Examen Parcial de PRED (9/5/2007)

1. **(1.5 puntos)** Dado el siguiente esquema de programa en un lenguaje con estructura de bloques:

```
acción P (X: entero)
     var Y1: entero
     acción P2 (X1: entero)
     … /* punto 1 */...
     facción
     acción P3 (X2: entero)
           var Y3: entero
           acción P (X1: entero)
                 … /* punto 2 */...

           facción
     … /* punto 3 */...
     facción
… /* punto 4 */...
facción
```

Contestad a las siguientes preguntas razonando las respuestas:

a) Si en el punto 1 apareciera la instrucción P(X1), sería correcta?. En caso de que lo fuera, a qué P se llamaría y quien sería X1?
b) Si en el punto 2 apareciera la instrucción P2(X1), sería correcta y quien sería X1?.
c) Si en el punto 3 apareciera la instrucción P(X), sería correcta?. En caso de que lo fuera, a qué P se llamaría?
d) Si en el punto 4 apareciera la instrucción P(X), sería correcta?. En caso de que lo fuera, a qué P se llamaría?

2. **(1.5 puntos)** Dadas las declaraciones:

```
clase C
            privado x: entero;
            función F(y: entero) retorna C1
            ...
            ffunción
fclase
clase C1 subclase de C
            x1: entero;
            función F1(y: entero) retorna C1
            ...
            ffunción
fclase
clase C2 subclase de C1
            x2: entero;
            función F(y: entero) retorna C1
            ...
            ffunción
fclase
clase C3 subclase de C
            x2: entero;
            función F(y: entero) retorna C1
            ...
            ffunción
fclase
```

Supongamos que las reglas que definen la herencia, los tipos y la visibilidad son las de Java y que tenemos las variables `a: C; a1: C1; a2: C2; a3: C3` y supongamos que `v, v1, v2` y `v3` son valores de las clases `C, C1, C2` y `C3`, respectivamente. En este contexto, cuáles de las siguientes secuencias de instrucciones serían correctas y por qué, suponiendo que las instrucciones se encuentran dentro de la función F1 de C1:

a) `a:= a.F(a.x);`
b) `a1:= a1.F(a1.x);`
c) `a2:= v1; a2.x1:=3;`
d) `a1:=v2; a1:=a1.F(a1.x2);`
e) `a:=v3; a3:=a.F(a2.x2);`

Además, en el caso de las llamadas que consideres correctas al método `F`, ¿a qué `F` se llamaría?.

3. (1 punto) Supongamos que tenemos definiciones de funciones con los siguientes tipos:

```
f: t1 x t1 --> t1      f: t2 x t2 --> t2
g: t2 x t2 --> t1      g: t1 x t1 --> t2
```

siendo `t1` subtipo de `t2`. Suponiendo que `a1:t1, a2:t2,` decid si podemos inferir lo siguiente (y cómo):

`a) f(g(a1,a2),f(a1,a2)): t1` `b) f(g(a1,a2),f(a1,a2)): t2`

4. (1.5 puntos) Suponiendo que se ejecuta el siguiente programa P ¿cuales serían los resultados que se escribirían según el paso de parámetros sea por valor o referencia?. Suponed que los argumentos y los resultados se evalúan y se pasan de izquierda a derecha.

```
acción P ()
     var v,i: entero; a: tabla [0..3] de entero;
     acción Q (X,Y:entero)
           X:= X+1;
           Y:= Y+1;
     facción
     función f(X:entero) retorna entero
           v:= v+1;
           X:= X+1;
           retorna X
     ffunción

     para i:= 0 a 3 hacer a[i]:= i; fpara
     v:=0;
     Q(a[f(v)],v);
     escribir (v);  escribir (a);
facción
```

5. (1.5 puntos) Supongamos que tenemos en Java una clase que nos implementa listas de pares de números con las siguientes declaraciones:

```
class par {
      int X;
      int Y; }

class nodo {
      par P;
      nodo sig;}

class lista {
      nodo primero;

      void insertar(par PP){
         nodo nuevo = new nodo (); nuevo.P = PP;  nuevo.sig = primero;
         primero = nuevo; }
      /* resto de la clase lista */  }
```

