Departamento de Cs. e Ingeniería de la Computación
Universidad Nacional del Sur

Compiladores e Intérpretes

Proyecto Nº 1

Enunciado general del Proyecto - Compilador de MiniJava

Segundo Cuatrimestre de 2014

Entre los requisitos de cursado de la materia se encuentra la implementación de un compilador para un lenguaje que es una simplificación del lenguaje Java, al cual llamaremos MiniJava. En este lenguaje no se considerarán elementos avanzandos de Java tales como modificadores de visibilidad, genericidad, interfaces, métodos abstractos, excepciones, hilos, y sincronización, entre otros. Por otra parte, MiniJava impodrá algunas restricciones sintácticas adicionales al lenguaje, como por ejemplo el uso de palabras reservadas especiales para declarar variables (locales y de instancia) y métodos de instancia, o el uso de la asignación como una sentencia en lugar de ser parte de una expresión.

A continuación se brinda una descripción de los aspectos más importantes MiniJava. Es importante tener en cuenta que esta descripción no es exhaustiva. En clase se presentarán los requisitos que deben tenerse en cuenta para la realización del compilador y las etapas de desarrollo del mismo.

1. Especificación de MiniJava

El lenguaje para el que se construirá el compilador cuenta con los siguientes elementos:

1. Declaración de entidades de tipos primitivos (`int`, `boolean`, `String` y `char`) y referencia. Se cuenta además con el tipo especial `void`.
2. Declaración de clases concretas y herencia entre las mismas.
3. Declaración de constructores, métodos y atributos de instancia.
4. Declaración de métodos de clase.
5. Declaración de variables locales en métodos y constructores.
6. Las siguientes sentencias y expresiones:
   - Asignaciones.
   - Invocación de métodos.
   - Creación de instancias de clase.
   - Sentencias compuestas.
   - Sentencias condicionales (`if (expr) sent` o `if (expr) sent else sent`)
   - Sentencias `while`.
   - Sentencias `for`.
   - Sentencias de retorno de métodos.
   - Expresiones aritméticas con los operadores: +, -, *, / y %.
   - Expresiones booleanas con los operadores: && (and), || (or) y ! (not), y aquellas formadas utilizando los operadores relacionales: >, <, == (=), >= (≥), <= (≤) y != (≠).

7. El siguiente conjunto de clases predefinidas:

   a) **Object**: La superclase de todas las clases de MiniJava (al estilo de la clase `java.lang.Object` de Java). En MiniJava, la clase Object no posee métodos ni atributos.

   b) **System**: Contiene métodos útiles para realizar entrada/salida (al estilo de la clase `java.lang.System` de Java). A diferencia de `java.lang.System`, esta clase sólo brinda acceso a los streams de entrada (System.in) y salida (System.out) pero de manera oculta, proveyendo directamente los siguientes métodos:

   - `static int read()`: lee el próximo byte del stream de entrada estándar (originalmente en la clase `java.io.InputStream`).
   - `static void print(boolean b)`: imprime un `boolean` por salida estándar (originalmente en la clase `java.io.PrintStream`).
   - `static void print(char c)`: imprime un `char` por salida estándar (originalmente en la clase `java.io.PrintStream`).
   - `static void print(int i)`: imprime un `int` por salida estándar (originalmente en la clase `java.io.PrintStream`).
   - `static void print(String s)`: imprime un `String` por salida estándar (originalmente en la clase `java.io.PrintStream`).
   - `static void println()`: imprime un separador de línea por salida estándar finalizando la línea actual (originalmente en la clase `java.io.PrintStream`).
   - `static void println(boolean b)`: imprime un `boolean` por salida estándar y finaliza la línea actual (originalmente en la clase `java.io.PrintStream`).
   - `static void println(char c)`: imprime un `char` por salida estándar y finaliza la línea actual (originalmente en la clase `java.io.PrintStream`).
   - `static void println(int i)`: imprime un `int` por salida estándar y finaliza la línea actual (originalmente en la clase `java.io.PrintStream`).
   - `static void println(String s)`: imprime un `String` por salida estándar y finaliza la línea actual (originalmente en la clase `java.io.PrintStream`).

