Modificación de la Práctica de CL **30 de Mayo de 2007**

A continuación se presenta la modificación de la práctica que debéis realizar, junto con una guía de ayuda, que no es obligatorio seguir, y que quizás requiera de adaptaciones en vuestra práctica dependiendo de cómo la hayáis programado. Esta guía contiene también la información de cómo se realizará la evaluación, en base a unos juegos de prueba que podéis encontrar en el racó, en `examens.fib.upc.edu` accediendo a la entrega correspondiente, incluidos en el fichero `infoexamen.tar`. Os recomendamos crear un directorio `/tmp/cl`, copiar allí vuestra práctica, así como `infoexamen.tar`, y ejecutar también allí `tar xf infoexamen.tar`, y seguidamente `tar xf jp.tar`. El fichero `infoexamen.tar` contiene además otros ficheros que os permitirán realizar una autoevaluación. Al final se indica cómo realizar dicha autoevaluación y la entrega via el racó. Es recomendable ir realizando entregas a medida que vuestra práctica vaya superando más juegos de pruebas, incluso hacer una desde el principio.

POR FAVOR, REALIZAD LA AUTOEVALUACIÓN Y UNA ENTREGA PRELIMINAR CUANDO HAYAIS COMPLETADO LA PRIMERA MODIFICACIÓN, Y ASEGURAOS DE HACER LA ÚLTIMA ENTREGA 15 MINUTOS ANTES DEL TIEMPO FINAL DE EXAMEN, MOMENTO EN QUE EL RACÓ QUEDA CERRADO.

Se quiere añadir al lenguaje CL la instrucción

```
Repeat  lista_instrs Until expr_booleana EndRepeat
```

La ejecución del programa:

```
 1: Program
 2:   Vars
 3:     Y Int
 4:   EndVars
 5:     Y := 0
 6:   Repeat
 7:     Y := Y + 1
 8:     Write(Y)
 9:     Write(" ")
10:   Until Y = 6
11:   EndRepeat
12: EndProgram
```

escribe por pantalla:

```
1 2 3 4 5 6
```

También se quiere añadir al lenguaje CL una instrucción de multiasignación

```
[id1, id2, ..., idn] := [expr1, expr2, ..., exprn]
```

que no tiene efectos laterales debido al orden de las asignaciones (de hecho, esa es la definición usual de la asignación múltiple), es decir, la ejecución del programa

```
 1: Program
 2:   Vars
 3:     X Int
 4:     Y Real
 5:   EndVars
 6:   X := 1
 7:   [X, Y] := [3, X]
 8:   Write(X) Write(" ")
 9:   Write(Y)
10: EndProgram
```

escribe por pantalla:

```
3 1.000000
```

Guía y puntuación:

- **3 puntos** de la nota ya vienen dados por pasar el juego de pruebas genérico que se encuentra en el fichero `jpbasic1`, junto con su salida esperada. El resto ya depende estrictamente de las modificaciones que se detallan a continuación.
- Nos ocuparemos primero de la instrucción `repeat-until` (es decir, en estos primeros juegos de prueba no aparece todavía ninguna multiasignación). Elementos obvios a añadir a la práctica básica son:
  - tokens `REPEAT`, `UNTIL` y `ENDREPEAT`.
  - Un nuevo nodo en ast.h para el árbol de sintáxis abstracta: nodo `Nrepeat`.
  - La correspondiente regla sintáctica en el programa PCCTS, que es una nueva instrucción similar a la de `while`.
- Hay que modificar el análisis semántico para que dé los errores correspondientes con la nueva instrucción. Básicamente, hay que añadir un nuevo caso en `TypeCheck` para tratar el nuevo nodo, que es muy similar al tratamiento de `Nwhile`. Necesitaremos un nuevo caso de error: "Condicion de Repeat no-booleana.". Se debería poder pasar el siguiente juego de pruebas (**2 puntos**):

```
 1: Program
 2:   Vars
 3:     x Int
 4:     y Real
 5:   EndVars
 6:   Procedure A(Ref x Real, Ref y Real)
 7:    y := y + 1
 8:   EndProcedure
 9:   Repeat
10:     A(x, y)
11:     Repeat
12:       A(y, 3)
13:     Until y div 3 > x
14:     EndRepeat
15:   Until y + x
16:   EndRepeat
17: EndProgram
```

