Práctica de Procesadores de Lenguaje
Primera Parte

Fecha de entrega: **Lunes 20 de enero de 2014**

Número de grupo:
Componentes del Grupo (ordenados alfabéticamente por primer apellido):

Requisitos

El lenguaje a procesar

- Los programas constarán de una *sección de declaraciones*, seguida de una *sección de acciones*.
- La sección de declaraciones puede estar vacía. En caso de aparecer, constará de una secuencia de una o más *declaraciones*.
- Cada declaración constará de un nombre de *tipo* seguido de una *variable*, seguida de **;**.
    - Los identificadores de las variables comienzan necesariamente por una letra o por _. A continuación puede aparecer cero o más letras, dígitos o _.
    - Las mayúsculas y minúsculas se consideran indistinguibles (así, por ejemplo, `medusa`, `MEDUSA`, `mEdUsa` y `MeduSA` representan el mismo identificador)
    - En la sección de declaraciones no podrá haber variables duplicadas
    - El nombre de tipo podrá ser `int` (números enteros) y `real` (números reales). En los nombres de tipo tampoco se distingue entre mayúsculas y minúsculas (así, por ejemplo, las cadenas `int`, `InT`, `INT` representan todas ellas el nombre de tipo `int`)
- La sección de acciones constará de una secuencia de una o más *acciones*.
- Una acción es una *expresión* seguida de **;**
- Las expresiones usarán los siguientes operadores: `in`, `out`, `=`, `<`, `>`, `<=`, `>=`, `==`, `!=`, `||`, `+`, `-`(resta), `*`, `/`, `%`, `&&`, `-`(cambio de signo) y `!`
    - El operando de `in` ha de ser una variable. Esta variable ha debido ser previamente declarada.
    - El primer operando de = ha de ser una variable.
    - El segundo operando de =, así como los operandos del resto de los operadores pueden ser expresiones arbitrarias, siempre y cuando se respeten las reglas de prioridad.
- Los operadores obedecen a las siguientes reglas de prioridad y asociatividad:
    - El menor nivel de prioridad (nivel 0) es el de los operadores de *lectura* (`in`) y *escritura* (`out`)
        - Ambos son operadores unarios, prefijos, no asociativos.
    - El siguiente nivel de prioridad (nivel 1) es el *operador de asignación*: =
        - Es un operador binario, infijo, que asocia a derechas.
    - El siguiente nivel de prioridad (nivel 2) es el de los *operadores de comparación*: < (menor), > (mayor), <= (menor o igual), >= (mayor o igual), == (igual), != (distinto):
        - Todos ellos son binarios, infijos, y no asociativos.
    - El siguiente nivel de prioridad (nivel 3) es el de los *operadores aditivos*: || (o lógico), + (suma), - (resta):
        - Todos ellos son binarios, infijos, y asocian a izquierdas.
    - El siguiente nivel de prioridad (nivel 4) es el de los *operadores multiplicativos*: * (multiplicación), / (división), % (módulo), && (y lógico):
        - Todos ellos son binarios, infijos, y asocian a izquierdas.

