Trabajo práctico especial

Detalles de la tercera entrega

El proyecto deberá ser capaz de realizar, además de todas las actividades de la primera y segunda entrega, de generar archivos .class que sean ejecutables por la máquina virtual de java.

Para ello se pide que cada compilador genere bytecode reconocible por el ensamblador Jasmin , y que a traves de éste se generen archivos .class.

Los archivos .class generados deberán ser escritos en la misma carpeta que el archivo fuente, con el mismo nombre (y la nueva extensión /a/test.gluck→ /a/test.class).

La inclusión de dependencias (keyword use) deberá tomar en cuenta la posible existencia de archivos .class ya generados. De encontrar un archivo .class, cuya fecha de modificacion sea mas reciente que el archivo gluck original (o en caso de encontrar un .class sin su archivo gluck original), el compilador deberá leer directamente los símbolos del archivo .class, evitando recompilar dependencias reiterativamente. Se recomienda el uso de la libreria 'asm' para analizar archivos .class.

A modo de estandarización, los archivos .class generados deberán seguir las siguientes convenciones:

- Su package coincidira con el especificado en el keyword location
- Exportaran una clase publica con el mismo nombre del archivo .class
- Cada funcion definida en el modulo será definida como un metodo estatico de la clase
- Cada restricción será definida como un metodo estatico de la clase, cuyo nombre será constraint__<nombre de la restriccion> (por ejmplo: contraint__positive notar que son dos guiones bajos y no uno).
- Cada tipo definido será una clase publica defnida dentro de la clase que se esta creando. Esto es, debera terminar en un archivo .class propio, con el nombre <nombre de modulo>$<nombre del tipo>.class (tomar en cuento esto al detectar y leer simbolos cuando se procesa el comando use)
- El tipo básico string de gluck se deberá construir a partir de la clase java.lang.String definida en la libreria estandard de Java.
- Los tipos básicos boolean e int se mapearán al tipo int de la maquina virtual (32 bits). El tipo basico real se representará con el tipo double de la maquina virtual (64 bits).
- Al generar código, se espera que se favorezcan las instrucciones reducidas sobre su contraparte genérico, cuando sea posible (por ejemplo: iconst_0 en lugar de bipush 0, istore_2 en lugar de istore 2, goto en lugar de goto_w, etc).
- Ante una violación de una restricción se levantará un excepcion del tipo java.lang.InvalidStateException, cuyo mensaje sera el nombre de la restriccion.

Tambien se pide la creacion de un archivo .class llamado io en el paquete 'system', que exporte las siguientes funciones:

- print:function x:string { ...} // Imprime el texto en la salida estandard y pasa a la siguiente linea
- str2i:function x:string → i:int{ ... } // convierte el string en un entero
- str2r: function x:string → r:real {... } // convierte el string en un real
- strlen:function x:string → i:int { ... } // obtiene la longitud de un string
- substring: function x:string, start:int, length:int → sub:string { ... } // obtiene una parte de unstring
- open: function file:string → handle:int { ... } // abre un archivo (lectura/escritura)
- close: function handle:int { ... } // cierra un archivo abierto
- readLine: function handle:int → line:string { ... } // lee una linea de texto de un archivo
- writeLine: function handle:int, line:string { ... } // escribe una linea de texto en un archivo

La implementacion de ichas funciones será en lenguaje Java, pero se deberán poder invocar desde archivos gluck (de hecho los casos de prueba que correra la catedra harán uso de estas funciones). Se deberá proveer en el entregable el archivo java original, además del .class

El jar resultante deberá aceptar los **nuevos** modos de operación:

java -jar dcc.jar -A <archivo>

Analizará el archivo y generara un bloque de codigo ensamblador compatible con jasmin. Si hubiese errores los reportará no generará la salida. Notar que los flags de optimizacion deberán influir en la salida de acuerdo a la tarea que realizan.

Java -jar dcc.jar -C <archivo>

Compilará el archivo, generando el .class correspondiente. Emitirá en la salida estandard solo warnings y errores. De no haber ningún problema, se espera que el comando genere el / los archivos .class y termine silenciosamente.