Examen Parcial de PRED

1. **(1.5 puntos)** Dado el siguiente esquema de programa:

```
acción P1 (X: entero)
    var X1: entero
    acción P2 (X: entero)
         var X2: entero
         acción P3 (X: entero)
              var X3: entero

         …
         facción
    …
    facción
    acción P4 (X: entero)
         var X4: entero
         acción P5 (X: entero)
              var X5: entero

         …
         facción
              …
    facción
…
facción
```

¿Cuales de las siguientes llamadas podrían estar en la acción P5 y cuales no (indicando el por qué)?

a) P2(X2)
b) P2 (X1)
c) P4 (X5)
d) P3 (X3)
e) P1(X)

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

```
    clase C
          x: entero;
          acción P(y: entero)
          ...
          facción
    fclase

    clase C1 subclase de C
          x1: entero;
          acción P1(y:entero)
          ...
          facción
```

```
fclase

clase C2 subclase de C1
       x2: entero;
       acción P(y: entero)
       ...
       facción
       acción P2(y:entero)
       ...
       facción
fclase

clase C3 subclase de C
       x2: entero;
       x3: entero;
       acción P(y: entero)
       ...
       facción
       acción P3(y:entero)
       ...
       facción
fclase
```

¿Hay algo erróneo en estas declaraciones?. Si lo hay, suprimidlo. Supongamos, además que hemos declarado las variables a: C; a1: C1; a2: C2; a3: C3. En este contexto, cuales de las siguientes llamadas serían correctas:

a) a.P1(a1.x1)
b) a1.P(a1.x)
c) a1.P3(a3.x3)
d) a.P(a3.x2)
Además, en el caso de las llamadas que consideres correctas al método P, ¿a qué P se llamaría? ¿al de C al de C2 o al de C3?

3. **(1 punto)** Dadas las siguientes definiciones de tipos:

```
f: nat x nat --> nat
f: int x nat --> int
g: real x real --> int
g: nat x real --> nat
g: int x int --> real
```

siendo `nat` subtipo de `int`, e `int` subtipo de `real`. Decir si podemos inferir lo siguiente (y cómo):

```
f(g(-2,2.5),g(2,2.5)): real
f(g(-2,2.5),g(2,2.5)): int
```

sabiendo que, obviamente, `2: nat, -2: int, 2.5: real`.

4. **(1 punto)** En algunos lenguajes de programación, por ejemplo C, se permite hacer que un puntero apunte a una variable cualquiera. Por ejemplo, si tenemos las declaraciones:

```
var  p: ^entero;
     x: entero;
```

en estos lenguajes, la asignación

```
p:= direcc(x)
```

ocasionaría que p contuviera la dirección de x, es decir apuntara a x. Por ejemplo, si ahora hiciéramos `p^ := p^ + 1;` esto incrementaría el valor de x en 1. En este contexto, ¿qué ocurriría si liberáramos la memoria apuntada por p, es decir si pidiéramos ejecutar `free(p)`?

5. **(1.5 puntos)** Supongamos que queremos implementar una acción que nos intercambie los valores apuntados por dos variables de tipo puntero, es decir, con la siguiente especificación:

```
acción intercambio (ent/sal X,Y:^entero)
Pre: (X^ = m) & (Y^ = n)
Post: (X^ = n) & (Y^ = m)
facción
```

Supongamos que nos dan las siguientes alternativas:

a)
```
var aux :^entero
new(aux);
aux^:= X^;
X^:= Y^;
Y^:= aux^;
free(aux)
```

b)
```
var aux :^entero
new(aux);
aux:= X;
X:= Y;
Y:= aux;
free(aux)
```

c)
```
var aux :^entero
new(aux);
aux:= X;
X:= Y;
Y:= aux;
```

¿Qué alternativa elegirías y por qué?

6. **(3.5 puntos)** Se desea especificar un tipo de secuencias de enteros con las operaciones:

s_vacía: que nos devuelve la secuencia vacía

añadir: dada una secuencia S y un entero N, nos devuelve la secuencia en que hemos añadido el entero.
concatenar: dadas dos secuencias S1 y S2, nos devuelve la secuencia en que hemos colocado todos los elementos de S2 después de los de S1.
suma: dada una secuencia S, nos devuelve la suma de los elementos que contiene.

