧ (0 ≤ i ≤ N), función de cota creciente: i }
  do  i < N  →  x := x +f[i]; i := i+1  od
  {  x = (∑j: 0 ≤ j < N : f [j] ) }
]
```

Veamos un nuevo problema donde utilizamos arreglos.

Problema: Dado un arreglo A de enteros con dominio [0,N), N ≥ 0, deseamos calcular el segmento A[p..q) de A cuya suma de sus elementos sea máxima entre todos los segmentos de A.

Una especificación formal de este problema es:

```
[  const N: entero;
   const A: arreglo [0..N) de entero;  {N ≥ 0}
   var suma: entero;
   { verdad }
   maxsegsum
   { suma = (max p,q: 0 ≤ p ≤ q ≤ N : (∑ j: p ≤ j < q : A[j]))  }
]
```

Para simplificar la notación, definimos, para $0 \le p \le q \le N$:

$S(p,q)$: $(\sum j: p \le j < q : A[j])$

Una estrategia para resolver este problema es la siguiente:

Supongamos que $N > 0$ y hemos resuelto el mismo problema pero para el segmento de A[0..N-1), es decir, conocemos $(\max p,q: 0 \le p \le q \le N-1 : S(p,q))$, que denotaremos por $suma_{N-1}$.

Por lo tanto $(\max p,q: 0 \le p \le q \le N : S(p,q))$ es el máximo valor entre $suma_{N-1}$ y la suma de cada uno de los segmentos A[j..N), para $0 \le j \le N$, esta suma es S(j,N). Esto se debe a que:

```
        (max p,q: 0 ≤ p ≤ q ≤ N : S(p,q))
=  como N > 0 existe un término que separar
        max ((max p,q: 0 ≤ p ≤ q ≤ N-1 : S(p,q)),  (max p: 0 ≤ p ≤  N : S(p,N))
=  por definición
        max (suma_{N-1},  (max p: 0 ≤ p ≤  N : S(p,N)) )
```

Note que $suma_{N-1}$ lo podemos expresar de igual forma como: $\max (suma_{N-2}, (\max p: 0 \le p \le N-1 : S(p,N-1)))$. Por lo tanto el proceso iterativo que resuelve nuestro problema original es: al comienzo de la iteración i (comenzando desde 0) la variable **suma** contiene el valor

(max p,q: $0 \leq p \leq q \leq i$: S(p,q)), este sería nuestro invariante. Inicialmente podemos establecer el invariante con suma := 0; i:= 0 (la suma del segmento máximo de un arreglo vacío es cero). Cuando llegamos al comienzo de la iteración N hemos resuelto el problema. Por lo que otro invariante es $0 \leq i \leq N$. Y i (ó N-i) sirve como función de cota. Por lo tanto el esquema de nuestro programa sería:

suma := 0;
i := 0;
{ Invariante (suma = (max p,q: $0 \leq p \leq q \leq i$: S(p,q))) $\wedge$ $0 \leq i \leq N$ }
do i<N $\rightarrow$ *Prog* od

Del análisis hecho anteriormente *Prog* debe consistir en asignar a **suma** el máximo entre el valor de la variable **suma** al comienzo de la iteración y (max p: $0 \leq p \leq i+1$: S(p,i+1))

Ejercicio: Hacer un programa que calcule el máximo entre el valor de una variable **suma** (inicializada en cero) y (max p: $0 \leq p \leq N$: S(p,N)), para $N \geq 0$.

Notemos de nuevo que:

(max p: $0 \leq p \leq i+1$: S(p,i+1))
= podemos separar el último término pues $i \geq 0$
max ((max p: $0 \leq p \leq i$: S(p,i+1)), S(i+1,i+1))
= por definición de S(p,q)
max ((max p: $0 \leq p \leq i$: S(p,i+1)), 0)
= por definición de S(p,q) y $i \geq 0$
max ((max p: $0 \leq p \leq i$: S(p,i)+A[i]), 0)
= por aritmética y $i \geq 0$ (+ se distribuye sobre max)
max ((max p: $0 \leq p \leq i$: S(p,i)) + A[i], 0)

Así:

(max p: $0 \leq p \leq i+1$: S(p,i+1)) = max ((max p: $0 \leq p \leq i$: S(p,i)) + A[i], 0)

Por lo tanto si al comienzo de la iteración i tenemos en una variable **r** la cantidad (max p: $0 \leq p \leq i$: S(p,i)) entonces al comienzo de la iteración i+1 podemos tener la misma cantidad para i+1, si en *Prog* hacemos la asignación **r := (r + A[i]) max 0**, donde **a max b** representa en máximo entre a y b. Luego, como ya vimos, este valor de r nos sirve para calcular el valor de **suma** al comienzo de la iteración i+1, haciendo la asignación **suma := suma max r**. Y podemos agregar r = (max p: $0 \leq p \leq i$: S(p,i)) a nuestro invariante (***aplicando la técnica de fortalecimiento del invariante***). Finalmente en *Prog* debemos incrementar i en 1 para reflejar la siguiente iteración.

