

Trabajos de Sintaxis y Semántica de los Lenguajes

Esp. Ing. José María Sola, profesor.

Revisión 0.4.0

2017-05-16

Tabla de contenidos

1. Introducción	1
2. Requisitos Generales para las Entregas de las Resoluciones	3
2.1. Requisitos de Tiempo	3
2.2. Requisitos de Forma	3
2.2.1. Repositorio del Equipo en GitHub	3
2.2.2. Front Page del Equipo	4
2.2.3. Carpetas para cada Resolución	4
2.2.4. Header Comments (Comentarios Encabezado)	5
2.2.5. Front Page de la Resolución	5
3. @ Trabajo #0 — "Hello, World!" en C	7
3.1. Objetivos	7
3.2. Tareas	7
3.3. Productos	8
3.4. Entrega	8
4. Trabajo #1 — Conversor de Temperaturas	9
4.1. Objetivos	9
4.2. Restricciones	9
4.3. Productos	10
4.4. Entrega	10
5. ? Trabajo #2 — Contador de Palabras	11
5.1. Temas	11
5.2. Tareas	11
5.3. Productos	12
6. Trabajo #3 — Removedor de Comentarios	15
6.1. Objetivo	15
6.2. Restricciones	15
6.3. Productos	16
6.4. Entrega	17
Bibliografía	19

Introducción

El objetivo de los trabajos es afianzar los conocimientos y evaluar su comprensión. Deben resolverse en equipo, salvo que se indique que es individual, como es el caso del trabajo #0.

Los trabajos prefijados con ? son opcionales y sirven como introducción a otros trabajos más complejos; pueden resolverse en equipo o individualmente. Pueden enviar la resolución para una devolución no prioritaria.

Los trabajos prefijados con @ son comunes para todos los cursos.

Requisitos Generales para las Entregas de las Resoluciones

Cada trabajo tiene sus requisitos particulares de entrega de resoluciones, esta sección indica los requisitos generales, mientras que, cada trabajo define sus requisitos particulares.

Una resolución se considera **entregada** cuando cumple con los **requisitos de tiempo y forma** generales, acá descriptos, sumados a los particulares descriptos en el enunciado de cada trabajo.

2.1. Requisitos de Tiempo

Cada trabajo establece la fecha y hora límite de entrega, los commits realizados luego de ese instante no son tomados en cuenta para la evaluación de la resolución del trabajo.

2.2. Requisitos de Forma

Requisitos de forma del repositorio, las carpetas de las resoluciones, y los encabezados de los archivos fuente.

2.2.1. Repositorio del Equipo en GitHub

La entrega de cada resolución debe realizarse a través de *GitHub*, para ello, a cada equipo se le asigna un **repositorio privado**. Lugo, el equipo debe crear una carpeta particular para cada resolución.

2.2.2. Front Page del Equipo

En la raíz del repositorio cada equipo debe diseñar un archivo `readme.md` que actúe como front page del equipo. Debe estar escrito en notación *Markdown* y debe tener, como mínimo, la siguiente información:

- Asignatura.
- Curso.
- Año de cursada, y cuatrimestre si corresponde.
- Número de equipo.
- Nombre del equipo (opcional).
- Integrantes del equipo actualizados, ya que, durante la transcurso de la cursada el equipo puede cambiar:
 - Usuario *GitHub*.
 - Legajo.
 - Apellido.
 - Nombre.

2.2.3. Carpetas para cada Resolución

La resolución de cada trabajo debe tener su propia carpeta en el repositorio del equipo.

Además de los productos solicitados por cada trabajo, la carpeta **sí debe incluir**:

- un archivo `readme.md` que actúe como front page de la resolución.

Para facilitar el desarrollo se **recomienda incluir**:

- un archivo `.gitignore`.
- un archivo `Makefile`.
- archivos tests.

Por último, la carpeta **no debe incluir**:

- archivos ejecutables.

- archivos intermedios producto del proceso de compilación o similar.

