Trabajos de Algoritmos y Estructura de Datos

Esp. Ing. José María Sola, profesor.

Revisión 2.0.0
2018-03-31

Tabla de contenidos

Lista de figuras

Lista de tablas

Lista de ejemplos

1

Introducción

El objetivo de los trabajos es afianzar los conocimientos y evaluar su comprensión.

En el curso se indican cuales de los trabajos acá definidos son **obligatorios** y cuales **opcionales**, cuando es la **fecha y hora límite de entrega**, como así también si se deben resolver **individualmente** o en **equipo**.

Hay trabajos opcionales que son introducción a otros trabajos más complejos, también pueden enviar la resolución para que sea evaluada.

Cada trabajo tiene un **número** y un **nombre**, y su enunciado tiene las siguientes secciones:

1. **Objetivos**: Descripción general de los objetivos y requisitos del trabajo.
2. **Temas**: Temas que aborda el trabajo.
3. **Tareas**: Plan de tareas a realizar.
4. **Restricciones**: Restricciones que deben cumplirse.
5. **Productos**: Productos que se deben entregar para la resolución del trabajo.

2

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.

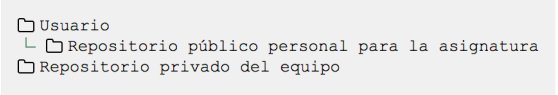
Una resolución se considera **entregada** cuando cumple con los **requisitos de tiempo y forma** generales, acá descriptos, sumados a los particulares definidos en el enunciado de cada trabajo.

La entrega de cada resolución debe realizarse a través de *GitHub*, por eso, cada estudiante tiene poseer una cuenta en esta plataforma.

2.1. Requisitos de Forma

2.1.1. Repositorios

En el curso usamos un repositorios *GitHub*. Uno público y personal y otro privado para del equipo.

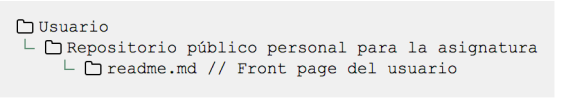


Repositorio Personal para Trabajos Individuales

Cada estudiante debe crear un repositorio público dónde publicar las resoluciones de los trabajos individuales. El nombre del repositorio debe ser el

de la asignatura. En la raíz del mismo debe publicarse un archivo `readme.md` que actúe como *front page* de la persona. El mismo debe estar escrito en notación *Markdown* y debe contener, como mínimo, la siguiente información:

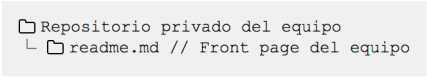
- Asignatura.
- Curso.
- Año de cursada, y cuatrimestre si corresponde.
- Legajo.
- Apellido.
- Nombre.



Repositorio de Equipo para Trabajos Grupales

A cada equipo se le asigna un **repositorio privado**. En la raíz del mismo debe publicarse un archivo `readme.md` que actúe como *front page* del equipo. El mismo debe estar escrito en notación *Markdown* y debe contener, 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 el transcurso de la cursada el equipo puede cambiar:
  - Usuario *GitHub*.
  - Legajo.
  - Apellido.
  - Nombre.



Carpetas para cada Resolución

La resolución de cada trabajo debe tener su propia carpeta, ya sea en el repositorio personal, si es un trabajo individual, o en el del equipo, si es un trabajo grupal. El nombre de la carpeta debe seguir el siguiente formato:

```
DosDígitosNúmeroTrabajo-NombreTrabajo
```

O en notación *regex*:

```
[0-9]{2}"-"[a-zA-Z]+
```

Ejemplo 2.1. Nombre de carpeta

00-Hello

Adicionalmente a los productos solicitados para la resolución de cada trabajo, la carpeta debe incluir su propio archivo `readme.md` que actúe como *front page* de la resolución El mismo debe estar escrito en notación *Markdown* y debe contener, como mínimo, la siguiente información:

- Número de equipo.
- Nombre del equipo (opcional).
- Autores de la resolución:
  - Usuario github.
  - Legajo.
  - Apellido.
  - Nombre.
- Número y título del TP.

