Universidad Nacional de Río Cuarto
Facultad de Ciencias Exactas Físico-Químicas y Naturales
Departamento de Computación

Taller de Diseño de Software

(Cod. 3306)

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 `C-TDS`. 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

Etapa	**Inicio/Entrega**	**Fecha**
Análizador Sintáctico y Léxico (Scanner y Parser)	Inicio: Entrega:	Lunes 24 de Agosto. Miercoles 07 de Septiembre.
Análizador Semántico	Inicio: Entrega:	Miercoles 09 de Septiembre. Lunes 28 de Septiembre.
Generador Código Intermedio	Inicio: Entrega:	Lunes 28 de Septiembre. Lunes 05 de Octubre.
Intérprete	Inicio: Entrega:	Lunes 05 de Octubre. Miercoles 14 de Octubre.
Generador Código Objeto	Inicio: Entrega Enteros: Entrega Reales:	Miercoles 14 de Octubre. Miercoles 28 de Octubre. Lunes 09 de Noviembre.
Optimizador	Inicio: Entrega:	Lunes 09 de Noviembre. Miercoles 18 de Noviembre.
Entrega Final	Entrega:	Viernes 20 de Noviembre.

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 `C-TDS` 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 `C-TDS` se presenta en otro documento. Es necesario separar la especificación del Analizador Léxico de la especificación del Analizador Sintáctico. Las herramientas para realizar estas actividades son:

`JFLex` (*http://jflex.de/*) y `CUP` (*http://www2.cs.tum.edu/projects/cup/*).

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 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.*

2.4 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.

2.5 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.

2.6 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.

Calificación

La calificación es dividida en los siguientes items:

Análisis Léxico y Sintáctico	5%
Análisis Semántico	15%
Generador de Código Intermedio	5%
Interprete	10%
Generador de Código Objeto	20%
Optimizador	15%
Entrega Final	30%

Cada etapa se evaluará de la siguiente manera:

- (30%) Diseño y documentación (subjetivo). La calificación se basará en la calidad del diseño, la claridad de la documentación, el conocimiento de la teoría y alternativas de implementación.
- (70%) Implementación (objetivo). La evaluación estará determinada por los casos de test que son exitosos. El 50% de la evaluación esta dado por el resultado de ejecutar los Casos de Tests Públicos y el restante 50% por los Casos de Tests No Públicos. El conjunto de tests públicos estarán disponibles, mientras que los test no públicos no serán distribuídos. Cada etapa incluye instrucciones de cual es la salida esperada. Los cambios deben ser consultados previamente a la entrega y deben estar bien justificados.

Todos los miembros de un grupo recibirán la misma calificación en cada etapa del proyecto. Esta condición puede cambiar en caso de detectar algún problema serio en el desempeño de las actividades por parte de algún integrante de un grupo. En ese caso deberán contactarse a la brevedad con los docentes.

2.7 Entregas

Cada etapa requiere entregar el código fuente completo (todos los archivos necesarios para compilar el proyecto), un ejecutable (para correr los casos de test), los casos de test utilizados y la documentación correspondiente a la etapa.

Cada etapa se realizará en grupo. Cada grupo deberá utilizar un repositorio de control de versiones (`svn` o `git`). Cada entrega será realizada utilizando el repositorio.

2.8 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.

2.9 Testing

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).

2.10 Documentación

La documentación deberá ser incluída en los archivos fuentes. La misma debe ser clara, concisa y aportar a la claridad del código. La documentación puede ser realizada en texto plano. Usar otros formatos (latex, html) es aceptable (y quizas deseable) pero no aporta en la calificación del proyecto. Si aporta la calidad de la documentación.

La documentación debe incluir las siguientes partes:

1. Una breve descripción de como fue dividido el trabajo en la presente etapa. Esto no afecta la calificación de los integrantes del grupo pero permite detectar posibles problemas a futuro.
2. Una lista de desiciones de diseño, asumpciones, aclaraciones o extensiones realizadas en la etapa. Las especificaciones del proyecto permiten que cada grupo tenga la responsabilidad de tomar decisiones. Esta es una situación que sucede habitualmente en el desarrollo de software.
3. Una descripción del diseño y las desiciones clave que se tomaron. Si es necesario analizar distintas alternativas y justificar las alternativas seleccionadas. Todas aquellas decisiones acompañadas de argumentos convincentes son aceptadas. Todas las deficiencias de diseño o implementación detectadas deben ser discutidas y se debe proponer una alternativasde diseño y/o implementación. Incluir los cambios realizados en etapas anteriores y explicar la necesidad de los mismos.

4. Una breve descripción de los detalles de implementación interesantes, por ejemplo, algoritmos no triviales, técnicas o estructuras de datos no usuales.

5. Una lista de problemas conocidos en el proyecto. Si es posible, incluir detalles de como se detecto el problema. En los casos en los cuales el compilador falla para un determinado caso de test (o conjunto de casos de test) y no se logró solucionar el el problema, se debe incluir una descripción detallada del mismo y las actividades realizadas en pos de solucionarlo.