Trinity

Analizador lexicográfico

16 de septiembre de 2014

Índice general

---

1 Introducción

Esta sección es informativa.

1. Este documento especifica los requerimientos para la primera entrega del intérprete para el lenguaje de programación *Trinity*.

2. Todas las versiones publicadas de este documento, incluyendo la más reciente y vigente, se encontrarán siempre disponibles en la página de la práctica, en http://ldc.usb.ve/~05-38235/cursos/CI3725/2014SD/.

3. La especificación del lenguaje de programación *Trinity* **no** es parte de este documento — la definición del lenguaje se encuentra en un documento separado. Solo una parte de lo especificado en *ese* documento debe ser implantada para la entrega especificada en *este* documento. La especificación del lenguaje está disponible en la página del curso.

2 Requerimientos generales

Esta sección es normativa.

1. En este curso, Ud. debe escribir un intérprete para el lenguaje de programación *Trinity* según lo especificado en el documento de especificación del lenguaje, este documento, y los documentos de requerimientos para las demás entregas del proyecto.

2. El proyecto se desarrollará en cuatro partes, a saber:

    1. El analizador lexicográfico,
    2. el analizador sintáctico,
    3. la tabla de símbolos y el análisis estático, y
    4. la ejecución de programas en forma interpretada.

3. Debe desarrollar su proyecto usando **uno** de los tres conjuntos de herramientas seleccionados para el curso:

    1. El lenguaje de programación *Haskell*, junto con el generador de analizadores lexicográficos *Alex* y el generador de analizadores sintácticos *Happy*.
    2. El lenguaje de programación *Ruby*, junto con el generador de analizadores sintácticos *Racc*.
    3. El lenguaje de programación *Python*, junto con el generador de analizadores sintácticos *PLY*.

    Una vez seleccione un conjunto de herramientas para realizar su proyecto, deberá realizar las entregas de los demás en el mismo. No es factible y no se aceptará que se combinen herramientas ni que cambien sus herramientas una vez realizada la primera entrega.

4. Las entregas deben realizarse vía correo electrónico a los encargados de la práctica en un archivo adjunto de nombre `XX.tar.gz`, donde `XX` es su número de grupo. El archivo deberá contener un directorio con todo lo necesario para compilar su proyecto y ejecutarlo.

5. No incluya en su entrega archivos producidos por herramientas generadoras de código ni compiladores que Ud. utilice para su proyecto, sino los archivos de fuente que Ud. escribió para suministrarle a esas herramientas.

6. Su proyecto debe incluir un `Makefile` que permita generar todo lo necesario para su ejecución al ejecutar el comando `make` en el directorio contenido en su entrega. Si Ud. elige realizar su proyecto en *Haskell*, se acepta y se recomienda que utilice *Cabal* en vez de Make para especificar lo necesario para que su proyecto compile.

7. Es **deseable** que su proyecto funcione sin modificaciones en las computadoras del Laboratorio Docente de Computación. Si su proyecto requiere alguna dependencia adicional por alguna situación excepcional, debe indicar las razones claramente al realizar su entrega y justificarlo en la documentación de su proyecto.

8. Una vez efectuada la generación de código y compilación correspondiente a su proyecto, éste debe poder ejecutarse con la orden `./trinity archivo.ty` desde el directorio raíz de su proyecto, donde `archivo.ty` será la ruta (relativa o absoluta) de un archivo de entrada para su proyecto. En caso de que utilice *Cabal* para compilar su proyecto usando las herramientas de *Haskell*, es esperado y aceptable que el ejecutable se genere en las ubicaciones usuales de esa herramienta, y no es necesario que pueda ejecutarse con ese comando exacto.

9. Es un hecho común y natural del diseño y la implantación de lenguajes de programación que el diseño se modifique durante el desarrollo para adaptarse al contexto en el cual el lenguaje se utilizará, el cual es, en este caso, el curso. En función de esto, tanto el documento de especificación del lenguaje como la especficación de cada etapa y entrega estarán sujetos a cambios y aclaratorias. Los encargados de la práctica le notificarán de tales cambios y aclaratorias y actualizarán los documentos relevantes siempre que ocurran tales modificaciones a la especificación original.

