Examen Final de PRED

La duración del examen es de 3 horas. El primer ejercicio permite recuperar la parte correspondiente del primer parcial. En concreto, la nota final de la parte de especificación, Espec, se calcula de la siguiente forma:

Espec = max(EspecP,EspecF)

donde EspecP es la nota obtenida en el examen parcial y EspecF es la nota obtenida en este examen. Aunque la nota obtenida en el examen parcial correspondiente a esta parte sea lo suficientemente alta como para no necesitar recuperación, se recomienda a todo el mundo que realice este ejercicio, ya que esto puede ayudar a la realización del segundo problema.

1. **Especificación:** Especificar una abstracción de datos que sirva para implementar el inventario de una tienda. Es decir esta estructura nos ha de servir para saber cuantas unidades de cada artículo hay en la tienda. Se supone que cada artículo se identifica con un código que es un número natural. Es decir, suponemos que los artículos son, simplemente, números. Además, se supone que tenemos (ya especificada e implementada) una operación *valor* que dado un artículo nos dice cuanto dinero cuesta. Esta abstracción ha de tener las siguientes operaciones:

   crear_inv: que nos devuelve el inventario vacío
   envío: dado el inventario I, un artículo N y una cantidad M nos devuelve un nuevo inventario en que el las existencias del artículo N se han incrementado en M unidades.
   existencias: dado el inventario I y un artículo N, nos dice cuantas existencias hay de dicho artículo.
   valor_inv: dado el inventario I nos dice cuanto cuesta el conjunto de artículos que tenemos en el mismo.

2. **Diseño de estructuras de datos:** Diseñar una estructura de datos y sus operaciones asociadas que implemente la abstracción de datos que se describe en el ejercicio de especificación. Evidentemente, se ha de justificar que esta implementación es correcta, describiendo su invariante y la relación de equivalencia asociada.

Soluciones

1)

```
especificación inv
tipos inv, nat
operaciones
      crear_inv: -> inv
      envío: inv x nat x nat -> inv
      existencias: inv x nat -> nat
      valor_inv: inv -> nat
axiomas
      envío(i, art,0) = i
      envío(envío(i,art,n),art,m)= envío(i,art,n+m)
      envío(envío(i,art,n),art',m)= envío(envío(i,art',m),art,n)
      existencias(crear_inv,art) = 0
      existencias(envío(i,art,n),art) = n+ existencias(i,art)
      art≠art' =>
            existencias(envío(i,art,n),art')= existencias(i,art)
      valor_inv(crear_inv)= 0
      valor_inv(envío(i,art,n))= n*valor(art)+ valor_inv(i)
fespecificación
```

2) La implementación adecuada de la estructura de datos del problema se basa en el uso de hash con encadenamiento. En concreto, esta implementación permite un acceso casi directo a los artículos tanto para procesar los envíos como para calcular las existencias. Por otra parte, la operación del cálculo del valor del inventario es relativamente eficiente ya que la tabla que da acceso a las listas de sinónimos no es, típicamente, excesivamente grande, por lo que su recorrido no sería costoso. Incluso, este coste se podría evitar utilizando un campo en la implementación del inventario que nos guarde su valor en todo momento. Evidentemente, este campo debería de ser actualizado tras cada envío e inicializado a 0 al crear el inventario.

El hash abierto es una implementación menos adecuada, ya que el cálculo del valor del inventario es muy poco eficiente: se tendría que recorrer una tabla típicamente muy grande. Este inconveniente se salva si, como en el caso anterior, se utilizar un campo para tener ese valor calculado en todo momento.

El uso de una lista ordenada es menos adecuada ya que para acceder a los artículos, tanto en las operaciones de envío como de existencias se ha de recorrer una lista que puede ser relativamente larga y por tanto estas operaciones serían inútilmente ineficientes. Si además la lista está representada de manera contigua en una tabla y, además no se usa búsqueda binaria en las búsquedas, entonces la representación sería aun peor ya que se pierde la mayor ventaja de esta representación manteniendo todas sus desventajas

Si la lista está desordenada la implementación es peor, ya que las ineficiencias aumentan en las operaciones de envío y existencias.