2. Consideraciones sobre el lenguaje

Es importante tener presente las siguientes restricciones del lenguaje MiniJava con respecto al lenguaje Java en el que se basa:

1. No se cuenta con declaraciones `package` ni con cláusulas `import`. Todas las declaraciones de clases se realizan en un único archivo fuente y todas las clases coexisten en un mismo espacio de nombres.

2. No se cuenta con la posibilidad de declarar clases abstractas ni interfaces.

3. No se cuenta con la posibilidad de utilizar modificadores de visibilidad para clases, métodos y variables de instancia.

4. Los métodos, las clases y las variables de instancia tendrán visibilidad pública (la semántica del modificador `public` en Java).

5. La declaración de métodos de instacia estará precedida por la palabra reservada **dynamic**, mientras que la declaración de métodos de clase (al igual que en Java) estará precedida por la palabra reservada **static**.

6. Cada declaración de variables de instancia estará precedida por la palabra reservada **varinst** mientras que la declaración de variables locales por la palabra reservada **varlocal**.

7. La asignación en MiniJava es una sentencia, a diferencia de en Java donde es un operador que se puede utilizar como parte de una expresión.

8. Las sentecias simples[1] deben ser parentizadas.

9. No se permite declarar excepciones definidas por el usuario ni, en general, manjear excepciones a nivel del lenguaje.

10. Además de lo indicado anteriormente, no se permite el uso de los modificadores `transcient` ni `volatile` para campos, ni de los modificadores `synchronized` ni `native` para métodos.

11. El método `main` en programas MiniJava tiene un encabezado sin parámetros, es decir, respeta la forma:

```
void static main()
```

MiniJava sólo provee las características indicadas en la sección precedente. Características no indicadas allí, tales como por ej., genericidad, anotaciones, `break`, `continue`, tipos primitivos de punto flotante, etc., no están soportadas por el lenguaje.

3. Etapas del compilador

0. Especificación del lenguaje: En esta etapa se define la gramática del lenguaje MiniJava en notación **BNF** extendida. La cátedra proveerá una versión parcial de esta gramática, la cual deberá ser extendida para considerar las reglas de precedencia/asociatividad en expresiones. Esta extensión se entregará como parte de la Etapa 1.

1. Análisis Léxico:

   a) Identifique el alfabeto de entrada.

   b) Identifique los componentes léxicos o tokens.

   c) Describa los patrones de los tokens como expresiones regulares.

   d) Implemente un analizador léxico utilizando alguna de las técnicas vistas en clase.

   En esta etapa deberá entregar un analizador léxico que reciba como entrada un programa en la sintaxis de MiniJava y retorne como salida la lista de tokens del mismo, especificando para cada token su lexema y el número de línea en la cual se encuentra. Este programa deberá ser capaz de detectar y reportar en forma adecuada todos los errores que puedan identificarse en la etapa de análisis léxico.

2. Análisis Sintáctico:

   a) Modifique la gramática de la Etapa 1 de manera tal que resulte apropiada para la construcción de un analizador sintáctico descendente predictivo y recursivo.

   b) ¿La gramática del inciso $2a$ es de tipo $LL(1)$? En caso negativo, especifique una manera de resolver los conflictos de acuerdo con la interpretación estándar de Java.

   c) Implemente un analizador sintáctico descendente predictivo y recursivo que respete lo especificado en el inciso $2b$, a partir de la gramática obtenida en el inciso $2a$.

[1] Una setencia simple es simplemente una expresión, es decir, una expresión que no se usa como parte derecha de una asignación o en un constructor como *if*, *while*, *for*.

En esta etapa deberá entregar un analizador sintáctico que reciba un programa MiniJava y retorne un mensaje adecuado que indique si el código recibido es o no es sintáctimente correcto. En caso de existir errores, el analizador deberá reportar en forma adecuada el primer error que encuentre (ya sea léxico o sintáctico) y luego interrumpir la ejecución.

Es importante que el informe técnico entregado en esta etapa refleje con precisión los pasos que se realizaron para obtener la gramática definitiva.

3. Análisis Semántico: A partir de la gramática de MiniJava modificada en la etapa anterior, se especificará un esquema de traducción que permita crear las estructuras para realizar los chequeos semánticos y de tipos de un programa MiniJava. Se debrá definir cómo es la forma de estas estructuras y cómo son utilizadas para realizar dos distintos controles semánticos. Entre estas estructuras de desarrollarán la tabla de símbolos y los árboles sintácticos abstractos.

