Examen Parcial de PRED (30/11/2005)

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

```
acción P1 (X1: entero)
     var Y1: entero
     acción P2 (X2: entero)
           var Y2: entero
           acción P3 (X3: entero)
                 var Y3: entero
                 …
           facción
     …
     facción
     acción P4 (X4: entero)
           var Y4: entero
           acción P2 (X2: entero)
                 var Z2: entero
                 …
           facción
     …
     facción
…
facción
```

Decid cuales de las siguientes llamadas podrían estar en la acción `P4` y cuales no (indicando el por qué). Además, en el caso de las llamadas a `P2` que sean correctas, decid a cual de los dos `P2` se llamaría.

a) `P2(X1)`
b) `P2(X2)`
c) `P2(Y2)`
d) `P2(Z2)`

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

```
clase C
            x: entero;
            función F(y: entero) retorna z: C
            ...
            ffunción
fclase
clase C1 subclase de C
            x1: entero;
            función F1(y: entero) retorna z: C1
            ...
            ffunción
fclase
clase C2 subclase de C1
            x2: entero;
            función F(y: entero) retorna z: C
            ...
            ffunción
            función F2(y: entero) retorna z: C
            ...
            ffunción
fclase
clase C3 subclase de C
            x2: entero;
            x3: entero;
            función F(y: entero) retorna z: C3
            ...
            ffunción
fclase
```

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

a) `a1:=v2; a1:= a1.F(a.x)`

b) `a:=v2; a:= a.F(a.x)`
c) `a:= v3; a:=a.F(a2.x2)`
d) `a1:=v; a=a1.F(a1.x);`
e) `a2:= a2.F1(a1.x);`
f) `a:=v2; a:= a.F2 (a.x)`

Además, en el caso de las llamadas que consideres correctas al método `F`, ¿a qué `F` se llamaría?
¿al de `C` al de `C2` o al de `C3`?

3. **(1.5 puntos)** Supongamos que tenemos definiciones de funciones con los siguientes tipos:

```
f: t2 x t3 --> t2
f: t3 x t2 --> t3
g: t2 x t3 --> t3
g: t3 x t2  --> t2
```

siendo `t1` subtipo de `t2`, `t1` subtipo de `t3`, `t2` subtipo de `t4` y `t3` subtipo de `t4`. Decid si podemos inferir lo siguiente (y cómo):

```
a) f(f(a1,a2),g(a1,a1)): t4
b) f(g(a1,a2),f(a2,a1)): t4
```

suponiendo que `a1:t1, a2:t2, a3:t3`.

4. **(1 punto)** Supongamos que tenemos declarada la siguiente acción en Java:

```
static void Inic1(int [] a, int [] b)
{int i;
 for (i=1; i < 100; i++)  a[i] = 1;
 b = a;
}
```

Supongamos además que tenemos el siguiente main:

```
public static void main (String[] args) throws Exception
{int [] a1 = new int[100];
 int [] b1 = new int[100];
 Inic1(a1,b1);
 System.out.print(a1[2]);
 System.out.print(b1[2]);
}
```

¿Qué valores se escribirían?. Y ¿qué valores se escribirían si Inic1 estuviera definido como sigue?

```
static void Inic1(int [] a, int [] b)
{int i;
 for (i=1; i < 100; i++)  a[i] = 1;
 for (i=1; i < 100; i++)  b[i] = 1;
}
```

Justificar la respuesta.

5. **(1.5 puntos)** Dado el siguiente programa

```
acción P1 (X: entero)
     var A: entero
     acción P2 (X:entero)
           var A: entero
           A:= 3;
           P3(A);
           X:= X+A;
     facción
     acción P3 (Y:entero)
           Y:= Y+1;
           X:= X+1;
           A:= X+A;
     facción
     A:= X+1;
     P2(A);
     X:= A;
     escribir (X);
facción
```

Suponiendo que se ejecutan las instrucciones B:= 1; P1(B), ¿qué se escribirá al final de la ejecución de P1 si el paso de parámetros se produce por valor-resultado? ¿y si se produce por referencia?.

6. **(3 puntos)** Se desea especificar una estructura de datos que sirva para representar una familia. En concreto, esta estructura ha de tener las siguientes operaciones:

- `f_vacía`: nos crea la familia vacía.
- `añadir`: dado un objeto F de tipo familia y dos personas P1 y P2, nos añade a F la información de que P1 es madre o padre de P2. Evidentemente, en F, P1 podría ser ya madre o padre de otra persona P3. Asimismo, se supone que P2 podría ya tener otro(s) progenitores en F. Por otra parte, se supone que el orden en que se añaden estas relaciones de parentesco es importante, por lo que no es lo mismo añadir a F primero P1 y P2 y luego P1' y P2' que hacerlo a la inversa.
- `primogénito`: dado un objeto F de tipo familia y dada una persona P nos devuelve el hijo mayor de P en F, es decir el primer hijo de P que se añadió a F.
- `antecesores`: dado un objeto F de tipo familia y dada una persona P nos devuelve el conjunto de antecesores (madre, padre, abuelos, etc) de P en F. Para especificar esta propiedad se pueden utilizar las operaciones sobre conjuntos que se vieron en clase.