El programa finalmente es:

[const N: entero;
const A: arreglo [0..N) de entero;

```
  var suma: entero;
  { N ≥ 0 }
  suma := 0;
  i := 0;
  { Invariante: (suma = (max p,q: 0 ≤ p ≤ q ≤ i : S(p,q))) ∧ 0 ≤ i ≤ N
                ∧ r = (max p: 0≤p≤i: S(p,i)) }
  do  i<N  →  r := (r + A[i]) max 0;
              suma := suma max r;
              i := i+1
  od
  { suma = (max p,q: 0 ≤ p ≤ q ≤ N : (∑ j: p ≤ j < q : A[j]))  }
]
```

Note lo simple de la solución si hacemos un buen análisis. La estrategia de solución que empleamos se denomina Divide-and-Conquer (divide y conquistarás), y consiste en expresar la solución del problema en términos de la solución de problemas "más pequeños" del mismo tipo que el original. En el ejemplo anterior teníamos:

$$(\max p,q: 0 \le p \le q \le N : S(p,q)) = \max ((\max p,q: 0 \le p \le q \le N-1 : S(p,q)), (\max p: 0 \le p \le N : S(p,N)))$$

Más adelante seguiremos ejercitando esta estrategia para conseguir soluciones a problemas.

Ver la derivación del mismo programa en Kaldewaij (pag.67-70).

Ejercicios:

1) Páginas 62 y 71 del Kaldewaij.
2) Hacer un programa que satisfaga la siguiente especificación:

```
[  const N: entero;
   const s: secuencia de enteros;
   var r: entero;
   { N ≥ 0 }
   S
   { r = (#i,j: 0 ≤ i < j < N: s[i] ≤0 ∧ s[j] ≥ 0) }
]
```

Ejemplo: Dada una matriz cuadrada de orden N de enteros hacer un programa que determine si la matriz es simétrica.

La especificación formal sería:

```
[  const N: entero;
   const M: arreglo [0..N)×[0..N) de entero;
   var simetrica: booleano;
```

```
  { N ≥ 0 }
  es simétrica
  { simetrica ≡ (∀i,j: 0 ≤ i, j < N: M[i][j] = M[j][i]  }
]
```

El programa sería:

```
[  const N: entero;
   const M: arreglo [0..N)×[0..N) de entero;
   var simetrica: booleano;
   var n, m: entero;
   { N ≥ 0 }
   simetrica := verdad;
   n := 0;
   do n ≠ N ∧ simetrica →
                  m := n;
                  do  (m ≠ N) ∧ simetrica →
                                    if M[n][m] ≠ M[m][n] → simetrica := falso
                                    [] M[n][m] = M[m][n] → skip
                                    if;
                                 m := m+1
                  od;
                  n := n+1
   od
   { simetrica ≡ (∀i,j: 0 ≤ i, j < N: M[i][j] = M[j][i]  }
]
```

Note que el programa anterior se deduce del invariante para el primer ciclo siguiente:

$$\text{simetrica} \equiv (\forall i: 0 \le i < n : (\forall j: 0 \le j < N: M[i][j] = M[j][i])) \ \wedge \ 0 \le n \le N$$

Ejercicio: probar la correctitud del programa anterior.

6.2. Manipulación de arreglos

En esta sección presentamos cómo razonar sobre arreglos en programas que puedan modificarlos.

Dado un arreglo b, es decir, una función con dominio un segmento [p..q), es posible en el pseudolenguaje modificar el valor de la variable b[i], para un i dado, mediante una asignación. Así, b[i]:=e es una instrucción válida del pseudolenguaje, donde i es una expresión cuyo valor debe estar en el segmento [p..q), y e es una expresión con igual tipo que los elementos del arreglo b. La interpretación operacional es “reemplazar el valor de la variable b[i] por el valor resultante de evaluar e”.