Supongamos que queremos crear una lista L que contenga los pares <1,1> y <3,3> (sin importarnos el orden en que aparezcan los elementos). ¿Sería correcto usar el siguiente código Java?:

```
lista L = new lista ();
par PP = new par ();
PP.X = 1; PP.Y = 1; L.insertar(PP);
PP.X = 3; PP.Y = 3; L.insertar(PP);
```

6. (3 puntos) Especificar una abstracción de datos que nos defina un Laberinto. En concreto, se considera que un laberinto está formado por una serie de habitaciones y puertas para pasar de una a otra. Específicamente, se desea tener las siguientes operaciones:

- l_vacio: nos crea el laberinto vacio
- Añadir_hab: Dado un laberinto L y una habitación H, nos añade la habitación al laberinto.
- Añadir_puerta: Dado un laberinto L, y dos habitaciones H1 y H2, nos añade una puerta que permite pasar de la habitación H1 a la habitación H2 (pero no al reves).
- hay_salida?: Dado un laberinto L y una habitación H, nos dice si en L hay una puerta que nos permite salir de la habitación H.
- hay _camino?: Dado un laberinto L y dos habitaciones H1 y H2, nos dice si es posible ir desde la habitación H1 a la habitación H2, quizá pasando por otras habitaciones.

Soluciones:

1. a) Es correcta. P es la acción más externa. X1 es el parámetro formal de P2.
b) Es correcta. X1 es el parámetro formal de P.
c) Es correcta. P es la acción más interna.
d) Es correcta. P es la acción más externa.

2. a) Es incorrecta. El atributo x no es visible en la función F1 de C1.
b) Es incorrecta. El atributo x no es visible en la función F1 de C1.
c) Es incorrecta. A a2 no se le puede asignar un valor de clase C1.
d) Es incorrecta. El objeto a1 no tiene el atributo x2.
e) Es incorrecta. A a3 no se le puede asignar un valor de clase C1.

3. a) No es posible inferir `f(g(a1,a2),f(a1,a2)): t1`. Para ello, teniendo en cuenta las definiciones de la función f, habría hecho falta inferir `g(a1,a2): t1` y `f(a1,a2): t1`. Sin embargo esto último no es posible, ya que no se puede inferir `a2:t1`.

b). Sí es posible inferir `f(g(a1,a2),f(a1,a2)): t2`. Por una parte

```
 a1:t1 t1<t2
_____________
   a1: t2          a2:t2   g: t2 x t2 --> t1
_____________________________________________
               g(a1,a2): t1
```

Por otra:

```
a1:t1 t1<t2
___________
  a1: t2          a2:t2        f: t2 x t2 --> t2
_____________________________________________
                 f(a1,a2): t2
```

Finalmente:

```
g(a1,a2): t1   t1<t2
___________________
    g(a1,a2): t2        f(a1,a2): t2   f: t2 x t2 --> t2
_____________________________________________________
               f(g(a1,a2),f(a1,a2)): t2
```

4. Si el paso es por valor se escribiría: 1 0 1 2 3. Si el paso es por referencia se escribiría 3 0 1 3 3.

5. El código Java propuesto no funcionaría correctamente. Después de hacer las dos inserciones la lista contendría dos nodos que contendrían el par <3,3>. El motivo es que la asignación `nuevo.P = PP;` hace que el nuevo nodo pase a ser un alias de PP. Como consecuencia, si se cambia de valor PP (por ejemplo para hacer una nueva inserción) también se modificaría el nodo de la lista. Con esa implementación de la operación insertar, el código adecuado sería (por ejemplo) el siguiente:

```
lista L = new lista ();
par PP = new par ();
PP.X = 1; PP.Y = 1; L.insertar(PP);
PP = new par ();
PP.X = 3; PP.Y = 3; L.insertar(PP);
```

De todas formas, en realidad lo que es incorrecto es la operación de insertar. Lo correcto habría sido que en esta operación se substituyera la instrucción `nuevo.P = PP;` por: `nuevo.P.X = PP.X; nuevo.P.Y = PP.Y;`.