que escribe por pantalla:

```
L. 10: Parametro 1 con tipos incompatibles.
L. 12: Parametro 2 no-referenciable pero pasado por referencia.
L. 13: Operador Div con tipos incompatibles.
L. 15: Condicion de Repeat no-booleana.
```

- Hay que modificar la generación de código para que la ejecución del siguiente programa sea las que se indica. Se consigue fácilmente adaptando y simplificando convenientemente la traducción del caso `Nwhile` (**1 punto**).

```
 1: Program
 2:   Vars
 3:     Y Int
 4:   EndVars
 5:     Y := 0
 6:   Repeat
 7:     Y := Y + 1
 8:     Write(Y)
 9:     Write(" ")
10:   Until Y = 6
11:   EndRepeat
12: EndProgram
```

escribe por pantalla:

```
1 2 3 4 5 6
```

- Vamos ya con la multiasignación. Necesitamos nuevos tokens `ABRECOR`, `CIERRACOR`, nuevos nodos en ast.h para el árbol de sintáxis abstracta, por ejemplo `Nmultasig, Nl_idents` y `Nl_exprs`, si no los teníamos ya antes, y un nuevo tipo de instrucción (la multiasignación),

  ```
  ABRECOR l_idents CIERRACOR ASIG ABRECOR l_exprs CIERRACOR
  ```

  Necesitamos también dos nuevos no terminales de la gramática para listas de identificadores y listas de expresiones, separadas por comas, junto con sus correspondientes partes derechas de regla, si no los teníamos ya antes.

- Para el análisis semántico, hay que detectar los errores relacionados con la nueva instrucción. Concretamente, necesitaremos de dos nuevos tipos de error, `"Falta variable en la multiasignacion"` y `"Sobra variable en la multiasignacion"`.

  Conviene hacer una función aparte (llamada desde el correspondiente caso `Nmulsiasig` del `TypeChek` que reciba un AST con la lista de variables, un AST con la lista de expresiones, la linea actual de programa (para indicar los posibles errores), que compruebe que haya la misma cantidad de variables y expresiones, que vaya llamando al `typecheck` de las expresiones, y que verifique que estas son asignables a las correspondientes variables; que a su vez deberán estar declaradas. Para simplificar el examen, no es necesario comprobar que no haya variables repetidas en la parte izquierda de la asignación.

  1. Debería poder pasar el siguiente juego de pruebas (**1 punto**):

     ```
      1: Program
      2:   Vars
      3:     X Int
      4:     Y Real
      5:     Z Int
      6:     T Int
      7:   EndVars
      8:   Function F(Val X Int, Ref Y Real) Return Int
      9:     Return 3
     10:   EndFunction
     11:   [Z, X, T] := [X+F(X,3.1), 1, X]
     12:   [X, Z] := [1]
     13:   [X, X] := [X, 1, X*X+5]
     14: EndProgram
     ```

     que debería dar los siguientes errores semánticos.

