METODOLOGIA DE LA PROGRAMACIÓN
Junio 2008

1.- (2 puntos) Documenta (con las aserciones que se marcan) el siguiente programa iterativo que calcula el máximo de la suma de todos los segmentos iniciales del vector A(1..n).

(1) { n≥1}

i := 1; m := A(1); seg := A(1);

(2) {1 ≤ i ≤n ∧ m= max { $\sum_{j=1}^{k}$ A(j) / 1≤ k ≤ i } ∧ seg= $\sum_{j=1}^{i}$ A(j) }

while i < n **loop**

E= n-i

i := i + 1;

(3){ 1 ≤ i ≤n ∧ m= max { $\sum_{j=1}^{k}$ A(j) / 1≤ k ≤ i-1 } ∧ seg= $\sum_{j=1}^{i-1}$ A(j) }

seg := seg + A(i);

(4){ 1 ≤ i ≤n ∧ m= max { $\sum_{j=1}^{k}$ A(j) / 1≤ k ≤ i-1 } ∧ seg= $\sum_{j=1}^{i}$ A(j) }

if seg > m **then**

(5){ 1 ≤ i ≤n ∧seg = max { $\sum_{j=1}^{k}$ A(j) / 1≤ k ≤ i } ∧ seg= $\sum_{j=1}^{i}$ A(j) }

m := seg;

end if;

end loop;

Post= { m= max { $\sum_{j=1}^{k}$ A(j) / 1≤ k ≤ n } }

Puntuación: 0'3 (1 ≤ i ≤n), 0'7 (m=…), 0'7 (seg=…), 0'3 (E=n-i)

2.- (2'5 puntos) Deriva formalmente una iteración que satisfaga la siguiente especificación:

Precondición: {1 ≤ i ≤ n}

Postcondición: {c = *N*j (1 ≤ i < j ≤ n ∧ A(i) * A(j) ≥ 0)}

ESPECIFICACIÓN:

INV ≡{ 1 ≤ i ≤ k ≤ n ∧ c = *N*j (1 ≤ i < j ≤ k ∧ A(i) * A(j) ≥ 0)}

E≡ n-k

INICIALIZACIÓN

{1 ≤ i ≤ n} **k:= i; c:=0;** { 1 ≤ i ≤ n ∧ k=i ∧ c= 0 } →

{ 1 ≤ i ≤ k ≤ n ∧ c = *N*j (1 ≤ i < j ≤ k ∧ A(i) * A(j) ≥ 0)}

FINALIZACIÓN

B≡ (k<n) $\quad\quad$ $\neg$**B≡ (k=n)**

INV$\wedge \neg$**B** ={ $1 \le i \le k \le n \wedge c = N j\ (1 \le i < j \le k \wedge A(i) * A(j) \ge 0) \wedge$ **k=n** } $\rightarrow$

{ c = *N*j $(1 \le i < j \le$ **n** $\wedge A(i) * A(j) \ge 0)$} = Postcondición

CUERPO DE LA ITERACIÓN:

{INV$\wedge$ **B} I {INV}**

$(1 \le i \le$ **k < n** $\wedge$ c = *N*j $(1 \le i < j \le k \wedge A(i) * A(j) \ge 0)$) $\rightarrow$

$(1 \le i \le$ **k+1 ≤ n** $\wedge$ c = *N*j $(1 \le i < j \le$ **k+1-1** $\wedge A(i) * A(j) \ge 0)$)

k:= k+1;

{ $1 \le i \le$ **k ≤ n** $\wedge$ c = *N*j $(1 \le i < j \le$ **k-1** $\wedge A(i) * A(j) \ge 0)$ }

if A(i)*A(k)≥ 0 then

{$1 \le i \le k \le n \wedge$ c = *N*j $(1 \le i < j \le k-1 \wedge A(i) * A(j) \ge 0) \wedge A(i) * A(k) \ge 0\} \rightarrow$

{$1 \le i \le k \le n \wedge$ c +1 = *N*j $(1 \le i < j \le k \wedge A(i) * A(j) \ge 0)$ }

c:= c+1;

{$1 \le i \le k \le n \wedge$ c = *N*j $(1 \le i < j \le k \wedge A(i) * A(j) \ge 0)$ }**≡ INV**

end if;