6. La especificación pedida tendría como signatura:

```
especificación Laberinto
tipos Laberinto, habitación, bool
operaciones
L_vacio: -> Laberinto
añadir_hab: Laberinto x habitación -> Laberinto
```

```
añadir_puerta: Laberinto x habitación x habitación -> Laberinto
hay_salida?: Laberinto x habitación -> bool
hay_camino?: Laberinto x habitación x habitación -> bool
```

donde serían constructores o generadores `L_vacio, añadir_hab,` y `añadir_puerta.` Con respecto de los axiomas tenemos unas cuantas alternativas. En primer lugar, podemos considerar que es irrelevante el orden en que se añaden las habitaciones o las puertas o que añadir por segunda vez la misma operación o la misma puerta no produce ningún efecto. Es decir, podríamos tener los axiomas:

```
añadir_hab(añadir_hab(L,H),H)= añadir_hab(L,H)
añadir_hab(añadir_hab(L,H1),H2)= añadir_hab(añadir_hab(L,H2),H1)
añadir_puerta(añadir_puerta(L,H1,H2),H1,H2)= añadir_puerta(L,H1,H2)
añadir_puerta(añadir_puerta(L,H1,H2),H3,H4)=
añadir_puerta(añadir_puerta(L,H3,H4),H1,H2)
```

También podríamos omitir estos axiomas. Por otra parte, podríamos considerar que sólo se puede añadir una puerta entre las habitaciones H1 y H2 si, previamente, se han añadido las habitaciones H1 y H2. La forma más simple de hacer esto es utilizar una función auxiliar que nos diga si una habitación o no está en el laberinto cuya especificación sería:

```
hay_hab?: Laberinto x habitación -> bool
axiomas
          hay_hab(L_vacio) = falso
          hay_hab(añadir_hab(L,H),H) = cierto
H1≠H2  ⇒  hay_hab(añadir_hab(L,H1),H2) = hay_hab(L,H2)
          hay_hab(añadir_puerta(L,H1,H2),H) = hay_hab(L,H)
```

Con ayuda de esta operación, podríamos especificar cuándo la función añadir_puerta está indefinida con el axioma:

```
¬ hay_hab(L,H) ∨ ¬ hay_hab(L,H) ⇒ añadir_puerta(L,H1,H2) = indef.
```

Alternativamente, podríamos considerar que se puede añadir una puerta aunque no se hayan añadido las habitaciones. Por último, también podríamos considerar que se pueden conmutar las operaciones de añadir_puerta y añadir_hab, al menos cuando se refieran a habitaciones distintas:

```
H1≠H2 ∧ H1≠H3   ⇒   añadir_puerta(añadir_hab(L,H1),H2,H3)=
                     añadir_hab(añadir_puerta(L,H2,H3),H1)
```

Por último, los axiomas para definir las operaciones hay_salida? y hay_camino? en su versión más sencilla (si se considera que la operación añadir puerta es parcial serían un poco más complicadas) serían:

```
          hay_salida?(L_vacio, H)= falso
          hay_salida?(añadir_hab(L,H1),H2)= hay_salida?(L,H2)
          hay_salida?(añadir_puerta(L,H1,H2),H1)= cierto
H1≠H3  ⇒ hay_salida?(añadir_puerta(L,H1,H2),H3)= hay_salida?(L,H3)
```

Alternativamente, también podríamos considerar que el primer axioma podría decir que el resultado de ver si hay salida de una habitación cuando el laberinto está vacío está indefinido, pero entonces habría que complicar un poco el segundo axioma. Por otra parte los axiomas de hay_camino? serían:

```
hay_camino?(L_vacio,H1,H2)= falso
hay_camino?(L,H,H)= cierto
hay_camino?(añadir_puerta(L,H1,H2),H1,H2)= cierto
hay_camino?(añadir_hab(L,H),H1,H2)= hay_camino?(L,H1,H2)
hay_camino?(añadir_puerta(L,H3,H4),H1,H2)=
(hay_camino?(L,H1,H3) ∧ hay_camino?(L,H4,H2)) ∨ hay_camino?(L,H1,H2)
```

aunque el tercer de estos últimos axiomas no sería estrictamente necesario, ya que es consecuencia del segundo y del último.