Trabajo Práctico Especial

1. Introducción

Los compiladores son elementos clave de la informática y los que permiten convertir prosa digital en acción. Hay cientos, miles y probablemente en cada clase de Lenguajes y Compiladores del mundo se contruya uno. No vamos a hacer menos entonces...

1.1. Objetivo

El objetivo principal es el desarrollo completo de un lenguaje y su compilador, construyendo los dos componentes principales de un compilador: el analizador léxico y el analizador sintáctico. Para ello, tienen que elaborar primero la idea del lenguaje, crear la gramática e implementar el compilador que genere un programa ejecutable para alguna plataforma. El programa tiene que estar desarrollado en C, utilizando Lex como Scanner (analizador léxico) y Yacc como Parser (analizador sintáctico).

2. Primera Parte: Gramática

La primera parte es la creación de la definición de la gramática que produce el lenguaje que el Compilador aceptará y transformará.

El mismo debe contener obligatoriamente y como mínimo:

- Tipos de datos: numérico y cadenas.
- Constantes, Delimitadores.
- Operadores aritméticos, relacionales, lógicos, de asignación.
- Bloque Condicional (if).
- Bloque Do-While (repetición hasta condición).

Ejemplo de un lenguaje posible:

```
int x;
int y;
main() {
  x = 6;
  y = 1;
  while (x>0) {
   y=y*x;
   x=x-1;
   }
  puts("El factorial de 6 es: ");
  puts(y);
  puts("\n");
  puts("El valor de x+1 es:");
  puts(x+1);
}
```

(el anterior es un ejemplo, pueden definir ustedes las palabras reservadas que deseen, y la estructura que les parezca más conveniente).

Para esta primera parte es importante que completen los siguientes puntos para la clase práctica de Lex (Guía 6).

- Proyecto Completo: código fuente, librerías adicionales, Makefile de Compilación.
- Gramática en un archivo BNF (Backus Nair Form)
- Scanner en LEX (archivo .l) que procese de entrada un programa en el lenguaje creado, e identifique los lexemas y genere como salida una palabra del lenguaje.
- Un README.md explicando cómo compilar y ejecutar el programa.

3. Segunda Parte: El Compilador

Los compiladores trabajan con 3(tres) lenguajes. Toman un lenguaje *fuente* que pertenence a una gramática bien definida, y lo transforman en un lenguaje de salida, normalmente de más bajo nivel (i.e. el lenguaje es menos estructurado y está más cerca del lenguaje máquina que un procesador puede interpretar). Esta transformación, compilación, la realizan en un tercer lenguaje que es el que se utilizó para escribir el propio código del compilador.

Esto se suele representar en diagrámas T, como se visualiza en la Figura 1. El primer diagrama T, identifica un compilador escrito en un lenguaje C que compila un programa escrito también en lenguaje C y con un lenguaje de salida M. Pero antes se requiere compilar el propio compilador en C a lenguaje M para poder ejecutarlo. Entonces necesito un segundo compilador, el segundo diagrama T, que toma el programa en lenguaje C del compilador y lo compila con un compilador en M, generando una salida en M que sí puede ejecutarse.

En la práctica, a la información de los lenguajes, se le adiciona la arquitectura para la cuál se genera el lenguaje, por ejemplo, uno podría compilar un programa escrito en lenguaje Swift, con un compilador en una Mac OSX escrito también en lenguaje Swift, y que como salida queda un programa binario ejecutable (assembler) para OS X (i.e. para el iPhone).

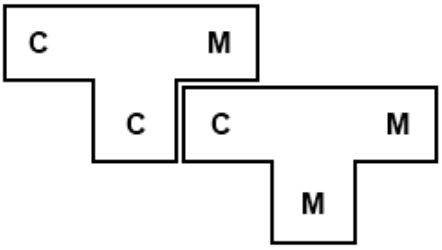

Figura 1: Diagramas T de dos compiladores donde cada uno tiene un lenguaje de entrada y uno de salida, y un lenguaje adicional que es el que se utilizó para codificar el propio compilador.

Los programas de los compiladores, suelen tener como entrada el programa a compilar por la entrada estándar (STDIN) y generan la salida, el programa en el lenguaje de salida, por STDOUT.

De esta manera, es común encontrar este tipo de código:

```
printf(" void %s(%s %s)\n", function_name, parameter_type, parameter_name);
```

Así este código estaría generando como salida, en STDOUT, código en C que luego y posteriormente hay que compilar con GCC para obtener finalmente el binario ejecutable.

En concreto entonces, y en el contexto de este TP:

- Lenguaje de Entrada del Compilador: Lo **crean** y definen ustedes.
- Lenguaje en el que está escrito el compilador: **Usan** C para Linux, usando LEX e YACC.
- Lenguaje de Salida del Compilador: Lo **eligen** ustedes. Puede ser de bajo nivel como assembler para x86, o IR de LLVM, o de más alto nivel como C o Java (que posteriormente tienen que encadenar y compilar para obtener un binario ejecutable).

3.1. Yet Another Compiler Compiler

Los compiladores de compiladores, son herramientas que permiten en base a una definición de un autómata finito y una gramática en BNF generar un parser que pueda aceptar la palabra del lenguaje, detectar posibles errores sintácticos, detectar errores semánticos y traducir el programa para generar un binario ejecutable para alguna plataforma.

La implementación deberá realizarse con Yacc, el gran compilador de compiladores, complementado lo realizado en la primera parte con Lex.

El compilador debe:

- Tener un mecanismo para indicar cuál es el punto de entrada (i.e. el *main*).
- Proveer un mecanismo de entrada de datos y de salida.
- Estar programado en C.
- Generar código ejecutable (en el lenguaje que quieran, preferentemente C, assembler o IR).