Se pide:

a) Elegir un conjunto de constructores.
b) Enunciar los axiomas que nos definen las relaciones entre constructores
c) Definir el resto de las operaciones
d) Escribir la especificación completa de este tipo de secuencias.

Soluciones

1)

a) P2(X2): No, la variable X2 no es visible desde P5
b) P2 (X1): Sí, tanto la acción P2 como la variable X1 son visibles desde P5
c) P4 (X5): Sí, tanto la acción P4 como la variable X5 son visibles desde P5
d) P3 (X3): No, ni P3 ni X3 son visibles desde P5
e) P1(X): Sí, tanto la acción P1 como X son visibles desde P5

2)

a) a.P1(a1.x1). En principio, incorrecta. La clase C (de a) no contiene el método P1. Sin embargo, podría ser correcta en un lenguaje con tipificación dinámica si a contuviera un valor de clase C1 (o C2).
b) a1.P(a1.x). Correcta. La clase C1 incluye el método P y el atributo x (heredados de C)
c) a1.P3(a3.x3). Incorrecta en cualquier caso. La clase C1 (clase de a1) no incluye el método P3 ni a1 puede contener un valor de clase C3, ya que no es subclase suya.
d) a.P(a3.x2). Correcta.

Por otra parte, las dos llamadas a P son correctas, pero no podemos saber a qué P se llamaría, ya que depende de los valores que estas variables tengan en ejecución. En concreto, en el caso de a1.P(a1.x), si a1 contiene un valor de clase C1, el P llamado sería el heredado de C. Sin embargo, si a1 contiene un valor de su subclase C2, entonces el P llamado sería el de C2. Análogamente, en el caso de a.P(a3.x2), si a contiene un valor de clase C o C1, el P llamado sería el de C. Sin embargo, si a contiene un valor de clase C2 o C3, entonces el P llamado sería el de C2 o el de C3, respectivamente.

3) Por una parte:

```
-2:int  int<real
_________________
    -2:real              2.5:real   g: real x real --> int
_____________________________________________________________
                  g(-2,2.5):int
```

Por otra:

```
2:nat      2.5:real   g: nat x real -> nat
_____________________________________________
             g(2,2.5):nat
```

Finalmente:

```
g(-2,2.5):int   g(2,2.5):nat g   f: int x nat -> int
_____________________________________________________
              f(g(-2,2.5),g(2,2.5)): int
```

Luego ya hemos comprobado que `f(g(-2,2.5),g(2,2.5)): int.` Por otra parte:

```
  f(g(-2,2.5),g(2,2.5)): int        int<real
_____________________________________________
         f(g(-2,2.5),g(2,2.5)): real
```

4) La posición apuntada por p (es decir la que asociada a la variable x) no puede ser liberada, ya que no pertenece al heap, sino a la pila. Esto quiere decir que el intento de ejecutar free(p) o bien no produciría ningún efecto o provocaría un error de ejecución.

5) La opción a) es la mejor de las tres. La segunda opción es errónea ya que al liberar aux Y pasaría a contener una referencia colgada. La tercera opción no es muy adecuada, ya que puede provocar efectos secundarios. En concreto, por ejemplo, si en la acción en que se llama a intercambio hubiera otra variable que también apuntara a la misma posición que X, después de ejecutar intercambio, esta variable pasaría a apuntar a la misma posición que Y. Es decir, además del intercambio de valores de X e Y tendríamos un efecto adicional inesperado.

6)
```
especificación seq
tipos seq, entero
operaciones
     s_vacia: -> seq
     añadir: seq x entero -> seq
     concatenar: seq x seq -> seq
     suma: seq -> entero
axiomas
     concatenar(s_vacia, S) = S
     concatenar(añadir(S1,n),S2) = añadir(concatenar(S1,S2),n)
     suma(S_vacia)=0
     suma(añadir(S,n))=N+suma(S)
fespecificación
```

a) Los constructores son s_vacia y añadir
b) No hay ningún axioma que defina relaciones entre estos consructores
c) Los cuatro axiomas de la especificación que se muestra arriba
d) Véase más arriba.