    - El nivel de prioridad más alto (nivel 5) es el de los *operadores unarios*: - (cambio de signo), ! (negación lógica). Son operadores unarios, prefijos, y asociativos.
  - `(int)` (conversión a entero), `(real)` (conversión a real):
    - `(int)` y `(real)` son operadores unarios, prefijos, y no asociativos.
- Reglas de tipo:
  - Operadores de lectura / escritura (`in`, `out`):
    - El operando puede ser de tipo entero o real.
    - El tipo del resultado coincidirá con el tipo del operando.
  - Operadores de comparación (<, >, <=, >=, ==, !=):
    - El primer operando puede ser de tipo entero o real.
    - El segundo operando puede ser de tipo entero o real.
    - El resultado será de tipo entero.
  - Operadores aritméticos binarios (+, - binario, *, /):
    - Si ambos operandos son de tipo entero, el resultado es entero.
    - Si ambos operandos son de tipo real, el resultado es real.
    - Si uno de los operandos es de tipo entero y el otro es de tipo real, el resultado es real.
  - Operadores lógicos binarios (||, &&):
    - El tipo de ambos operandos ha de ser entero.
    - El tipo del resultado será entero.
  - Módulo (%):
    - El tipo de ambos operandos ha de ser entero.
    - El tipo del resultado será entero.
  - Negación lógica (!):
    - El tipo del operando ha de ser entero.
    - El tipo del resultado será entero.
  - Cambio de signo (- unario):
    - El tipo del operando puede ser entero o real.
    - El tipo del resultado será el del operando.
  - Asignación (=):
    - Si el tipo del primer operando es real, el del segundo operando puede ser entero o real; el resultado será real.
    - Si el tipo del primer operando es entero, el del segundo operando ha de ser necesariamente entero; el resultado será entero.
- Reglas de evaluación:
  - Regla de *alineamiento de tipos*: Si un operador binario: (1) admite operadores de distinto tipo, (2) uno de los operandos es de tipo entero, y (3) el otro es de tipo real, entonces: el valor del operando de tipo entero se convertirá a un número real equivalente antes de llevar a cabo la operación.
  - Operador de lectura: `in` *v*:
    - Si *v* es de tipo entero, leer un valor entero de la entrada estándar
    - Si *v* es de tipo real, leer un valor real de la entrada estándar
    - Almacenar el valor leído en *v*
    - El resultado de la expresión es el valor leído
  - Operador de escritura: `out` *e*:
    - Obtener el valor de *e*

    - Escribir dicho valor por la salida estándar, seguido de un salto de línea
    - El resultado de la expresión es dicho valor.
- Operadores de comparación (<, >, <=, >=, ==, !=):
    - Obtener el valor del primer operando
    - Obtener el valor del segundo operando
    - Aplicar la regla de alineamiento de tipos
    - Comparar los valores
    - Si la comparación es cierta, el resultado es 1. En otro caso, el resultado es 0
- Operadores +, - binario y *:
    - Obtener el valor del primer operando
    - Obtener el valor del segundo operando
    - Aplicar la regla de alineamiento de tipos
    - Realizar la operación
    - El resultado es el valor obtenido
- Operador /:
    - Obtener el valor del primer operando
    - Obtener el valor del segundo operando. Si es 0, terminar la ejecución con error (*división por* 0).
    - Aplicar la regla de alineamiento de tipos
    - Si el tipo de ambos operandos es entero, realizar la división entera. En otro caso, realizar la división real
    - El resultado es el valor obtenido
- Operador %:
    - Obtener el valor del primer operando
    - Obtener el valor del segundo operando. Si es menor o igual que 0, terminar la ejecución con error (*segundo operando de % no positivo*).
    - Encontrar el resto de la división entera del valor del primer operando entre el del segundo operando
    - El resultado es el valor obtenido
- Operador ||:
    - Obtener el valor del primer operando
    - Obtener el valor del segundo operando
    - Si alguno de los valores es distinto de 0, el resultado es 1. En otro caso, el resultado es 0
- Operador &&:
    - Obtener el valor del primer operando
    - Obtener el valor del segundo operando
    - Si ambos valores son distintos de 0, el resultado es 1. En otro caso, el resultado es 0
- Operador !:
    - Obtener el valor del operando.
    - Si es distinto de 0, el resultado es 0. En otro caso, el resultado es 1.
- Cambio de signo (- unario):
    - Obtener el valor del operando.
    - El resultado es dicho valor, cambiado de signo

    - Asignación (=):
        - Obtener el valor del segundo operando
        - Si el tipo del primer operando es real y el del segundo operando es entero, alinear este último al valor real equivalente.
        - Almacenar dicho valor en *v*
        - El resultado de la expresión será dicho valor
- Como expresiones básicas, que podrán ser combinadas mediante los operadores anteriores, se consideran las siguientes:
    - Literales *enteros positivos*. Secuencias de uno o más dígitos, no admitiéndose ceros a la izquierda
    - Literales *reales*. Secuencia formada por una parte entera, seguida de una de estas tres cosas: una parte decimal. La parte entera tiene la misma estructura que un literal entero positivo. La parte decimal está formada por el símbolo **.** seguido de uno o más dígitos, no admitiéndose ceros a la derecha.
    - Variables, que han debido ser convenientemente declaradas en la sección de declaraciones
- En las expresiones es posible utilizar paréntesis para alterar la forma en la que se aplican los operadores sobre los operandos
- El lenguaje admite *comentarios de línea*. Dichos comentarios comienzan con el símbolo **@** y se extienden hasta el fin de la línea

