Universidad Nacional de Córdoba
Facultad de Astronomía, Matemática y Física

Compiladores

Descripción del Proyecto
2015

En este documento se presenta una descripción general del proyecto, los requisitos y condiciones establecidas para la realización y aprobación del mismo. El objetivo es establecer un panorama general que permita apreciar el tipo de proyecto, actividades y fechas propuestas. Los detalles técnicos del proyecto serán introducidos en clase y/o presentados en otros documentos, por lo cual, no es necesario que conozca todos los términos técnicos de este documento.

El proyecto consiste en el diseño e implementación de un compilador para un lenguaje de programación simple, denominado `COMPI`. El trabajo se abordará de una manera incremental, y se ha dividido en distintas etapas, para las cuales se han establecido plazos de entrega. En cada etapa se añade a la anterior una nueva fase del compilador. Se espera que todos los grupos completen todas las fases en exitosamente.

Los Proyectos están propuestos para ser realizados en grupos de un tamaño máximo de 3 personas. En la valoración de estos Proyectos no se tendrá en cuenta el número de alumnos que componen el grupo, las dificultades de coordinación surgidas dentro del grupo, etc.

1 Cronograma de Actividades

A definir.

2 Etapas del Proyecto

Descripciones de cada una de las 6 etapas en el orden en se realizarán para implementar el compilador.

2.1 Análisis Léxico y Sintáctico

El Analizador Léxico toma como entrada un archivo con código fuente `COMPI` y retorna *tokens*. Un *token* representa a una clase de símbolos del lenguaje. Por ejemplo, operadores (`*`,`+`,`<`), delimitadores (`(`,{), palabras reservadas (`while`, `else`), literales (`342`, `3.5`) o identificadores (`var1`, `cant`). Símbolos que no son tokens son descartados, por ejemplo, espacios, tabulaciones, nuevas líneas y comentarios. Aquellos símbolos que no son permitidos en el lenguaje deben ser reportados, por ejemplo, `$`, `#`.

El Analizador Sintáctico, toma como entrada la secuencia de tokens y verifica que esta secuencia sea una secuencia válida, es decir, que cumpla con la especificación sintáctica del lenguaje. La verificación controla que, por ejemplo, los paréntesis y llaves esten balanceados, la presencia de operadores, etc. Verificaciones de tipado, nombres de variables y funciones no son realizadas en esta etapa. La salida de esta etapa (para un programa correcto) es el árbol sintáctico.

La gramática (especificación sintáctica) del lenguaje `COMPI` se presenta en otro documento. Es necesario separar la especificación del Analizador Léxico de la especificación del Analizador Sintáctico. Las herramientas ha usar serán seleccionadas por cada grupo.

2.2 Análisis Semántico

Esta etapa verifica las reglas semánticas del lenguaje, por ejemplo, compatibilidad de tipos, visibilidad y alcance de los identificadores, etc. En esta etapa es necesario implementar una tabla de símbolos para mantener la información de los símbolos (identificadores) de un programa. El análisis semántico se realizá sobre el árbol sintáctico utilizando la información almacenada en la tabla de símbolos. Las actividades de esta etapa se realizan sobre el árbol sintáctico abstracto (AST) generado durante el parsing.

En esta etapa se concluye con la construcción del *front-end* del compilador.

2.3 Análisis de Código y Optimizaciones

En esta etapa se realizaran análisis de flujo de datos y se aplicarán sus resultados a optimizaciones del código. Primero se debe construir el grafo de control de flujo (CFG) y luego implementar los análisis para generar la informacin necesaria para aplicar las optimizaciones.

2.4 Generador Código Intermedio

Esta etapa del compilador retorna una representación intermedia (IR) de bajo nivel del código. A partir de esta representación intermedia se generará el código objeto. En esta etapa se utilizará como código intermedio *Código de Tres Direcciones*.

2.5 Intérprete

En esta etapa se diseña e implementa un intérprete del código intermedio. Es decir, el intérprete toma como entrada código intermedio y retorna el resultado de interpretar (evaluar, ejecutar) dicho código.

2.6 Generador Código Objeto

En esta etapa se genera código assembly `x86-64` (sin optimizaciones) a partir del código intermedio. Dado el tiempo asignado a esta etapa, se recomemienda concentrarse en la corrección del código generado y no en la eficiencia o elegancia del código assembly generado.

Esta etapa se divide en dos entregas, primero la generación de código para operaciones con tipos enteros y lógicos y para las operaciones de control de flujo.

La segunda entrega corresponde a la generación de código assembly para las operaciones con tipos reales (opcional).

En esta etapa es muy importante el proceso de testing que se realice para detectar posibles errores y/o funcionalidades no implementadas.

2.7 Interface de la línea de Comandos

El compilador deberá tener la siguiente interface para la línea de comando.

```
> compi [opcion] nombreArchivo.compi
```

Los argumentos de la línea de comandos permitidos son los indicados en en la Tabla 1. El compilador toma como entrada un archivo de texto (el código fuente del programa a compilar), el nombre del archivo no puede empezar con '`-`' y la extensión del archivo debe ser `.compi`.

Opción	Acción
`-o <salida>`	Renombra el archivo ejecutable a `<salida>`
`-target <etapa>`	`<etapa>` es una de `scan`, `parse`, `codinter`, `intérprete` o `assembly`. La compilación procede hasta la etapa dada.
`-opt [optimización]`	Realiza la lista de optimizaciones. `all` realiza todas las optimizaciones soportadas.
`-debug`	Imprime información de debugging. Si la opción no es dada, cuando la compilación es exitosa no debería imprimirse ninguna salida.

Table 1: Argumentos de la línea de comandos del Compilador

El comportamiento por defecto del compilador es realizar la compilación, del archivo pasado por parámetro, hasta la etapa corriente (la etapa que se este implementando) y generar un archivo de salida con extensión basada en la etapa (`.lex`, `.sint`, `.sem`, `.ci`, `.ass`, `.opt`) o el ejecutable con extensión `.out` si no se indica el `target`. El nombre del ejecutable por defecto debe ser `nombreArchivo.out`.

Por defecto, ninguna optimización es realizada. La lista de optimizaciones posibles para implementar serán establecidas con cada grupo.

La responsabilidad de realizar el testing del proyecto (elaborar y correr los casos de test) es exclusiva de cada grupo. La efectividad y exhaustividad del testing realizado en cada etapa se verá reflejado en los resultados obtenidos con los casos de test de evaluación (estos casos de test no son públicos).