Veremos que esta asignación se diferencia de la asignación ordinaria, en el hecho de que

esta afecta a la función b y no sólo a la variable b[i], y si no estamos conscientes de esto podemos llegar a conclusiones incorrectas. Veamos un ejemplo:

Suponga que b[0] = 1 y b[1] = 1. Entonces, b[b[1]] = b[1] = 1 y la instrucción b[b[1]] := 0 es equivalente a b[1] := 0 y después de esta asignación tendremos b[b[1]] = b[0] = 1. Parece paradójico que habiendo asignado 0 a b[b[1]] al final contenga 1; sin embargo, veremos que esto se debe a que estamos afectando el valor de una función completa, y no sólo el valor de una variable simple. Concluimos entonces que se cumple:

$$\{ b[0] = 1 \wedge b[1] = 1 \}\ b\,[b[1]] := 0\ \{ b[b[1]] = 1 \}$$

Vemos la diferencia con la asignación a una variable x, donde se cumple:

$$\{ \text{verdad} \}\ x := 0\ \{ x = 0 \}$$

Para expresar el cambio de valor de un arreglo (por lo tanto, de una función) introducimos la siguiente notación:

Sea **b** un arreglo, i una expresión cuyo valor está en el dominio de **b**, y e una expresión del tipo de los elementos del arreglo. Entonces b(i:e) denotará el arreglo (la función) que es igual a b salvo que la imagen de i es e:

$$b(i:e)[j] = \begin{cases} e & \text{si} \quad i = j \\ b[j] & \text{si} \quad i \neq j \end{cases}$$

Por ejemplo, si b=<2, 4, 6> con dominio [0..3) entonces:

- b(0:8)[0] = 8 (es decir, la función b(0:8) aplicada a 0 da 8)
- b(0:8)[1] = b[1] = 4 (es decir, la función b(0:8) aplicada a 1 da 4)
- b(0:8)[2] = b[2] = 6 (es decir, la función b(0:8) aplicada a 2 da 6)
- b(1:8) = <2, 8, 6>
- (((b(0:8))(2:9)) = <8, 4, 9>
- (((b(0:8))(0:9)) = <9, 4, 6>, para simplificar paréntesis colocaremos b(0:8)(0:9).

Será necesario saber simplificar expresiones con la nueva notación. Por ejemplo, queremos simplificar b(i:5)[j] = 5, es decir, tratar de encontrar un predicado equivalente pero en términos de b solamente. Podemos seguir el siguiente razonamiento: un j dado cumple con $j=i \vee j\neq i$. Para j=i tenemos que b(i:5)[j] = 5 se reduce a b(i:5)[i] = 5 y esto a 5=5; para $j\neq i$, se reduce a b[j]=5. Así:

$$\begin{aligned} & b(i:5)[j] = 5 \\ \equiv\ & \text{por tercero excluido y neutro de } \wedge \\ & (i=j \vee i\neq j) \wedge b(i:5)[j] = 5 \\ \equiv\ & \text{distributividad de } \wedge \text{ sobre } \vee \\ & (i=j \wedge b(i:5)[j] = 5) \vee (i\neq j \wedge b(i:5)[j] = 5) \end{aligned}$$

≡ por definición de b(i:5)
(i=j ∧ 5=5) ∨ (i≠j ∧ b[j] = 5)
≡ 5=5 es verdad y simplificación del ∧
(i=j) ∨ (i≠j ∧ b[j] = 5)
≡ distributividad de ∨ respecto a ∧
(i=j ∨ i≠j) ∧ (i=j ∨ b[j] = 5)
≡ ley del tercero excluido
V ∧ (i=j ∨ b[j] = 5)
≡ simplificación del ∧
i=j ∨ b[j] = 5

Ejercicios: página 92 del Gries.