- Transcripción del enunciado.
- Hipótesis de trabajo que surgen luego de leer el enunciado.

Opcionalmente, para facilitar el desarrollo se **recomienda incluir**:

- un archivo `.gitignore`.
- un archivo `Makefile`.[1]
- archivos tests.[1]

```
Carpeta de resolución de trabajo
├ .gitignore
├ Makefile
├ readme.md // Front page de la resolución
└ Archivos de resolución
```

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

- archivos ejecutables.
- archivos intermedios producto del proceso de compilación o similar.

Ejemplo de Estructura de Repositorios

```
usuario // Usuario GithHub
└ aed // Repositorio personal público para a la asignatura
    ├ readme.md // Front page del usuario
    ├ 00-Hello // Carperta de resolución de trabajo
    │  ├ .gitignore
    │  ├ readme.md // Front page de la resolución
    │  ├ Makefile
    │  ├ hello.cpp
    │  └ output.txt
    └ 01-Otro-trabajo

2019-051-02 // Repositorio privado del equipo
├ redme.md // Front page del equipo
├ 04-Stack // Carperta de resolución de trabajo
│  ├ .gitignore
│  ├ readme.md // Front page de la resolución
│  ├ Makefile
│  ├ StackTest.cpp
│  ├ Stack.h
│  ├ Stack.cpp
│  └ StackApp.cpp
└ 01-Otro-trabajo
```

[1] Para algunos trabajos, el archivo `Makefile` y los tests son obligatorios, de ser así, se indica en el enunciado del trabajo.

2.1.2. Lenguaje de Programación

En el curso se establece la versión del estándar del lenguaje de programación que debe utilizarse en la resolución.

2.1.3. Header Comments (Comentarios Encabezado)

Todo archivo fuente 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. 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.

3

Problemas y Soluciones

Todos los archivos `readme.md` que actúan como *Front Page* de la resolución, deben contener una descripción del *análisis del problema* y una *síntesis de la solución*.

Luego del enunciado y las hipótesis, cada `readme.md` debe contener:

- **Etapa de Análisis**. Consta del diagrama del *Modelo IPO* con:
  - Entradas: nombres y tipos de datos.
  - Proceso: nombre descriptivo.
  - Salidas: nombres y tipos de datos.
- **Etapa de Diseño**. Consta del algoritmo que define el método por el cual el proceso obtiene las salidas a partir de las entradas:
  - Léxico del Algoritmo.
  - Representación visual o textual del Algoritmo.

La resolución incluye archivos fuente que forman el programa que implementan el algoritmo definido. Es importante el programa debe seguir la definición del algoritmo, y no al revés.

4

Trabajo #0 — "Hello, World!" en C++

4.1. 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 abordar la resolución de los trabajos posteriores.

4.2. Temas

Sistema de control de versiones, lenguaje de programación C++, proceso de compilación, pruebas.

4.3. Tareas

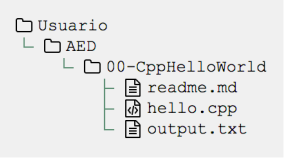
1. Solicitar inscripción al Grupo Yahoo, la aprobación demora un par de días.
2. Si no posee una cuenta *GitHub*, crearla.
3. Crear un repositorio público llamado `AED`.
4. Escribir el archivo `readme.md` que actúa como *front page* del repositorio personal.
5. Crear la carpeta `00-CppHelloWorld`.
6. Escribir el archivo `readme.md` que actúa como *front page* de la resolución.
7. Seleccionar, instalar, y configurar un compilador **C++ 17** (ó **C+ + 14** ó **C+ + 11**).
8. Probar el compilador con un programa `hello.cpp` que envíe a `cout` la línea `Hello, World!` o similar.

9. Ejecutar el programa, y capturar su salida en un archivo de texto `output.txt` .
10. Publicar en el repositorio personal `AED` la carpeta `00-CppHelloWorld` con `readme.md`, `hello.cpp`, y `output.txt` .
11. La última tarea es informar por email a UTNFRBAAED@yahoogroups.com[1] el usuario usuario GitHub.