Ejemplo de programa en el lenguaje definido:

```
@ Programa de ejemplo 1
real cantidadTotal;
int euros;
real centimos;

out euros = (in euros);
out centimos = (in centimos);
out cantidadTotal = euros + centimos;
$
```

Escrito de forma más *clara*, el programa es equivalente a:

```
@ Programa de ejemplo 2
real cantidadTotal;
int euros;
real centimos;

in euros;
in centimos;
cantidadTotal = euros + centimos;

out euros;
out centimos;
out cantidadTotal;
$
```

Partes para entregar:

1. **Memoria con toda la especificación**
2. **Implementación**

- **Memoria:**
    1. Especificación del léxico del lenguaje
    2. Especificación de la sintaxis del lenguaje
    3. Estructura y construcción de la tabla de símbolos
    4. Especificación de las restricciones contextuales
    5. Especificación de la traducción
    6. Diseño del Analizador Léxico
    7. Acondicionamiento de las gramáticas de atributos
    8. Esquema de traducción orientado a las gramáticas de atributos
    9. Esquema de traducción orientado al traductor predictivo–recursivo
    10. Formato de representación del código P
    11. Notas sobre la Implementación

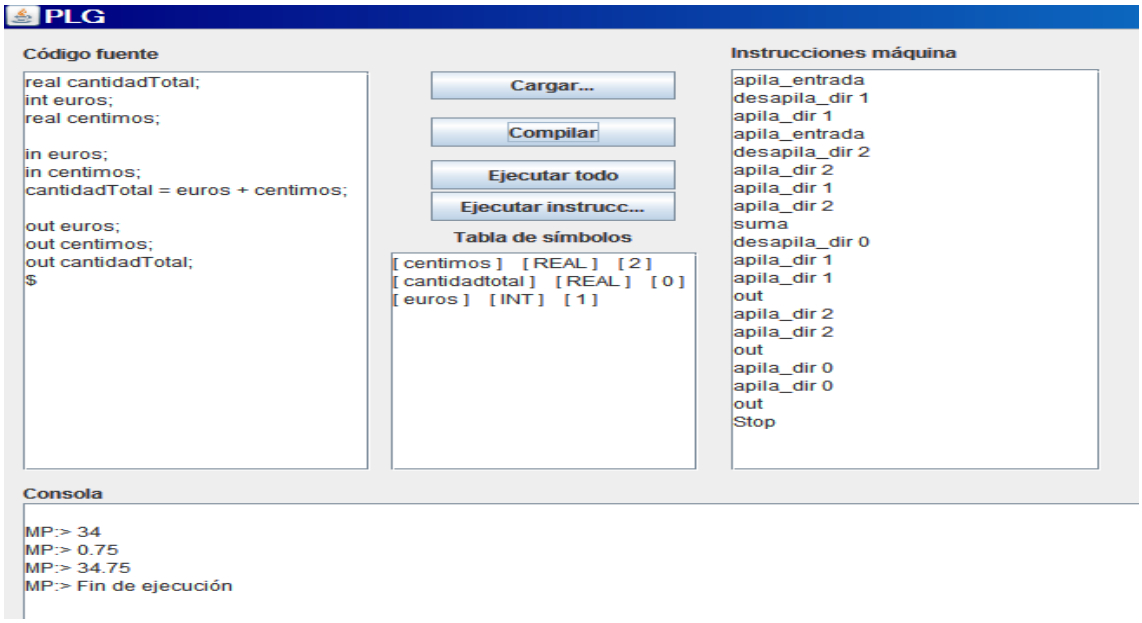
- **Código : implementación**

Debes implementar dos programas separados:

1. **Traductor** del lenguaje fuente a código de una máquina P. Este programa tomará el nombre del archivo que contiene el programa fuente y el nombre del archivo donde se va a generar el código objeto. El resultado dependerá de si el programa fuente es o no correcto:
    - Si es correcto, se generará el archivo con el código objeto.
    - Si no es correcto, se mostrará por pantalla un listado de errores y no se generará ningún archivo.
        - Cada error estará precedido por los números de fila y de columna que identifican el punto exacto en el programa fuente en el que se ha producido el error.
        - La ejecución del programa se interrumpe en el momento en que se encuentra el primer error sintáctico. El resto de los errores (léxicos y de violación de restricciones contextuales) no interrumpen la ejecución del traductor.

