UNIVERSITAT JAUME·I

4º Ingeniería Informática

II26 Procesadores de lenguaje

Control de la práctica de miniint (17 de abril de 2010)

INSTRUCCIONES:

- La duración del examen es de dos horas.
- Antes de empezar, asegúrate de que el usuario con el que estás trabajando coincide con el del recuadro del final de este enunciado.
- Rellena el recuadro con tus datos.
- Ejecuta el programa `./prepara.py` pasándole como parámetro tu DNI. Este creará un fichero con tu DNI y nombre y un directorio llamado `miniint`, donde debes guardar tu intérprete.
- Al introducir el USB, debería montarse automáticamente. Si fuera necesario, puedes montarlo y desmontarlo mediante `mount` y `umount` sobre el directorio apropiado.
- Cuando termines el examen, entréganos esta hoja.

EJERCICIO (1 PUNTO)

El objetivo de este ejercicio es que tu intérprete MINIINT acepte, además de lo que se pedía en la práctica 3, sentencias iterativas con un bucle con condición final y con una condición opcional en su cuerpo. Para ello, introduce en el esquema de traducción la siguiente producción correspondiente al bucle `repetir`[1]:

```
1  <Sentencia> ->
2    repetir
3      @sentencias1= []@
4    ( <Sentencia> @sentencias1.append(Sentencia1.arb)@ )*
5      @condicion1= None@
6      @sentencias2= None@
7    ( salir si <Expresion> ";" @condicion1= Expresion1.arb@
8        @sentencias2= []@
9      ( <Sentencia> @sentencias2.append(Sentencia2.arb)@ )*
10   )?
11   hasta <Expresion> ";" @condicion2= Expresion2.arb@
12     @Sentencia.arb= AST.NodoRepetir(sentencias1, condicion1, sentencias2, condicion2, repetir.nlinea)@
13   ;
```

y añade `"hasta"`, `"repetir"` y `"salir"` a la lista de palabras reservadas que se encuentra en el mismo fichero. Tras modificar ese fichero, debes compilarlo de nuevo utilizando, por ejemplo, `../metacomp/metacomp miniint.mc -s miniint`. Sin modificar nada más en el esquema de traducción, debes implementar lo necesario para poder ejecutar programas en los que se utilice ese tipo de sentencias. El significado de la construcción

```
repetir
  sentencias1
salir si condición1 ;
  sentencias2
hasta condición2 ;
```

es que en cada iteración del bucle, incluyendo la primera, que siempre se da, sucede lo siguiente:

- Se ejecutan las *sentencias1*;
- si existe, se evalúa *condición1*; si es cierta, se interrumpe el bucle, en caso contrario se sigue adelante;

[1] Tienes esta producción en el fichero `produccionBucle.mc` de tu *home*.

- si existen, se ejecutan las *sentencias2*;
- se evalúa *condición2*; si es cierta, se interrumpe el bucle, en caso contrario se inicia una nueva iteración.

En tu implementación, debes tener en cuenta lo siguiente:

1. Tanto *condicion1* como *sentencias2* son opcionales.
2. Se debe comprobar que *condicion1*, si existe, y *condicion2* son de tipo lógico. Si no es así, el mensaje que debes pasar a `errores.semantico` es exactamente `"Error de tipos en bucle repetir."`.
3. Se deben poder utilizar bucles dentro de otros bucles, de forma anidada.
4. Se deben poder utilizar bucles dentro de funciones. En ese caso, la ejecución de una sentencia `devuelve` mientras se ejecuta el bucle debe tener el comportamiento habitual, finalizando la ejecución de la función.

El siguiente ejemplo (del que tienes una copia en el fichero `ejemplo.min` de tu *home*) ilustra algunos de estos aspectos:

```
1  globales
2    numero: entero;
3  fin
4
5  funcion es_perfecto(n : entero) : entero
6  es
7    locales
8      suma, divisor: entero;
9    fin
10 secuencia
11   suma := 1;
12   divisor := 1;
13   repetir
14     divisor:= divisor + 1;
15   salir si divisor * divisor >= n;
16     si n % divisor = 0 entonces
17       suma := suma + divisor + n / divisor;
18     si_no secuencia fin
19     fin
20   hasta suma > n;
21   si divisor * divisor = n entonces
22     suma:= suma + divisor;
23   si_no secuencia fin
24   fin
25   si suma = n entonces
26     devuelve 1;
27   si_no
28     devuelve 0;
29   fin
30 fin
31
32 secuencia
33   numero := 1;
34   repetir
35     si llama es_perfecto(numero) = 1 entonces
36       secuencia
37         escribe numero; nl;
38       fin
39     si_no secuencia fin
40     fin
41     numero := numero + 1;
42   hasta numero>500;
43 fin
```

Al ejecutar este ejemplo con tu solución, la salida debe ser la siguiente:

```
1
6
28
496
```

Diseña tú mismo las pruebas adicionales que consideres conveniente.

Importante:

- Guarda los ficheros del intérprete, una vez modificados, en el directorio `miniint` de tu *home*.
- El fichero principal debe llamarse `miniint` y ser ejecutable.
- Utiliza el *script* `verifica.sh` de tu directorio *home* para comprobar que la estructura es la que pedimos. **Se penalizarán los exámenes para los que este script dé errores.**