   En esta etapa se entregará un analizador semántico que extiende las funcionalidades del analizador sintáctico desarrollado en la etapa anterior, retornando un mensaje adecuado que indique si el código recibido es o no semánticamente correcto. Al igual que en la implementación de la etapa anterior, en caso de existir errores el analizador deberá reportar el primer error encontrado (ya sea léxico, sintáctico o semántico) e interrumpir la ejecucion.

   En el informe técnico entregado en esta etapa deberá especificarse el esquema de traducción y el diseño de las estructuras utilizadas para el desarrolo del analizador semántico.

4. Compilador Completo: A partir del árbol sintáctico abstracto desarrolado en la etapa anterior, deberá especificarse la generación de código intermedio para un programa MiniJava.

   En esta etapa se entregará el compilador completo de MiniJava, el cual extiende los desarrollos de las etapas anteriores, añadiendo la generación de código intermedio. De esta manera, en caso de recibir un programa MiniJava correcto, el compilador generará un archivo de salida con el código intermedio asociado al programa. En caso de que el programa contenga errores léxicos, sintácticos o semánticos, se reportará el primero encontrado y no se generará archivo de salida.

   En el informe técnico de esta etapa deberá mostrarse cómo se modificaron las estructuras y controles del árbol sintáctico abstracto para reflejar la generación de código.

5. Documentación Completa: Esta última entrega del proyecto consiste en la presentación de los dos manuales necesarios en un software: el manual del usuario y el del programador. Si los informes correspondientes a cada etapa del compilador son desarrollados a consciencia, el esfuerzo de realizar el manual del programador es considerablemente menor.

4. Consideraciones sobre el proyecto

Para realizar el trabajo se formarán comisiones de dos integrantes exclusivamente. Cada comisión tendrá un tutor asociado, el cual será un ayudante o el asistente. La implementación se llevará a cabo de manera gradual y se deberán respetar los plazos de entrega de las distintas etapas del compilador de acuerdo a lo establecido en el cronograma de la materia. Toda entrega, ya sea parcial o final, deberá estar acompañada por la documentación **impresa** que corresponda. Esta documentación deberá entregarse en la fecha indicada en el cronograma, en el horario de clase. Además, en cada entrega, se deberá incluir un directorio "*pruebas*" donde se encuentren los programas que fueron utilizados por la comisión para testear el funcionamiento de la porción del compilador entregada, especificando en el informe qué técnicas de testing fueron empleadas.

Las entregas de la implementación, ya sea parcial o final, deberán realizarse personalmente en el horario de clase mediante CD o DVD (no pendrives). Todo CD/DVD deberá estar debidamente identificado. Es responsabilidad de los alumnos que los archivos posean el formato adecuado y no estén corruptos. Es de vital importancia que las entregas de implementación contengan todos los archivos

fuente y compilados. De ser necesario, se deberá agregar a la documentación una sección en la que se detalle cualquier tipo de problema o limitación relacionada con la entrega.

4.1. Lenguaje de Implementación

La implementación del proyecto deberá llevarse a cabo en Java. Se requierá que el proyecto se pueda ejecutar utilizando el *Java Runtime Environment*. Los trabajos que no respeten este formato no serán evaluados por la cátedra, y la entrega se considerará desaprobada.

4.2. Entregas fuera de término

El día de entrega señalado en el cronograma representa la fecha límite de entrega y, por lo tanto, debe ser respetado. Toda entrega fuera de término será considerada desaprobada, debiendo re-entregarse. Para más información consultar el ítem *Condiciones de cursado* en el cronograma de la materia.

4.3. Nota de las entregas

Cada entrega será calificada con *Aprobado* o *Desaprobado*. Las entregas parciales serán aprobadas si cumplen los requisitos mínimos considerados por la cátedra. Aún en este caso, el testeo completo de las mismas sigue siendo responsabilidad de la comisión. Si la entrega estuviese desaprobada, esta deberá ser re-entregada en el plazo establecido en el ítem *Condiciones de cursado* del cronograma de la materia. Aquellos alumnos que desaprueben el proyecto (al desaprobar una re-entrega) perderán el cursado de la materia.