Sea b un arreglo de dimensión 2 con dominio [0..2)×[1..4) en los enteros, b=<<-4, -5, 2>, <5, 0, -3>>. Note que la notación b(x:A), donde x es una expresión definida en el segmento [0..2) representa a un arreglo de dimensión 2 con dominio [0..2)×[1..4) y cuyos elementos son los mismos de b salvo en x donde su valor es A, note que A debe ser un arreglo de enteros con dominio [1..4). Extendemos esta notación como sigue: b([x][y]:A) representa un arreglo de dimensión 2 con dominio [0..2)×[1..4) y cuyos elementos son todos iguales a los de b salvo el elemento [x][y] cuyo valor es A. Note que b([1][2]:10)[1] = b[1](2:10)=<5,10,-3>.

Ahora estamos en condiciones de explicar la asignación b[i] := e, en términos de la definición como función de los arreglos. La asignación b[i] := e no es más que una abreviación de la siguiente asignación a la variable b:

b := b(i:e)

Cuando describimos b[i] := e como una abreviación de b := b(i:e), la definición como función de b es usada para describir el *efecto* de la ejecución de la asignación, pero no para indicar cómo se implementa la asignación. En la ejecución de la asignación b[i]:=e vemos al arreglo como una colección de variables independientes: se evalúa i y e, se selecciona la variable b[i] cuyo índice es el valor de i, y se asigna el valor de e a b[i]. No se crea un nuevo arreglo completo b(i:e) que se asigna a b. Esto lo vemos mejor en la figura 8.

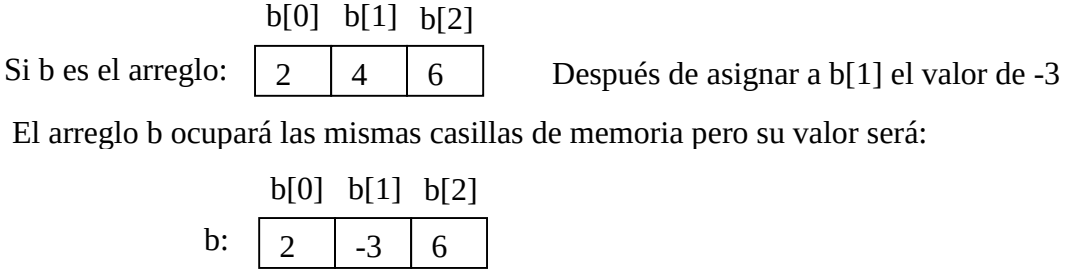

Figura 8

Por lo tanto la definición formal de la asignación a un elemento de un arreglo b con dominio [p..q), **la regla de la asignación a arreglos**, es:

{ P } b[i] := e {Q} se cumple
si y sólo si
[P $\Rightarrow$ e está bien definida $\wedge$ i está bien definida $\wedge$ p $\leq$ i < q $\wedge$ Q(b := b(i:e))]

donde Q(b := b(i:e)) denota a Q reemplazando las ocurrencias de b por b(i:e).

Ahora estamos en capacidad de probar que se cumple:

{ b[0] = 1 $\wedge$ b[1] = 1 } b [b[1]] := 0 { b[b[1]] = 1 }

Supongamos que se cumple b[0] = 1 $\wedge$ b[1] = 1, entonces:

b[b[1]] (b := b(b[1]:0))
= { sustitución }
b(b[1]:0)[b(b[1]:0)[1]]
= { b[1] = 1 }
b(1:0)[b(1:0)[1]]
= { definición de b(x:A) y 1 = 1 }
b(1:0)[0]
= { definición de b(X:A) y 1≠0 }
b[0]
= { b[0] = 1 }
1

<u>Ejemplo donde se usa asignación de valores a elementos de un arreglo</u>:

<u>Problema</u>: colocar en cero todos los elementos de un arreglo h de largo N≥0.

La especificación formal sería:

```
[  const N: entero;
   var h: arreglo [0..N) de entero;
   todos cero
   { (∀i: 0 ≤ i <N: h[i] = 0) }
]
```

Reemplazando la constante N por una variable entera n nos lleva a los invariantes P_0 y P_1 siguientes:

P_0: $(\forall i: 0 \leq i < n: h[i] = 0)$
P_1: $0 \leq n \leq N$

La guardia sería n ≠ N y los invariantes son establecidos inicialmente con n := 0. Investigamos ahora un incremento de 1 en n suponiendo que se cumple $P_0 \wedge P_1 \wedge n \neq N$:

$(\forall i: 0 \leq i < n+1: h[i] = 0)$
≡ como $0 \leq n$, podemos separar la sumatoria
$(\forall i: 0 \leq i < n: h[i] = 0) \wedge h[n] = 0$
≡ por definición de h(x:A)
$(\forall i: 0 \leq i < n: h[i] = h(n:0)[i]) \wedge h[n]=h(n:0)[n]$
≡ compactando la sumatoria
$(\forall i: 0 \leq i < n+1: h[i] = h(n:0)[i])$

La última línea nos dice que si reemplazamos a h por h(n:0), es decir, si asignamos 0 a h[n], se cumple $P_0(n := n+1)$. Note que $P_1(n := n+1)$ se cumple directamente.