2. **Intérprete** trivial capaz de simular el comportamiento de la máquina P. Este programa toma como argumento el archivo donde está el código P a ejecutar y el único resultado visible será el producido por las instrucciones específicas de lectura/escritura. Opcionalmente podrá funcionar en modo modo traza, :
    - mostrará una traza por la consola con los distintos pasos de ejecución. En cada paso el intérprete debe: Mostrar el contenido de la pila de ejecución y de las celdas relevantes en la memoria de datos, esperar a que el usuario pulse una tecla para proseguir la ejecución y mostrar la instrucción a ejecutar y ejecutarla, avanzando al siguiente paso

- Se valorará la interfaz de usuario de la aplicación. Ejemplo de interfaz de usuario:



- Para la implementación del procesador puedes utilizar ANTLR, con las facilidades que ofrece para el análisis léxico, sintáctico y semántico.

Forma de entrega

- A través del campus virtual (se habilitará una herramienta de entrega para cada grupo).
- Deberá subirse un archivo comprimido (.zip o .rar) con la siguiente estructura de carpetas:
  - Carpeta *memoria*. En dicha carpeta se dejará un archivo MS Word `memoria.doc`, conteniendo la presente plantilla convenientemente rellena. En la portada deberán aparecer los nombres y apellidos de los miembros del grupo, ordenados alfabéticamente por primer apellido (aquéllos miembros que hayan renunciado a participar en la práctica no deberán aparecer en la portada de la memoria).
  - Carpeta *fuentes*. Los fuentes de los programas implementados (archivos. java y .g)
  - Carpeta *pruebas*. Los casos de prueba.

Los archivos deberán subirse antes del 20 de Enero de 2014, a las 10:00 am.

Especificación del léxico del lenguaje

Especificación formal del léxico del lenguaje utilizando definiciones regulares.

<A rellenar>

2. Especificación de la sintaxis del lenguaje

Especificación formal de los aspectos sintácticos del lenguaje utilizando una gramática incontextual. Dicha gramática debe representar de manera natural las prioridades y asociatividades de los operadores

<A rellenar>

3. Estructura y construcción de la tabla de símbolos

Deberán contemplarse tanto los aspectos necesarios para comprobar las restricciones contextuales, como los necesarios para llevar a cabo la traducción

3.1. Estructura de la tabla de símbolos

Descripción de las operaciones de la tabla de símbolos, definiendo la cabecera de dichas operaciones, así como describiendo informalmente su cometido, incluyendo el propósito de cada uno de sus parámetros

<A rellenar>

3.2. Construcción de la tabla de símbolos

Formalización de la construcción de la tabla de símbolos mediante una gramática de atributos

3.2.1 Funciones semánticas

Descripción, si procede, de las funciones semánticas adicionales utilizadas en la especificación. Para cada función debe indicarse explícitamente su cabecera, así como informalmente su cometido, incluyendo el propósito de cada uno de sus parámetros.

Esta sección puede dejarse vacía si no se van a usar funciones semánticas adicionales.

<A rellenar>

3.2.2 Atributos semánticos

Para cada categoría sintáctica relevante en este procesamiento deben enumerarse sus atributos semánticos, indicando si son heredados o sintetizados, y describiendo informalmente su propósito.

<A rellenar>

3.2.3 Gramática de atributos

Gramática de atributos que formaliza la construcción de la tabla de símbolos

<A rellenar>

4. Especificación de las restricciones contextuales

4.1. Funciones semánticas

Descripción, si procede, de las funciones semánticas adicionales utilizadas en la especificación. Para cada función debe indicarse explícitamente su cabecera, así como informalmente su cometido, incluyendo el propósito de cada uno de sus parámetros.

Esta sección puede dejarse vacía si no se van a usar funciones semánticas adicionales.