4.4. Restricciones

- Ninguna.

4.5. Productos



[1] mailto:UTNFRBAAED@yahoogroups.com

5

Trabajo #1 — Adición

5.1. Problema

Escribir los pasos para mostrar la suma de dos números que ingresa el usuario.

5.2. Productos

- Sufijo del nombre de la carpeta: `Adición`
- `readme.md`
- `Adición.cpp`.

5.3. Entrega

- Abr 6, 13hs.

6

Trabajo #2 — Mayor de dos Números

6.1. Problema

Dado dos números informar cuál es el mayor.

6.2. Productos

- Sufijo del nombre de la carpeta: `Mayor`
- `readme.md`.
- `Mayor.cpp`.

6.3. Entrega

- Abr 13, 00hs.

7

Trabajo #3 — Repetición de Frase

7.1. Problema

Enviar una frase a la salida estándar muchas veces.

7.2. Restricciones

Realizar dos versiones del algoritmo y una implementación para cada uno:

- Salto condicional.
- Iterativa esctructurada.

7.3. Productos

- Sufijo del nombre de la carpeta: `Repetición`
- `readme.md` con los dos algoritmos.
- `Saltos.cpp`.
- `Iteración.cpp`.

7.4. Entrega

- Abr 27, 13hs.

8

Trabajo #4 — Ejemplos de Valores y Operaciones de Tipos de Datos

8.1. Tarea

Esta es una *tarea no estructurada*, la cual consiste en escribir un programa que ejemplifique el uso de los tipos de datos básicos de C++ vistos en clase: `bool`, `char`, `unsigned`, `int`, `double`, y `string`.

8.2. Productos

- Sufijo del nombre de la carpeta: `EjemploTipos`
- `EjemploTipos.cpp`.

8.3. Entrega

- May 4, 13hs.

9

? Trabajo #5 — Especificación del Tipo de Dato Fecha

9.1. Tarea

Especficar el tipo de dato "Fecha", lo cual implica especificar su conjunto de valores y su conjunto de operaciones sobre esos valores.

9.2. Productos

- `readme.md`:
  - Conjunto de Valores.
  - Conjunto de Operaciones.

10

Trabajo #9 — Browser

10.1. Necesidad

Implementar la funcionalidad *back* y *forward* común a todos los browsers.

10.2. Restricciones sobre la Interacción

- Procesamiento línea a línea.
- Una línea puede contener B para *back*, F para *forward*, el resto de las líneas de las se las considera como *URL* destino correctas.
- Por cada línea leída, se debe enviar una línea a la salida estándar: si es una URL, se envía esa URL, si es B, se envía la anterior URL, y si es F, se envía la siguiente URL.
- El procesamiento finaliza cuando no hay más líneas.

Tabla 10.1. Ejemplo de interacción

Secuencia	Entrada	Salida
1	alfa	alfa
2	beta	beta
3	gamma	gamma
4	delta	delta
5	B	gamma
6	F	delta
7	B	gamma

Secuencia	Entrada	Salida
8	epsilon	epsilon
9	B	gamma
10	F	epsilon

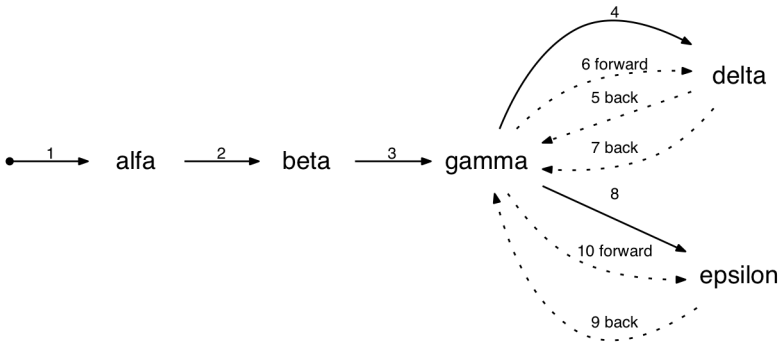


Figura 10.1. Líneas de tiempo (BTTF2) para la interacción ejemplo.