Esto nos lleva a la siguiente solución de *todos cero*:

```
[  const N: entero;
   var h: arreglo [0..N) de entero;
   var n: entero;
   n := 0;
  ;do  n ≠ N → h[n] , n := 0, n+1 od
   { (∀i: 0 ≤ i <N: h[i] = 0) }
]
```

Ejemplo:
Calcular una tabla de frecuencias para N lanzamientos de un dado.

La especificación formal sería:

```
[  const N: entero; X: arreglo [0,N) de entero;
   var  h: arreglo [1..7) de entero;
  {  N ≥ 0  ∧ (∀i: 0 ≤ i < N: X[i] ≤ 6 )
   tabla de frecuencias
   { (∀i: 1 ≤ i < 7: h[i] = (#k: 0 ≤ k < N: X[k] = i))  }
]
```

Reemplazando la constante N por la variable n nos lleva a los invariantes:

P_0: $(\forall i: 1 \leq i < 7: h[i] = (\#k: 0 \leq k < n: X[k] = i))$
P_1: $0 \leq n \leq N$

La guardia sería n ≠ N y los invariantes son establecidos inicialmente con n := 0 y estableciendo luego la condición $(\forall i: 1 \leq i < 7: h[i] = 0)$, que ya hemos visto en el ejemplo anterior inicializar un arreglo en cero.

Investigamos ahora un incremento de 1 en n suponiendo que se cumple $P_0 \wedge P_1 \wedge n \neq N$. Para cualquier i, $1 \leq i \leq 6$, tenemos:

(#k: $0 \leq k < n+1$: X[k] = i)
= como $0 \leq n$, podemos separar la sumatoria
(#k: $0 \leq k < n$: X[k] = i) + #(X[n]=i)
= análisis de casos

$$\begin{cases} (\#k : 0 \leq k < n : X[k] = i) & si \;\; i \neq X[n] \\ (\#k : 0 \leq k < n : X[k] = i) + 1 & si \;\; i = X[n] \end{cases}$$

= por P_0

$$\begin{cases} h[i] & si \; i \neq X[n] \\ h[X[n]]+1 & si \; i = X[n] \end{cases}$$

= por definición de h(x:A)
h(X[n]:h[X[n]]+1)[i]

Por lo tanto, h debe ser reemplazado por h(X[n]:h[X[n]]+1). Y llegamos a la siguiente solución de *tabla de frecuencias*:

```
[  const N: entero; X: arreglo [0..N) de entero;
   var  h: arreglo [1..7) de entero;
   var n: entero;
  {  N ≥ 0  ∧ (∀i: 0 ≤ i < N: h[i] ≤ 6 )
   n := 0;
   [ var m: entero; m := 1; do  m ≠ 7 → h[m] , m := 0, m+1  od]
  ;do  n ≠ N →
                  h[X[n]] , n := h[X[n]]+1, n+1
   od
   { (∀i: 1 ≤ i < 7: h[i] = (#k: 0 ≤ k < N: X[k] = i))  }
]
```

En el programa anterior hemos introducido un nuevo constructor que llamaremos “bloque” y que corresponde al programa completo:

[var m: entero; m := 1; do m ≠ 7 → h[m] , m := 0, m+1 od]

En un bloque, el alcance de las variables que se declaran en él es el bloque mismo. Por ejemplo, la variable m no se puede utilizar fuera del bloque. Sin embargo toda variable declarada en un bloque B puede ser utilizada en los bloques que se definan dentro de B; por ejemplo, el arreglo h (declarado en el bloque del programa principal) puede ser utilizado dentro del bloque donde se declara la variable m.

Podemos tener anidamiento de bloques:

[var n:... [var m:... [var p:...] ...] ...]

El identificador de una variable declarada en un bloque dado B debe ser distinto de los identificadores de las variables declaradas en los bloques que contienen a B.