TERMINACIÓN
(a) INV $\wedge$ **B** $\rightarrow$ **E**
{ $1 \le i \le$ **k < n** $\wedge$ c = *N*j $(1 \le i < j \le k \wedge A(i) * A(j) \ge 0)\} \rightarrow$ **n-k >0** $\in N \rightarrow$ **n-k ≡ E**
(b) {INV $\wedge$ **B** $\wedge$ **E=v} I {E<v}**
- $(1 \le i \le$ **k < n** $\wedge$ c = *N*j $(1 \le i < j \le k \wedge A(i) * A(j) \ge 0) \wedge$ n-k=v) $\rightarrow$ (n-(k+1) <v)
- { n-(k+1) <v} **k:= k+1 {n-k<v}**

ITERACIÓN DOCUMENTADA:

Precondición= $\{1 \le i \le n\}$

k:= i; c:=0;

{ $1 \le i \le k \le n \wedge$ c = *N*j $(1 \le i < j \le k \wedge A(i) * A(j) \ge 0)\}$ =**INV**

while k<n loop $\quad\quad$ **E= n-k**

k= k+1;

{ $1 \le i \le$ **k ≤ n** $\wedge$ c = *N*j $(1 \le i < j \le$ **k-1** $\wedge A(i) * A(j) \ge 0)$ }

if A(i)*A(k)≥ 0 then

{$1 \le i \le k \le n \wedge$ c +1 = *N*j $(1 \le i < j \le k \wedge A(i) * A(j) \ge 0)$ }

c:= c+1;

{$1 \le i \le k \le n \wedge$ c = *N*j $(1 \le i < j \le k \wedge A(i) * A(j) \ge 0)$ }**= INV**

end if;

end loop;

Puntuación: 0'5(ESPECIF.), 0'5(INI), 0'25(FIN), 0'75(CUERPO), 0'5(TERMI.)

3.- (1'5 puntos) Verifica la corrección de la función recursiva:

function num_combinatorio (n, k: natural) **return** c: natural **is**
Pre: {1 ≤ k ≤ n}
  if k = 1 **then** c := n;
  else c := num combinatorio(n − 1, k − 1);
    c := c * n ;
    c := c/k;
  end if
Post: {c = $\frac{n!}{k!(n-k)!}$ }

CASO(S) SIMPLE(S)

$(1 \le k \le n \wedge k = 1) \rightarrow (n = \frac{n!}{1!\,(n-1)!} \wedge k = 1) \rightarrow (n = \frac{n!}{k!\,(n-k)!})$

{ n = $\frac{n!}{k!(n-k)!}$ } c:= n; { c = $\frac{n!}{k!(n-k)!}$ } **por AA**

CASO(S) INDUCTIVO(S)

h.i. = {1 ≤ k-1 ≤ n-1} c:= num_combinatorio (n-1, k-1); {c = $\frac{(n-1)!}{(k-1)!(n-k)!}$ }

$(1 \le k \le n \wedge k > 1) \rightarrow (1 \le k-1 \le n-1)$

{1 ≤ k-1 ≤ n-1} c:= num_combinatorio (n-1, k-1); {c = $\frac{(n-1)!}{(k-1)!(n-k)!}$ } **por h.i.**

$(c = \frac{(n-1)!}{(k-1)!(n-k)!}) \rightarrow (c*n = \frac{n!}{(k-1)!(n-k)!})$

{ c*n = $\frac{n!}{(k-1)!(n-k)!}$ } c:= c*n; { c = $\frac{n!}{(k-1)!(n-k)!}$ } **por AA**

$(c = \frac{n!}{(k-1)!(n-k)!}) \rightarrow (c/k = \frac{n!}{k!(n-k)!})$

{ c/k = $\frac{n!}{k!(n-k)!}$ } c:= c/k; { c= $\frac{n!}{k!(n-k)!}$ } **por AA**

VALIDACIÓN DE LA INDUCCIÓN:

Expresión cota = k

Caso simple
- k=1 ∈ **N**

Caso inductivo
- k>1 ∈ **N**
- num_combinatorio (n, k) llama a num_combinatorio (n-1, k-1)
  - Se cumple que k>1 → k-1 >0 → k-1 ∈ **N**
  - **k-1 < k**

Puntuación: 0'25 (SIMPLE), 0'75(INDUC,tiene 3 pasos), 0'5 (VALIDA)

4.- (2 puntos) Especifica ecuacionalmente una función que dada una secuencia de enteros *s*, un natural *n*, y un entero *x*, decida si *s* contiene al menos *n* elementos mayores que *x*.