10. Se acostumbra a hacer públicas las respuestas a consultas hechas por el equipo de la práctica referentes a la especificación del lenguaje y a las condiciones de entrega de cada etapa, lo cual beneficia al resto de los alumnos. Si Ud. hace una consulta y no desea que se publique, notifíquelo al hacer su consulta. **Todas las consultas que se publiquen serán anónimas.**

11. Deberá enviar sus entregas antes de la medianoche al final del día especificado para las entregas de cada etapa del proyecto, en hora legal de Venezuela. Las entregas se realizarán siempre los viernes de la semana indicada para cada etapa. Cualquier modificación al cronograma de entregas será notificada por los encargados de la práctica en la página del curso y en las horas de práctica.

12. Las entregas de cada etapa se realizarán en las siguientes semanas y tendrán la siguiente ponderación correspondiente en la evaluación del curso:

Etapa	Semana	Valor (puntos)	Asignación
1	3	5	Analizador lexicográfico
2	5	5	Analizador sintáctico
3	8	9	Tabla de símbolos y verificaciones estáticas
4	11	11	Ejecución con verificaciones dinámicas

3 Requerimientos específicos

Esta sección es normativa.

1. Para esta etapa del proyecto, Ud. debe escribir un analizador lexicográfico utilizando las herramientas que haya seleccionado.

2. En caso de haber seleccionado las herramientas de *Haskell*, deberá utilizar el generador de analizadores lexicográficos *Alex*. En caso de haber seleccionado otras herramientas, deberá programar un ciclo que verifique la porción inicial del código *Trinity* a analizar contra expresiones regulares correspondientes a cada tipo de lexema, consuma la porción reconocida, y genere el lexema correspondiente.

3. En caso de haber seleccionado las herramientas de *Haskell*, deberá definir un tipo de datos con un constructor por cada clase de lexema del lenguaje — su analizador lexicográfico producirá un valor de este tipo por cada lexema reconocido. En caso de haber seleccionado las herramientas de *Ruby* o *Python*, deberá definir una clase abstracta para los lexemas, con subclases concretas para cada tipo de lexema del lenguaje — su analizador lexicográfico producirá instancias de estas clases concretas por cada lexema reconocido.

4. En caso de haber seleccionado las herramientas de *Haskell*, deberá definir las reglas para reconocer cada tipo de lexema del lenguaje en el archivo de entrada para *Alex*, que debería llamarse *Lexer.x*. En caso de haber seleccionado las herramientas de *Ruby* o *Python*, deberá usar las expresiones regulares provistas por el lenguaje para reconocer cada tipo de lexema del lenguaje, y queda a su criterio cómo organizar su implantación — en el caso de Python, utilice el módulo `re`.

5. Al ejecutarse, su analizador lexicográfico deberá abrir el archivo de entrada e intentar consumir por completo la entrada reconociendo lexemas del programa a analizar, que se encontrará en un archivo en la ruta especificada en el primer y único argumento de línea de comando.

6. Si en algún momento del análisis su programa no es capaz de reconocer lexema alguno del lenguaje, deberá reportar ese error lexicográfico con el número de línea y columna donde ocurrió, ignorar un caracter, y continuar el reconocimiento. Por lo tanto, debe reportar *todos* los errores lexicográficos — no solo el primero.

7. Al final de la ejecución, su programa deberá terminar con un estado de salida distinto de cero si se encontraron errores, y con el estado de salida cero si no se encontraron errores.

8. Los números de línea y de columna se cuentan desde 1.

9. Cada lexema o error debe representarse en la salida estándar con una línea donde reporte la ubicación en el archivo de entrada donde comienza el lexema reconocido o el error encontrado. El formato exacto de salida para representar cada lexema o error queda a su criterio.

10. Para esta etapa del proyecto, **no** se utilizarán las herramientas generadoras de analizadores sintácticos — ni *Happy*, ni *Racc*, ni *PLY*. **No** debe hacer análisis sintáctico —únicamente lexicográfico— y no debe detectar errores en la estructura gramatical de los programas, ni errores semánticos o de contexto.