<A rellenar>

4.2. Atributos semánticos

Para cada categoría sintáctica relevante en este procesamiento deben enumerarse sus atributos semánticos, indicando si son heredados o sintetizados, y describiendo informalmente su propósito.

<A rellenar>

4.3. Gramática de atributos

Gramática de atributos que formaliza la comprobación de las restricciones contextuales.

<A rellenar>

5. Especificación de la traducción

5.1. Lenguaje objeto

5.1.1. Arquitectura de la máquina P

Explicar cómo es la arquitectura de la máquina P que se va a emplear en esta práctica (como la vista en clase)

<A rellenar>

5.1.2. Instrucciones en el lenguaje objeto

Enumeración de todo el repertorio de instrucciones del lenguaje objeto de la máquina a pila (máquina P) que se van a utilizar, así como descripción informal de su cometido.

<A rellenar>

5.2. Funciones semánticas

Descripción, si procede, de las funciones semánticas adicionales utilizadas en la especificación. Para cada función debe indicarse explícitamente su cabecera, así como informalmente su cometido, incluyendo el propósito de cada uno de sus parámetros.

Esta sección puede dejarse vacía si no se van a usar funciones semánticas adicionales.

<A rellenar>

5.3. Atributos semánticos

Para cada categoría sintáctica relevante en este procesamiento deben enumerarse sus atributos semánticos, indicando si son heredados o sintetizados, y describiendo informalmente su propósito.

<A rellenar>

5.4. Gramática de atributos

Gramática de atributos que formaliza la traducción

<A rellenar>

6. Diseño del Analizador Léxico

Diagrama de transición que caracterice el diseño del analizador léxico. La implementación del analizador léxico debe estar guiada por este diseño

<A rellenar>

7. Acondicionamiento de las gramáticas de atributos

Transformaciones realizadas sobre las gramáticas de atributos para permitir la traducción predictivo-recursiva.

Únicamente deben incluirse la transformación de las producciones que se ven afectadas. Si alguna de las gramáticas no necesitan acondicionamiento, dejar el correspondiente subapartado en blanco.

7.1. Acondicionamiento de la Gramática para la Construcción de la tabla de símbolos

<A rellenar>

7.2. Acondicionamiento de la Gramática para la Comprobación de las Restricciones Contextuales

<A rellenar>

7.3. Acondicionamiento de la Gramática para la Traducción

<A rellenar>

8. Esquema de traducción orientado a las gramáticas de atributos

Esquema de traducción en el que se muestre, mediante acciones semánticas, los lugares en los que deben evaluarse las distintas ecuaciones contempladas en todas las gramáticas de atributos: la de construcción de la tabla de símbolos, la de comprobación de las restricciones contextuales y la de especificación de la traducción.

El recorrido del árbol será el realizado por un analizador predictivo-recursivo.

<A rellenar>

9. Esquema de traducción orientado al traductor predictivo – recursivo

Esquema de traducción en el que se haga explícito los parámetros utilizados para representar los atributos, así como en los que se muestre la implementación de las ecuaciones semánticas como asignaciones a dichos parámetros.

9.1. Variables globales

Descripción y propósito de las variables globales usadas.

Si no se usan variables globales, dejar este apartado en blanco.

<A rellenar>

9.2. Nuevas operaciones y transformación de ecuaciones semánticas

En caso de introducir nuevas operaciones (por ejemplo: el procedimiento "emite"), deben describirse aquí.

Debe describirse, así mismo, qué ecuaciones semánticas se ven transformadas y cómo (por ejemplo: la generación de código se implementa emitiendo la última instrucción).

<A rellenar>

9.3. Esquema de traducción

<A rellenar>

10. Formato de representación del código P

Deberá describirse el formato de archivo aceptado por el intérprete de la máquina P, llamado código a pila (código P). Se valorará la eficiencia de dicho formato (por ejemplo: uso de *bytecode* binario en lugar de texto).

<A rellenar>

11 Notas sobre la Implementación

Descripción de la implementación realizada.

11.1. Descripción de archivos

Enumeración de los archivos con el código fuente de la implementación, y descripción de lo que contiene cada archivo.

El código en sí deberá estar adecuadamente comentado, pero NO se incluirá en la memoria de la práctica. Aquí sólo debe describirse someramente qué contiene cada archivo.

<A rellenar>