Ejercicio:

1) Demuestre que si h es un arreglo de enteros con dominio [0,N), P es el predicado "$0 \le n \le N \wedge (\forall i: 0 \le i < n: h[i] = H(i))$" y H(i) es una expresión en la cual no aparece h[], entonces se cumple:

$$\{ P \wedge n \neq N \wedge E = H(n) \}\ h[n] := E\ \{ P(n := n+1) \}$$

(Note que asignar ceros en un arreglo es un caso particular de esta proposición cuando H(i)=0).

6.3. Intercambio de dos elementos de un arreglo

Muchos problemas de programación pueden ser resueltos intercambiando los elementos de un arreglo. Denotaremos por **intercambio(E,F)** a la acción:: "intercambiar los valores de h[E] y h[F]". Esta interpretación informal no ayuda mucho. Una definición formal la daremos definiendo el arreglo h(x, y : A, B) de la siguiente forma:

$$h(x, y : A, B)[i] = \begin{cases} h[i] & \text{if } i \neq x \wedge i \neq y \\ A & \text{if } i = x \\ B & \text{if } i = y \end{cases}$$

Por lo tanto **la regla de intercambio de elementos de un arreglo** es la siguiente:

{ P } intercambio(E,F) { Q } se cumple si y sólo si

[P $\Rightarrow$ E está bien definida $\wedge$ F está bien definida $\wedge$ Q(h := h(E, F : h[F], h[E]))] es una tautología.

Si E y F no dependen de h, es fácil predecir el efecto de intercambio(E,F) sin hacer una demostración formal. De lo contrario, es difícil predecir el efecto sin hacer los cálculos correspondientes. Por ejemplo, sea h[0]=0 y h[1]=1. Por lo tanto intercambio(h[0],h[1]) es lo mismo que intercambio(0,1) lo cual resulta en h[0]=1 y h[1]=0. En particular, tendremos h[h[1]]=h[0]=1. Por lo que:

{ h[h[0]]=0 } intercambio(h[0],h[1]) { h[h[1]] = 0 } NO SE CUMPLE

Si E y F no dependen de h entonces la operación intercambio(E,F) puede escribirse como:

[var r: entero; r := h[E]; h[E] := h[F]; h[F]:= r]

Existe una regla, llamada **la regla de intercambio simple**, que es muy útil cuando queremos probar la correctitud de programas que involucran intercambio de elementos de un arreglo:

Si h no aparece en las expresiones E y F entonces se cumple:

{ $(\forall i: i \neq E \wedge i \neq F : h[i] = H(i)) \wedge h[E]=A \wedge h[F]=B$ }
intercambio(E,F)
{ $(\forall i: i \neq E \wedge i \neq F : h[i] = H(i)) \wedge h[E]=B \wedge h[F]=A$ }

Ejercicios:

1) Demuestre formalmente que se cumple:

$$\{ h[0]=0 \wedge h[1]=1 \} \text{ intercambio}(h[0],h[1]) \{ h[h[1]] = 1 \}$$

2) Demuestre formalmente la regla de intercambio simple.

Por lo tanto esta regla la podemos utilizar para demostrar la correctitud de programas que realicen intercambios.

Ejemplo: (reubicación respecto a un elemento pivote) Dado un arreglo h de N enteros queremos reubicar los elementos del arreglo utilizando sólo intercambio de elementos como la única operación permitida de arreglos (aparte de la operación de observación de sus elementos), de forma tal que los primeros k elementos del arreglo, hasta un cierto k a determinar, sean menores o iguales que un número entero dado X, y los últimos N-k elementos del arreglo desde k sean mayores que X.

La especificación formal de este problema es:

[const N, X: entero;
  var h: arreglo [0..N) de entero;
  var k: entero;
  { $N > 0 \wedge h = H$ }
  redistribuir
  { $(\forall z: z \in \mathbb{Z} : (\#j: 0 \leq j < N : h[j] = z) = (\#j: 0 \leq j < N : H[j] = z)) \wedge 0 \leq k \leq N$
    $\wedge (\forall i: 0 \leq i < k : h[i] \leq X) \wedge (\forall i: k \leq i < N : h[i] > X)$ }
]

El predicado $(\forall z: z \in \mathbb{Z} : (\#j: 0 \leq j < N : h[j] = z) = (\#j: 0 \leq j < N : H[j] = z))$ indica que el valor final de h es una permutación de su valor original (están los mismos elementos originales y en igual cantidad). Los otros dos predicados establecen la propiedad que cumplirá h una vez concluya el programa.