Especificad esta abstracción. Más concretamente, se pide:

a) Decir qué posibles conjuntos de operaciones constructoras o generadoras podríamos elegir (no hay que especificar axiomas entre los constructores porque, como se dice más arriba, importa el orden en que se añaden las relaciones de parentesco).
b) Especificar el resto de las operaciones.
c) Por último, utilizando los apartados anteriores, escribir la especificación completa.

Soluciones

Solución a la pregunta 1

a) Correcto. Tanto P2 como X1 son visibles en P4. Se llamaría al P2 que está dentro de P4.
b) Incorrecto. Ninguno de los X2 declarados son visibles en P4
c) Incorrecto. Y2 no es visible en P4
d) Incorrecto. Z2 no es visible en P4

Solución a la pregunta 2

a) a1:= v2; a1:= a1.F(a.x). Incorrecto: a1.F(a.x) es de tipo C, luego no puede asignarse a a1.
b) Correcto. La F llamada es de C2.
c) Correcto. La F llamada es de C3.
d) Incorrecto: a a1 no se le puede asignar v.
e) Incorrecto: a2.F1(a1.x) es de tipo C1, luego no puede asignarse a a2.
f) Incorrecto: en Java el tipo de a es C y F2 no pertenece a esa clase.

Solución a la pregunta 3

a) Por una parte:

$$\frac{\dfrac{a1:t1 \quad t1<t3}{a1:\ t3} \qquad a2:t2 \quad f:\ t3 \times t2 \rightarrow t3}{f(a1,a2):\ t3}$$

Por otra:

$$\frac{\dfrac{a1:t1 \quad t1<t3}{a1:\ t3} \qquad \dfrac{a1:t1 \quad t1<t2}{a1:t2} \qquad g:\ t3 \times t2 \rightarrow t2}{g(a1,a1):\ t2}$$

Finalmente:

$$\frac{f(a1,a2):\ t3 \qquad g(a1,a1):\ t2 \qquad f:\ t3 \times t2 \rightarrow t3 \qquad t3 < t4}{f(f(a1,a2),\ g(a1,a1)):\ t4}$$

b) `f(g(a1,a2), f(a2,a1)): t4` no puede ser inferido. En concreto, sabemos que `g(a1,a2):t2`. Por otra parte, la única declaración de `f` que tiene un primer parámetro de tipo `t2` es `f: t2 x t3 --> t2`. Sin embargo, no podemos demostrar que `f(a2,a1): t3`, ya que la única declaración de `f` que tiene un primer parámetro de tipo `t2` tiene un resultado de tipo `t2`.

Solución a la pregunta 4

En el primer caso se escribiría un uno y un cero. En el segundo dos unos. El motivo es que, en Java, el paso por parámetros es por valor. Esto hace que, en el primer caso, la asignación b = a; no tenga efecto una vez terminada la ejecución de Inici1. Como consecuencia, los elementos de la tabla b contendrán los valores iniciales que asigna Java por defecto. Esto es, estarán todos a cero.

Solución a la pregunta 5

Paso por valor-resultado: 6
Paso por referencia: 8

Solución a la pregunta 6

a) f_vacía y añadir nos permiten definir todas las familias de la especificación.
b y c) Una especificación simple que cumple con lo que se pide en el enunciado es la siguiente:

```
especificación familia
tipos familia, persona, conj_persona
operaciones
f_vacia: -> familia
añadir: familia x persona x persona -> familia
primogénito: familia x persona -> persona (parcial)
antecesores: familia x persona -> conj_persona
axiomas
1) primogénito(f_vacia,P)= indefinido
2) primogénito(añadir(f_vacia,P,P'),P)= P'
3) P ≠ P1 => primogénito(añadir(F,P1,P2),P)= primogénito(F,P)
4) primogénito(añadir(añadir(F,P,P1),P,P2),P)=
    primogénito(añadir(F,P,P1),P)
5) P ≠ P1 => primogénito(añadir(añadir(F,P1,P2),P,P3),P)=
    primogénito(añadir(F,P,P3),P)
6) antecesores(f_vacia,P)= Conj_vacio
7) antecesores(añadir(F,P',P),P) =
    {P'} U antecesores(F,P) U antecesores(F,P')
8) P2∈antecesores(F,P) => antecesores(añadir(F,P1,P2),P) =
    {P1} U antecesores(F,P) U antecesores(F,P1)
9) P ≠ P1 ∧ P2.antecesores(F,P) => antecesores(añadir(F,P1,P2),P) =
    antecesores(F,P)
fespecificación
```