4. Material a entregar

Para la Entrega final del compilador deberán presentar:

- Proyecto Completo: código fuente, librerías adicionales, Makefile de Compilación.
- Los ejecutables del compilador. Pueden incluir herramientas y librerías adicionales que ustedes utilicen.
- Un README.md explicando cómo compilar y ejecutar el programa.
- Cinco(5) programas de ejemplo.
- Un informe o presentación DIGITAL que contenga, en este orden:
  - Carátula
  - Índice
  - Idea subyacente y objetivo del lenguaje.
  - Consideraciones realizadas (no previstas en el enunciado).
  - Descripción del desarrollo del TP.
  - Descripción de la gramática.
  - Dificultades encontradas en el desarrollo del TP.
  - Futuras extensiones, con una breve descripción de la complejidad de cada una.
  - Referencias.

Adicionales deseables que se evaluarán positivamente:

- Optimizaciones.
- Benchmarking de los tiempos de ejecución de alguno de los programas de ejemplo (e.g. test de primalidad) generados por el Compilador vs. otros lenguajes.
- Agregados extra al lenguaje como manejo de concurrencia para el procesamiento de los mensajes, matrices y vectores, tipos de datos de punto flotante, manejo de listas y colas, etc

Importante: No hay ningún inconveniente en utilizar librerías públicas, soluciones similares públicas, soluciones de foros, etc., pero es necesario aclarar, y enumerar cada una de ellas en la sección Referencias. No se aceptan bloques de código públicos implementados verbatim sin ningún tipo de análisis. Tampoco implementaciones que resuelven problemas que no están detallados (e.g. implementa un garbage collector sin explicar cómo). Tampoco hay inconveniente en que interactuen con otros grupos. *La diferencia entre plagio y ciencia es una referencia.*

5. Sugerencias

- Pueden subir todo el proyecto a GIT y dejar el informe en el README.md. El informe pienselo como el detalle del lenguaje y compilador que ustedes crearon, que incluya la información de cómo utilizarlo (el "getting started") y todas las explicaciones de lo que desarrollaron.
- Los terminales del lenguaje pueden ser más generales como IDENTIFICADOR. Utilizando LEX pueden después asociar exactamente qué conjunto de caracteres de entrada permiten formar un identificador válido del lenguaje.
- Pueden usar una tabla de símbolos donde almacenar a los objetos del programa.
- Si ejecutan

  ```
  gcc -S file.c -o file.s
  ```

  la salida será un programa file.s que contiene código en assembler que luego pueden compilar con

  ```
  gcc -c file.s -o file
  ```

- Pueden acceder al linux de pampero conectandose remotamente mediante

  ```
  ssh username@pampero.it.itba.edu.ar
  ```

- Si la salida elegida para el compilador es Java, pueden utilizar BCEL, CGLIB y ASM para generar bytecode, assembler, para Java.
- Los compiladores suelen generar en la primera pasada del parsing del programa, un AST, un Abstract Syntax Tree, que luego recorren para generar el código. El AST es una estructura de árbol donde los nodos son los terminales y no-terminales que componen el programa.

6. Grupos

- TLAtubbies
- JEFA
- Dinopicinito
- The Roxys
- ALT
- ChetosMal
- Grupo X
- Hinjenieros
- Lorant y Los Pibes

7. Fecha de entrega

Cada grupo deberá coordinar la presentación del material enviando un email a `rramele@itba.edu.ar` con el nombre del grupo, indicando la URL del Branch o Tag de Git, Bitbucket u otro SCM similar. La presentación no demorará más de 20 minutos, donde se presentará el compilador, el abordaje, los problemas encontrados y se correrán algunos ejemplos.

Importante tener en cuenta que si el proyecto no compila en Pampero, la entrega se considerará inválida.

El TP se debe entregar antes del día 21/06/2018 23:59 GMT -3.

8. Criterio de Corrección

- A. Informe: secciones, prosa, errores ortográficos, claridad en la exposición.
- B. Cumplimiento de consignas.
- C. Implementación del TP: investigación, pertinencia e impacto de la idea, creatividad, innovación, practicidad , definición y exposición de la idea subyacente, alineamiento con los temas vistos en la materia.
- D. Calidad en la presentación: armado del proyecto, documentación del código, estructura del código, scripts de automatización, test de regresión, etc.
- E. $+\alpha$: Puntos extras agregados más allá de los requerimientos del enunciado e incluso del trabajo mismo.

Cada item es puntuado del 1-10 y la nota del TP surge del promedio de todos los puntos.

9. Material de consulta

1. Compiladores, Principios, Técnicas y Herramientas,Aho, Sethi, Ullman, Addison Wesley.(El Libro del Dragón), capítulos 1,2 ,3 y 5,6,7.
2. Engineering a Compiler, Appeal, Ginsbur
3. Stevens.Advanced Programming in the UNIX Environment.1992
4. `http://peter.michaux.ca/articles/assembly-hello-world-for-os-x`
5. Free BSD Developer Handbook `http://www.freebsd.org/doc/en/books/developers-handbook/book.html#X86-SYSTEM-CALLS`
6. `http://web.cecs.pdx.edu/~apt/cs322/CompilingOO.pdf`
7. `http://www.doesnotunderstand.org/public/DSLRob2015`
8. `http://www.springer.com/gp/book/9781461446989`
9. `flex.sourceforge.net/manual/`
10. BCEL `https://commons.apache.org/proper/commons-bcel/`
11. CGLib `https://github.com/cglib/cglib`
12. ASM `http://asm.ow2.org/`
13. LLVM `https://en.wikipedia.org/wiki/LLVM`
14. BNF `http://matt.might.net/articles/grammars-bnf-ebnf/`
15. `https://github.com/faturita/YetAnotherCompilerClass`