Una forma de atacar el problema es utilizando la técnica de reemplazo de constante por variable, por ejemplo N por n en el último predicado. Sin embargo, esto resulta en un programa complicado pues no aprovecha la simetría del problema que indica que los primeros elementos del arreglo son menores o iguales que X y los últimos mayores que X (la simetría de problema sugiere que a medida que vayamos considerando uno a uno los

elementos originales del arreglo, ir colocando los menores o iguales al comienzo y los mayores al final del mismo arreglo).

Note que si para p,q, $0 \leq p \leq q \leq N$, tenemos que los primeros p elementos de h, el segmento h[0,p), son menores o iguales que X y los últimos N-q elementos de h, el segmento h[q,N), son mayores que X, entonces, si $p \neq q$, podemos decidir fácilmente donde reubicar al elemento h[p] o al elemento h[q-1] (los extremos del segmento h[p,q)) de forma que aumentemos en 1 el número de elementos ya ubicados sea al comienzo o al final del arreglo dependiendo respectivamente de si es menor o igual, o mayor que X . Si h[p] es menor o igual que X, no hace falta reubicar a h[p] y tendremos que los primeros p+1 elementos de h son menores o iguales a X y los últimos N-q elementos de h son mayores que X. Si h[p] es mayor que X entonces podemos intercambiar los elementos h[q-1] y h[p], y así obtendremos que los primeros p elementos de h son menores o iguales a X y los últimos N-q+1 elementos de h son mayores que X. De esta forma, continuando este proceso llegamos a reubicar todos los elementos de h y el valor final de p será el k buscado en nuestro problema original.

Esto sugiere que el invariante sea:

$(\forall z: z \in \mathbb{Z} : (\#j: 0 \leq j < N : h[j] = z) = (\#j: 0 \leq j < N : H[j] = z)) \wedge (\forall i: 0 \leq i < p : h[i] \leq X) \wedge (\forall i: q \leq i < N : h[i] > X) \wedge 0 \leq p \leq q \leq N$

Note que reemplazamos en la postcondición una misma variable (k) por dos distintas (p y q). Sin embargo hemos podido dejar igual la primera ocurrencia de k y reemplazar la segunda ocurrencia de k por la variable nueva q, quedando el invariante:

$$P_0 \wedge P_1 \wedge P_2 \wedge P_3$$

Donde:
P_0: $(\forall z: z \in \mathbb{Z} : (\#j: 0 \leq j < N : h[j] = z) = (\#j: 0 \leq j < N : H[j] = z))$
P_1: $(\forall i: 0 \leq i < k : h[i] \leq X)$
P_2: $(\forall i: q \leq i < N : h[i] > X)$
P_3: $0 \leq k \leq q \leq N$

La postcondición se obtiene con k = q, por lo que la guardia será $k \neq q$. El invariante puede ser establecido inicialmente con k, q := 0, N. Como k debe aumentar o q disminuir en cada iteración, una función de cota decreciente sería q-k la cual es no negativa pues q es siempre mayor o igual a k.

Nuestro programa tiene el esquema siguiente:

k, q := 0, N;
do $k \neq q \rightarrow$ S od

Determinemos S. Si $k \neq q$ entonces k < q, y los elementos del arreglo en el segmento [k,q) son todos candidatos a ser reubicados, los más obvios a considerar son h[k] o h[q-1].