     ```
     L. 11: Parametro 2 no-referenciable pero pasado por referencia.
     L. 12: Sobra variable en la multiasignacion.
     L. 13: Falta variable en la multiasignacion.
     ```

2. Y también el siguiente (**1 punto**).

```
 1: Program
 2:   Vars
 3:     X Int
 4:     Y Real
 5:     Z Int
 6:     T Int
 7:   EndVars
 8:   Function F(Val X Int, Ref Y Real) Return Int
 9:     Return 3
10:   EndFunction
11:   [W, X] := [1, X+true]
12:   [X, Y] := [3*Z+Z, F(X,Y)]
13:   [X, Y] := [3*Z+Z, F(X,X+Y)]
14:   [X, Y] := [3*Y+Z, F(X,Y)]
15:   [X, Z, Y] := [X, 1, X*X+5]
16:   [X, F, Y] := [X, 1, X*X+5]
17: EndProgram
```

que debería dar los siguientes errores semánticos.

```
L. 11: Identificador W no declarado.
L. 11: Operador + con tipos incompatibles.
L. 13: Parametro 2 no-referenciable pero pasado por referencia.
L. 14: Asignacion con tipos incompatibles.
L. 16: Identificador F no es asignable.
```

- Para la generación de código, nuevamente, hay que recorrer al mismo tiempo la lista de variables y la lista de expresiones. Para cada par $\{v, e\}$, hay que poner la dirección de la variable y la evaluación de la expresión en la pila por ese orden (pero no hacer un `stor` todavía). Una vez se han empilado todos los pares dirección-valor, hay que hacer entonces tantos `stor` como variables haya (fijaos que, haciendo los `stor` al final, las evaluaciones de las expresiones no se han visto afectadas por estos, evitando así los efectos laterales).

  1. **(1 punto)**
     La ejecución del Q-programa que compila este CL-programa

     ```
      1: Program
      2:   Vars
      3:     X Int
      4:     Y Real
      5:   EndVars
      6:   X := 1
      7:   [X, Y] := [3, X]
      8:   Write(X) Write(" ")
      9:   Write(Y)
     10: EndProgram
     ```

     será la siguiente:

     ```
     3 1.000000
     ```

  2. **(1 punto)**
     La ejecución del Q-programa que compila este CL-programa

     ```
      1: Program
      2:   Vars
      3:     X Int
      4:     Y Int
      5:     Z Real
     ```

```
 6:   EndVars
 7:   Function FACT(Val X INT) Return Int
 8:   Vars
 9:     F Int
10:   EndVars
11:     If X = 1 Then F := 1
12:     Else F := X* FACT(X-1)
13:     EndIf
14:     Return F
15:   EndFunction
16:   Procedure P(Ref T Real,Val D Int)
17:     [D,T,Y]:=[5*D,2*D+T,D+1]
18:   EndProcedure
19:   X := 3
20:   [X, Y, Z] := [FACT(X), FACT(X+1), FACT(X+2)]
21:   WriteLn(X)
22:   WriteLn(Y)
23:   WriteLn(Z)
24:   P(Z,X)
25:   WriteLn(X)
26:   WriteLn(Y)
27:   WriteLn(Z)
28:   P(Z,Y)
29:   WriteLn(X)
30:   WriteLn(Y)
31:   WriteLn(Z)
32: EndProgram
```

será la siguiente:

```
6
24
120.000000
6
7
132.000000
6
8
146.000000
```

Autoevaluación: Basta con ejecutar `./fesentrega.sh` para crear automáticamente un fichero llamado `entrega.tar`. Si este comando emite algún mensaje, informad inmediamamente a los profesores de la asignatura, pues aun cuando el resto parezca funcionar, vuestra entrega no se realizará correctamente.

Si ejecutáis entonces `./checker.sh`, al cabo de un rato os indicará qué juegos de pruebas habéis pasado junto con la correspondiente nota final. La evaluación definitiva se hará con ligeras modificaciones de dichos juegos de pruebas. De hecho, el `checker` es una ayuda bastante fidedigna para comprobar que vuestras modificaciones funcionan, pero queda bajo vuestra responsabilidad el comprobar que emitís los errores esperados y que no dais errores absurdos de más.

Entrega: Conectaros en prácticas via web a `examens.fib.upc.edu`. Debéis entregar el fichero `entrega.tar` creado tal y como se indica en la autoevaluación.