UTN FRBA – SSL – Examen Final – 2019-02-25

Apellido, Nombre:		Legajo:		Nota:	

- Resuelva el examen en tinta y en esta hoja; no se aceptan hojas adicionales.
- Para los ítems de *una mejor respuesta*, marcados con una círculo (❍), tilde (✓) sólo una opción, la mejor.
- Para los ítems de *respuestas múltiple*, marcados con un caja (❑), tilde (✓) todas las respuestas correctas.
- Durante el examen no se responde consultas; si lo necesita, escriba hipótesis de trabajo, las cuales también se evalúan.

1. Analice la siguiente declaración: `int var;`
   a. *(2 puntos)* Nivel léxico — ¿Cúantas invocaciones a `getchar` son necesarias para realizar su análisis léxico? Asuma que todas las invocaciones son exitosas y que la última retorna `EOF`. Justifique.
   b. *(2 puntos)* Nivel semántico — ¿Es semánticamente correcta? Justifique.
   c. *(2 puntos)* Niveles léxico y sintáctico — Si se eliminan los espacios, ¿sigue siendo un constructo sintáctico válido? Justifique.
   d. *(Punto extra)* Nivel sintáctico — Si el lexema `int` se reemplaza por `T` ¿sigue siendo una declaración? Justifique.
2. Dada la expresión `a.b[42].c`:
   a. *(1 punto)* Enumere los operadores:
   b. *(2 puntos)* Escriba las declaraciones para que sea una expresión `int`.
   c. *(Punto extra)* Resuelva el anterior ítem utilizando una sola declaración, que no use `typedef`, y haga que el valor de expresión sea cero.
3. *(1 punto)* Tilde la afirmación **falsa** con respecto a BNF:
   ❍ Es un meta lenguaje.
   ❍ Posee metasímbolos.
   ❍ Es útil para definir LP.
   ❍ Puede describir cualquier LF tipo 2.
   ❍ Puede describir el LF *identificadores de C*.
   ❍ Puede describir el LF *expresiones de C semánticamente correctas*.

1. Una Resolución

1. a. Once.
   Secuencia de lectura: (`'i'`, `'n'`, `'t'`, `' '`, `' '`, `'v'`, `'a'`, `'r'`, `';'`, `';'`, `EOF`.)
   Todos los caracteres se leen una vez, salvo el espacio y el punto-y-coma que se leen dos veces. La primera vez marca el fin del lexema anterior y es seguido por una invocación a `ungetc` para *deshacer* la lectura. Para el caso del espacio, la segunda lectura avanza en el flujo, pero para el punto-y-coma arma un lexema, justamente, el lexema formado por solo punto-y-coma. Luego del caracter punto-y-coma ya no hay más datos, eso lo sabemos por una última invocación a `getchar` que retorna `EOF`.

   b. Depende del contexto. Si en el mismo alcance y espacio de nombres existe una declaración de `var` incompatible, es semánticamente incorrecta, si no, es correcta. Por ejemplo:

      i.
      ```
      int var; // OK, única
      // resto de la unidad de traducción.
      ```

      ii.
      ```
      struct var{double var;};
      int var; // OK, diferentes espacios de nombres.
      // resto de la unidad de traducción.
      ```

      iii.
      ```
      double var;
      int var; // Error, redefinción con tipo diferente.
      // resto de la unidad de traducción.
      ```

   c. Sí, pero deja de ser una *declaración*, `intvar;` es una *sentencia-expresión*, donde la *expresión* es una *expresión-primaria* formada por un *identificador*.

   d. No podemos analizar la sección `T var;` completamente independiente del contexto, necesitamos tener acceso a la información semántica en la tabla de símbolos. Si `T` es un *nombre-typedef*, entonces es sintácticamente correcta, si no, no.

2. a. `.`, `[]`, y `..`

   b.
   ```
   struct X { int c; }; // estrcutura llamada X, que tiene un int llamado c.
   struct Y { struct X b[43]; }; // estructura llamada Y, que tiene un array
    de 43 estructuras X, llamado b.
   struct Y a; // variable llamada a, de tipo estructura Y.
   ```

   c.
   ```
   struct{struct{int c;}b[42+1];}a={0};
   ```

3.
   - ❍
   - ❍
   - ❍
   - ❍
   - ❍
   - ✓

v1.0.0, 2019-03-02