Consideremos h[k], lo cual nos lleva al esquema:

k, q := 0, N;
do $k \neq q$
  $\rightarrow$ if $h[k] \leq X \rightarrow S_1$
    [] $h[k] > X \rightarrow S_2$
    fi

od

Como ya observamos, si $h[k] \leq X$ entonces basta con incrementar k en 1 para que se siga cumpliendo el invariante, y si $h[k] > X$ entonces basta con intercambiar h[p] con h[q-1] (al ser $k < q$, tenemos $k \leq q-1$) y disminuir q en 1 para que se siga cumpliendo el invariante. El programa final con las anotaciones es el siguiente:

[const N, X: entero;
  var h: arreglo [0..N) de entero;
  var k,q: entero;
  { $N > 0 \wedge h = H$ }
  k, q := 0, N;
  {Invariante: $P_0 \wedge P_1 \wedge P_2 \wedge P_3$, demo 0; función de cota decreciente: q-k}
  do $k \neq q$
    $\rightarrow$ { $P_0 \wedge P_1 \wedge P_2 \wedge P_3 \wedge k \neq q$ }
      if $h[k] \leq X \rightarrow k := k+1$
      [] $h[k] > X \rightarrow$ intercambio(k,q-1); q := q-1
      fi
      { $P_0 \wedge P_1 \wedge P_2 \wedge P_3$, demo 1}
  od
  { $(\forall z: z \in Z : (\#j: 0 \leq j < N : h[j] = z) = (\#j: 0 \leq j < N : H[j] = z)) \wedge 0 \leq k \leq N$
    $\wedge (\forall i: 0 \leq i < k : h[i] \leq X) \wedge (\forall i: k \leq i < N : h[i] > X)$, demo 2; Terminación: demo 3}
]

Ejercicios:

1) Demuestre la correctitud del programa anterior (ayuda: utilice la regla de intercambio simple).
2) Utilizando intercambios como única operación sobre arreglos, hacer un programa que dado un arreglo de enteros de largo N, reubique los elementos del arreglo de forma tal que los primeros elementos del arreglo sean menores que un pivote dado X, los elementos del medio del arreglo sean iguales a X y los últimos elementos del arreglo sean mayores que X..
3) Utilizando intercambios como única operación sobre arreglos, hacer un programa (rotación de los elementos de un arreglo) que cumpla:
   [const K, N: entero;
     var h: arreglo [0,N) de entero;
     { $N \geq 0 \wedge h = H$ }
     rotación

```
    { (∀i: 0 ≤ i < N : h[(i+K) mod N]] = H[i] )
  [
```

Cuando definimos una función o un procedimiento, la manera como declaramos un parámetro formal tipo arreglo en el pseudolenguaje es como sigue:

proc <nombre procedimiento> (... ; <tipo parámetro> x : arreglo de <tipo>; ....)

donde,

- <nombre procedimiento> es el nombre del procedimiento.
- <tipo parámetro> es entrada, salida o entrada-salida. La interpretación operacional es exactamente la misma que para variables simples, es decir, si el parámetro formal es de entrada entonces en una llamada al procedimiento se copia el valor del parámetro real en la variable correspondiente al parámetros formal, por lo tanto en tiempo de ejecución el parámetro formal es un arreglo que ocupa un lugar de memoria completamente distinto al del parámetro real.

En una llamada a un procedimiento que contenga arreglos como parámetros, los parámetros reales determinan los límites inferiores y superiores del los rangos de los parámetros formales. Por ejemplo, si en la llamada el parámetro real es un arreglo de enteros con dominio [4,10) entonces en la ejecución de la llamada el parámetro formal será un arreglo de enteros con dominio [4,10). Por lo tanto es conveniente pasar también como parámetros a los límites inferior y superior de los dominios de cada arreglo que se pase como parámetro.

Por ejemplo, el programa de reubicación según un pivote visto antes lo podemos convertir en un procedimiento que dado un arreglo y un pivote, redistribuya los elementos del arreglo según el pivote:

```
{ Pre: h = H ∧ H es un arreglo con dominio [N1..N2)  ∧  0 ≤ N1 ≤ N2     }
{ Post: (∀z: z ∈ Z : (#j: N1 ≤ j < N2 : h[j] = z ) = (#j: N1 ≤ j < N2 : H[j] = z ))
          ∧  N1 ≤ k ≤ N2  ∧ (∀i: N1 ≤ i < k : h[i] ≤ X ) ∧ (∀i: k ≤ i < N2 : h[i] > X )  }
proc Pivote( entrada X : entero; entrada N1, N2: entero; entrada-salida h: arreglo de entero;
                salida k: entero)
[
  var q: entero;
   k, q := N1, N2;
  do k ≠ q
        → if  h[k] ≤ X → k := k+1
           []  h[k] > X → intercambio(p,q-1); q := q-1
           fi
  od
]
```

Ejercicios:

1) Hacer un procedimiento que implemente la operación de intercambio de dos elementos de un arreglo dado.
2) Hacer una función que reciba como parámetro una matriz cuadrada y devuelva verdad si y sólo si la matriz es simétrica.
3) Página 168 Kaldewaij