Por último, la implementación usando directamente una tabla en que los artículos corresponden a los índices de la tabla sólo sería correcta si se dan dos condiciones, que el número de claves de artículos que, potencialmente, puede haber en un inventario sea relativamente pequeño y que, además estas claves estén numeradas del 1 (o el 0) al N, siendo N relativamente pequeño. No hay nada en el enunciado que permita asumir estas condiciones. De hecho, en la vida real, los códigos de los artículos son códigos de barras que representan números muy altos (de 10 o más dígitos) lo que iría en contra de la 2a suposición.

Como consecuencia, la representación podría ser la siguiente:

```
Clase nodo
  art: nat;
  cant: nat;
  sig: ^nodo
fClase

Clase Inventario
acción crear_inv(sal i: Inventario);
acción envío(ent/sal i: Inventario; ent a,c: nat);
función existencias (ent i:Inventario; ent a:nat) retorna c: nat;
función valor_inv (ent i:Inventario) retorna v: nat;

Implementación
  t: tabla [1:N] de ^nodo
  val_inv: nat

acción crear_inv(sal i: Inventario);
Pre: cierto
Post: t contiene sólo accesos a listas vacías; val_inv = 0
var j: nat;
para j:= 1 a N hacer i.t[j]:= nil;
fpara
val_inv: = 0
f_acción

acción envío(ent/sal i: Inventario; ent a,c: nat);
Pre: cierto
Post: si el artículo a estaba en la tabla, se ha sumado c a su
cantidad; si no estaba y c no es 0 se ha incluido <a,c> en la lista de
sus sinónimos. Además, val_inv se ha incrementado en c*valor(a).
var p:nat; encontrado: bool; aux: ^nodo;

si c>0 entonces
  p:= hash(a); encontrado:= falso; aux:= i.t[p];

  Inv: {(encontrado => aux^.art = a y la cantidad asociada al artículo
       a ya incluye los c nuevos artículos); (si aux no apunta a nil
       entonces apunta a un elemento de la lista de sinónimos de a & a
       no está en los elementos de la lista anteriores a aux); (si aux
       apunta a nil entonces a no está en su lista de sinónimos)}.
  mientras aux ≠ nil & no(encontrado) hacer
       si aux^.art = a entonces
            encontrado: = cierto;
            aux^.cant:= aux^.cant + c;
       sino aux:= aux^.sig;
       fsi;
  fmientras;
  si no(encontrado) entonces
       new(aux);
       aux^.art:= a;
       aux^.cant:= c;
       aux^.sig:= i.t[p];
       i.t[p]:=aux;
  fsi;
  i.val_inv:= i.val_inv + c*valor(a);
fsi;
facción
```

```
función existencias (ent i:Inventario; ent a:nat) retorna c: nat;
Pre: cierto
Post: si el artículo a estaba en la tabla entonces c es la cantidad
que estaba guardada en el nodo de a; si el artículo a no estaba en la
tabla entonces c = 0.
var p:nat; encontrado: bool; aux: ^nodo;

  p:= hash(a); encontrado:= falso; aux:= i.t[p];

  Inv: {(encontrado => aux^.art = a); (si aux no apunta a nil entonces
       apunta a un elemento de la lista de sinónimos de a & a no está
       en los elementos de la lista anteriores a aux); (si aux apunta
       a nil entonces a no está en su lista de sinónimos)}.
  mientras aux ≠ nil & no(encontrado) hacer
       si aux^.art = a entonces
            encontrado: = cierto;
       sino aux:= aux^.sig;
       fsi;
  fmientras;
  si encontrado entonces
       c:= aux^.cant;
  sino
       c:= 0;
  fsi;
  retornar c
ffunción

función valor_inv (ent i:Inventario) retorna v: nat;
Pre: cierto
Post:v = i.val_inv
retornar i.val_inv;
ffunción

función hash (ent a:nat) retorna p: nat;
Pre: cierto
Post:1 ≤ p ≤ N.
ffunción
```

Justificación de la implementación:

Invariante de la representación:

Para cada j ($1 \leq j \leq N$), i.t[j] apunta a una lista de nodos que cumplen que sus campos cant son distintos de 0 y corresponden a la cantidad de ejemplares del artículo del mismo nodo en el inventario; y si a es el contenido del campo art de un nodo de la lista, entonces:

1) a no aparece repetido en otro nodo de la lista.
2) hash(a)=j.

i.val_inv contiene el valor de los artículos del inventario.