El siguiente patrón como la carpeta se **debe nombrar**:

ÚltimosTresDígitosDelCurso-DosDelEquipo-DosDígitosNúmeroTrabajo-
NombreTrabajo

Ejemplo 2.1. Nombre de carpeta

051-02-00-Hello

2.2.4. Header Comments (Comentarios Encabezado)

Todo archivo fuente de debe comenzar con un comentario que indique el "qué", "Quiénes", "Cuándo" :

```
/* Qué: Nombre
 * Breve descripción
 * Quiénes: Autores
 * Cuando: Fecha de última modificación
 */
```

Ejemplo 2.2. Header comments

```
/* Stack.h
 * Interface for a stack of ints
 * JMS
 * 20150920
 */
```

2.2.5. Front Page de la Resolución

Cada de resolución debe contar con un archivo `readme.md`, escrito en *Markdown* que contenga, como mínimo, la siguiente información:

- Número de equipo.

- Nombre del equipo (opcional).
- Autores:
 - Usuario github.
 - Legajo.
 - Apellido.
 - Nombre.
- Número y título del TP.
- Transcripción del enunciado.
- Hipótesis de trabajo que surgen luego de leer el enunciado.

3

@ Trabajo #0 — "Hello, World!" en C

Este es un trabajo individual.

3.1. Objetivos

- Demostrar con, un programa simple, que se está en capacidad de editar, compilar, y ejecutar un programa C.
- Contar con las herramientas necesarias para la resolución de los trabajos.

3.2. Tareas

1. Solicitar inscripción al Grupo Yahoo.
2. Registrarse en GitHub.
3. Crear un repositorio público con el nombre `che1lowor1d`.
4. Seleccionar, instalar, y configurar un compilador **C11**.
5. Probar el compilador con un programa `he1lo.c` que envíe a `stdout` la línea `he1lo, wor1d!` o similar.
6. Ejecutar el programa, y capturar su salida en un archivo de texto `output.txt`.
7. Publicar `he1lo.c` y `output.txt` en GitHub.
8. Escribir y publicar el archivo `readme.md` del repositorio.
9. Enviar a UTNFRBASSL@yahoogroups.com¹ usuario GitHub.

¹ <mailto:UTNFRBASSL@yahoogroups.com>

3.3. Productos

- Repositorio público `chelloworld`.
- `readme.md`.
- `hello.c`
- `output.txt`.

3.4. Entrega

- Mar 20 (segunda clase), 13hs.

4

Trabajo #1 — Conversor de Temperaturas

Este trabajo está basado en el ejercicio 1-15 de [\[KR1988\]](#):

1-15. Reescriba el programa de conversión de temperatura de la sección 1.2 para que use una función de conversión.

4.1. Objetivos

- Realizar el primer trabajo en equipo en el repositorio privado del equipo en **GitHub**.
- Demostrar conocimiento de:
 - Funciones.
 - Archivos header (.h).
 - Interfaces e Implementación.
 - Uso de make.

4.2. Restricciones

- Utilizar `const`.
- Utilizar `for` con declaración (C99).

4.3. Productos

- Sufijo del nombre de la carpeta: Temperatura.
- FahrCel.c.
- Conversion.h.
- Conversion.c.
- Makefile.

4.4. Entrega

- Abr 17, 13hs

? Trabajo #2 — Contador de Palabras

Este trabajo está basado en el ejemplo de la sección 1.5.4 *Conteo de Palabras* de [\[KR1988\]](#):

```
"... cuenta líneas, palabras, y caracteres, con la definición
ligera que una palabra es cualquier secuencia de caracteres que
no contienen un blanco, tabulado o nueva línea."
```

5.1. Temas

- Árboles de expresión.
- Representación de máquinas de estado.
- Implementación de máquinas de estado.

5.2. Tareas

1. Árboles de Expresión

- a. Dibuje el árbol de expresión para la inicialización de los contadores: `n1 = nw = nc = 0`.
- b. Dibuje el árbol de expresión para la expresión de control del segundo `if`:
`c == ' ' || c == '\n' || c == '\t'`

2. Enum y Switch

- a. Escriba una segunda versión del programa, `wc-enum-switch.c`, que:

- i. Utilice `typedef` y `enum` en vez de `define`, de tal modo que la variable estado se pueda declarar de la siguiente manera: `State s = out;`
 - ii. Utilice `switch` en vez de `if`.
 - b. Responda: ¿Cómo implementa los estados este programa? ¿Y las transiciones?
3. Funciones Recursivas
- a. Escriba una tercera versión del programa, `wc-enum-switch.c.c`, que, en vez de una variable, utilice funciones recursivas para representar los estados.
4. Sentencias `goto` (sí, el infame *goto*)
- a. Lea la sección 3.8 *Goto and labels* de [\[KR1988\]](#)
 - b. Lea *Go To Statement Considered Harmful* de [\[DIJ1968\]](#).
 - c. Responda: ¿Tiene alguna aplicación *go to* hoy en día? ¿Algún lenguaje moderno lo utiliza?
 - d. Escriba una cuarta versión del programa, `wc-goto.c`, que, en vez de variable o funciones, utilice etiquetas para representar los estados y sentencias `goto` para las transiciones.
5. Eficiencia
- a. Construya una tabla comparativa a modo de *benchmark* que muestre el tiempo de procesamiento para cada una de las tres implementaciones, para tres archivos diferentes de tamaños diferentes, el primero en el orden de los kilobytes, el segundo en el orden de los megabytes, y el tercero en el orden de los gigabytes.

5.3. Productos

- Sufijo del nombre de la carpeta: Contador.
- `readme.md`:
 - Árboles de expresión.
 - Respuestas.
 - Tabla de mediciones.

- `wc-enum-switch.c`.
- `wc-rec.c`.
- `wc-goto.c`.
- `Makefile`.

6

Trabajo #3 — Removedor de Comentarios

Este trabajo está basado en el ejercicio 1-23 de [\[KR1988\]](#):

Escriba un programa que remueva todos los comentarios de un programa C. Los comentarios en C no se anidan. No se olvide de tratar correctamente las cadenas y los caracteres literales

6.1. Objetivo

El objetivo es diseñar una máquina de estado que remueva comentarios, implementar dos versiones, e informar cual es la más eficiente mediante un benchmark.

6.2. Restricciones

- Primero diseñar y especificar la máquina de estado y luego derivar dos implementaciones.
- Utilizar el lenguaje `dot` para dibujar los digrafos.
- Incluir comentarios de una sola línea (`//`).
- Considerar las variantes no comunes de literales carácter y de literales cadenas que son parte del estándar de C.
- Diseñar el programa para que pueda invocarse de la siguiente manera:
`RemoveComments < Test.c > NoComments.c`

- Ninguna de las implementaciones debe ser la *Implementación #1: estado como variable y transiciones con selección estructurada*.
- Indicar para cada implementación cómo se representan los estados y cómo las transiciones.
- Respetar la máquina de estado especificada, en cada implementación utilizar los mismos nombres de estado y cantidad de transiciones.
- En el caso que sea necesario, utilizar `enum`, y no `define`.
- En el caso que sea necesario, utilizar `switch`, y no `if`.
- Realizar una prueba funcional y tres pruebas de volumen.
- Construir una tabla comparativa a modo de *benchmark* que muestre el tiempo de procesamiento para cada una de las dos implementaciones, para tres archivos diferentes de tamaños diferentes, el primero en el orden de los kilobytes, el segundo en el orden de los megabytes, y el tercero en el orden de los gigabytes.
- Crear a mano el archivo de test funcional: `Test.c`.
- Construir el programa `GenerateTest.c` que genere automáticamente los tres archivos para pruebas de volumen: `Testkilo.c`, `Testmega.c`, y `Testgiga.c`.
- No incorporar al repositorio los archivos de prueba de volumen, sí el de prueba funcional.
- Diseñar el archivo `Makefile` para que construya una, otra o ambas implementaciones, y para que ejecute las pruebas.

6.3. Productos

- `/Readme.md`
 - Autómata finito para cada lenguaje.
 - Diagrama de transiciones.
 - Definición Formal.
 - Expresión regular para cada lenguaje.
 - Máquina de Estados del programa.
 - Descripción de la implementación A: `rc-a.c`.

- Descripción de la implementación B: rc-a.c.
- Benchmark.
- /rc-a.c
- /rc-b.c
- /tests/Test.c
- /tests/GenerateTest.c
- /Makefile

6.4. Entrega

- Jun 5, 13hs

Bibliografía

[KR1988] Brian W. Kernighan and Dennis Ritchie. The C Programming Language, 2nd Edition. 1988.

[DIJ1968] Edsger W. Dijkstra. Go To Statement Considered Harmful. Reprinted from Communications of the ACM, Vol. 11, No. 3, March 1968, pp. 147-148.