sobre: secuencia(integer), nat
usa: bool

operación

contiene: secuencia(integer) x nat x integer → bool

ecuaciones (usando en los dos parámetros las constructoras)

contiene(< >, 0, x) = true
contiene(< >, succ(n), x) = false

contiene(a•S, 0, x) = true

$$\text{contiene}(a\bullet S, \text{succ}(n), x) = \begin{cases} \text{contiene}(S, n, x) & \text{si } a>x \\ \text{contiene}(S, \text{succ}(n), x) & \text{e.o.c} \end{cases}$$

ecuaciones (NO usando en los dos parámetros las constructoras)

$$\text{contiene}(<>, n, x) = \begin{cases} \text{true} & \text{si } n=0 \\ \text{false} & \text{e.o.c. (si } n\neq 0) \end{cases}$$

$$\text{contiene}(a\bullet S, n, x) = \begin{cases} \text{true} & \text{si } n= 0 \\ \text{contiene}(S, n-1, x) & \text{si } a>x \wedge n>0 \\ \text{contiene}(S, n, x) & \text{e.o.c (si } a\leq x \wedge n>0) \end{cases}$$

Puntuación: 0'5(INV), 0'25(INI), 0'5(FIN), 0'5(CUERPO), 0'25(TERM)

5.- (2 puntos) Utilizando el método de **Burstall**, diseña un algoritmo iterativo que compute la función $f(n) = 2^{n-1} - 1$, según la siguiente definición recursiva:

$$f(n) = \begin{cases} (n - 1) & \text{si } 1 \leq n \leq 2 \\ 3 * f(n - 1) - 2 * f(n - 2) & \text{si } n > 2 \end{cases}$$

PRE: {n ≥ 1}

POST: { $f(n) = 2^{n-1} - 1$ }

INVARIANTE (relación de recurrencia):

desplegado desplegado plegado
f(5) = 3* f(4) – 2 * f(3) = 3*(3* f(3) – 2* f(2)) – 2* f(3) = 7 * f(3) – 6 * f(2) =
desplegado plegado
= 7 * (3* f(2) – 2 * f(1)) – 6 * f(2) = 15 * f(2) – 14 * f(1) = 15

INV ≡ {f(n) = k * f(m+1) – (k-1) * f(m) }

Puedo haber otras opciones: {f(n) = (k+1) * f(m+1) – k * f(m) } ó {f(n) = k * f(m+1) – l* f(m) } ó {f(n) = k * f(m) – (k-1) * f(m-1) } ó variaciones

Cada una de ellas afectará especialmente en el proceso de inicialización.

INICIALIZACIÓN:

k:= 0; m:= n;

f(n) = k * f(m+1) – (k-1) * f(m)= 0 * f(n+1) - (0-1) * f(n) = 0 + 1* f(n) = f(n)

¡OJO! Otro tipo de inicialización puede aparentar correcta pero hace que algunos valores para f no cumplan las precondiciones. Por ejemplo, k:=1; m:= n-1;

f(n) = k * f(m+1) – (k-1) * f(m)= 1 * f(n-1+1) - (1-1) * f(n-1) = 1* f(n) + 0* **f(n-1)**

Si **n ≥ 1** entonces **n-1 ≥ 0**, es decir no está definido. No cumple la precondición

FINALIZACIÓN:

B ≡ (m>1) **¬B≡ (m=1)**

f(n) = k * f(m+1) – (k-1) * f(m)= k * f(2) – (k-1) * f(1) = k * 1 – (k-1) * 0= **k**

r:= k;

CUERPO DE LA ITERACIÓN Y TERMINACIÓN:

{INV ∧ B } I {INV}

f(n) = k * f(m+1) – (k-1) * f(m) = k* (3* f(m) - 2 * f(m-1)) – (k-1) * f(m) =

= 3k-(k-1) * f(m) – 2k * f (m-1) = $\underset{k'}{2k+1}$ * f(m) – 2k * f($\underset{m'}{m-1}$)

k:= 2*k + 1;

m:= m-1;

TERMINACIÓN
(a) (INV ∧ B) → E ∈ N
m ≡ E
(b) {INV ∧ B ∧ m=v} I {m< v}
- **(INV ∧ B ∧ m=v) → (m=v) → (m-1< v)**
- **{ m-1< v} m:= m-1; {m<v}**

ITERACIÓN DOCUMENTADA:

function f_it(n: integer) return r:integer is
PRE≡ {n ≥ 1}
k:= 0; m:= n;
{f(n) = k * f(m+1) – (k-1) * f(m) } ≡ INV **E ≡ m**
while m>1 loop
k:= 2*k + 1;
m:= m-1;
end loop;
r := k;
POST≡ { f(n) = $2^{n-1} - 1$ }