Es fácil ver que el invariante siempre se cumple. Después de inicializar un inventario. Para cada j ($1 \leq j \leq N$), i.t[j] apunta a una lista vacía que, obviamente, cumple las condiciones que se dan en el invariante. Por otra parte, i.val_inv=0, que es el valor de los artículos del inventario (no hay ninguno).

Si suponemos que el invariante se cumple antes de realizar un envío, de acuerdo con la postcondición, después de realizarlo tenemos que:

1. Si el artículo a estaba en la tabla, en la tabla de listas sólo hemos cambiado su cantidad asociada (sumando c a su cantidad); Por tanto, a) todas las cantidades serán mayores que 0, ya que c es un natural y siguen correspondiendo a la cantidad de ejemplares del artículo del mismo nodo en el inventario; b) no habrá artículos repetidos; y c) cada artículo seguirá en la lista de sinónimos que le corresponde.

2. Si el artículo a no estaba en la tabla, y c no es 0 se ha incluido <a,c> como elemento de la lista apuntada por i.t[hash(a)]. Evidentemente también se siguen cumpliendo las condiciones sobre i.t que figuran en el invariante.

3. Si c es 0, hemos dejado el inventario intacto, por lo que se sigue cumpliendo el invariante

4. val_inv se ha incrementado en c*valor(a) por lo que sigue manteniendo el valor global de los artículos en el inventario.

Equivalencia de la representación

Dos inventarios i1 e i2 son equivalentes, $i1 \equiv i2$ si para todo j ($1 \leq j \leq N$), i1.t[j] e i2.t[j] contienen los mismos artículos con las mismas cantidades, aunque quizás en distinto orden.

Finalmente, podemos ver que la implementación cumple los axiomas de la especificación:

```
envío(i, art,0) ≡ i
```

Si la cantidad es 0, envío no hace nada, por lo que se cumple el axioma.

```
envío(envío(i,art,n),art,m) ≡ envío(i,art,n+m)
```

Si enviamos el artículo art primero con cantidad n y luego con cantidad m, tenemos dos casos. Si antes del primer envío art no estaba en el inventario, después de este envío lo habremos insertado en la lista de sinónimos correspondiente con la cantidad n y, después del segundo envío habremos incrementado esta cantidad en m unidades. Esto es lo mismo que insertar directamente el artículo con n+m unidades. Si art ya estaba en el inventario, entonces las dos operaciones de envío habrían incrementado su cantidad primero en n y luego en m unidades. Esto es lo mismo que incrementar, directamente, la cantidad en n+m unidades. Por otra parte, algo similar sucede con val_inv.

```
envío(envío(i,art,n),art',m)≡ envío(envío(i,art',m),art,n)
```

Si enviamos el artículo art con cantidad n y luego art' con cantidad m, lo más que nos puede ocurrir es que los nodos correspondientes a los dos artículos aparezcan en las listas correspondientes en distinto orden, pero de acuerdo con la definición los dos inventarios serían equivalentes.

```
existencias(crear_inv,art) = 0
```
Un inventario recién creado no contiene ningún artículo, por lo que art no estará en él. De acuerdo con la postcondición de existencias, el resultado será 0.

```
existencias(envío(i,art,n),art) = n+ existencias(i,art)
```
Si acabamos de enviar el artículo art, éste estará en su lista de sinónimos y, de acuerdo con el invariante, su valor asociado es el número de existencias de ese artículo, que es lo que obtendríamos como resultado. Por otra parte, si m era el número de existencias de art antes de hacer el envío, el número total de existencias después del envío es n+m.

```
art≠art' =>
     existencias(envío(i,art,n),art')= existencias(i,art)
```
El cálculo de las existencias de un artículo es independiente de que, previamente, se haya realizado un envío de otro artículo.

```
valor_inv(crear_inv)= 0
```
Consecuencia directa de la postcondición de crear_inv.

```
valor_inv(envío(i,art,n))= n*valor(art)+ valor_inv(i)
```

De acuerdo con el invariante, el valor de los artículos de un inventario es el contenido del campo val_inv, que es lo que devuelve la operación. Por otra parte, cuando se hace un envío de n ejemplares del artículo art, se incrementa val_inv en n*valor(art) que es lo que tenemos en la parte derecha de la ecuación.