10.3. Restricciones de solución

- Obtención de líneas
  - En *C++*:

```
string línea; // guarda la línea obtenida de cin.
while(getline(cin, línea)) ... // obtiene una línea de cin y la
 guarda en línea.
```

  - En *C*:

```
#define MAX_LINE_LENGTH 1000 // cantidad máxima de caracteres en
 una línea.
char line[MAX_LINE_LENGTH+1+1]; // guarda la línea obtenida de
 stdin.
while(fgets(línea, sizeof línea, stdin)) ... // obtiene una línea
 de stdin y la guarda en línea.
```

- Diseñar las siguientes funciones:
  - `GetLínea() // retorna una línea de la entrada estándar.`

- `GetTipo(línea) // retorna un código para los diferentes tipos de líneas.`
- `AccionarSegún( GetTipo(línea) ) // realiza la acción correspondiente.`
- `Mostrar(unaUrl) // Envía unaUrl a la salida estándar.`
- `Back() // vuelve una URL atrás y la muestra.`
- `Forward() // avanza a la URL siguiente y la muestra.`
- `GuardarUrl() // realiza lo necesario para guardar una URL.`
- `GetPrevUrl() // obtiene la anterior URL.`
- `GetNextUrl() // obtiene la sieugiente URL.`

10.3.1. Mejoras

Las siguientes mejoras son ejercicios opcionales y avanzados que completan la funcionalidad.

Nuevos Comandos para el Manejo del Historial

- `Refresh`: Envía por la salida estándar la URL actual.
- `PrintHistory`: Envía por la salida estándar todas las URL visitadas en orden, primero la primera visitada y último la la última.
- `ClearHistory`: Borra el historial.
- `PrintThisTimeLine`: Envía por la salida estándar una representación textual en *dot* [DOT] de la línea temporal actual. Para el ejemplo original, si estamos en el paso N mostraría:

- `PrintAllTimeLines`: Lo mismo que `PrintThisTimeLine` pero para todas las líneas de tiempo en forma de árbol, en vez de secuencia, cuya raíz es la primera URL visitada.
- Agregar al historial la fecha y hora de cada visita. En *C++* con `<chrono>`, y en *C* con `<time.h>`.
- Al finalizar el procesamiento, generar los archivos `History.txt`, `ThisTimeLine.gv`, y `AllTimeLines.gv`.

Mejoras al Intérprete de Comandos

- Requeerir que los comandos comiencen con `.` (punto).
- Agregar a los comandos `PrintX` una opción `-f` para indicar que la salida se envía a un file, y no a la salida estándar. Los filenames por defecto son `History.txt`, `ThisTimeLine.gv`, y `AllTimeLines.gv`, respectivamente.
- Agregar a la opción `-f` de los comandos `PrintX` un argumento para indicar el nombre del file destino, para que se puedan paersonalizar los archivos destino.
- Agregar validación de las líneas, para que el programa pueda emitir mensajes del tipo `Comando inválido.`, `Opción inválida.`, `Argumento inválido.`, y `URL inválida.`. La función que implementa la validación es `GetComandooUrl(línea)` que retorna un valor de la enumeración `{NoHayMásLíneas, Back, Forward, Url, Refresh, ClearHistory, PrintHistory, PrintThisTimeLine, PrintAllTimeLines, UrlInválida, ComandoInválido};`. Esta función de validación se puede implementar de tres formas:
  - Implementar las validaciones con las tres estructuras de control de flujo de ejecución.
  - Implementar las validaciones con un autómata finito con tantos estados finales como situaciones posibles.
  - Implementar las validaciones con expresiones regulares. En *C++* utilizar `regex`, en *C* utilizar `lex`.
- Agregar *alias* a los comandos y hacer el intérprete *case-insensitive*:

Comando	**Alias**
`Back`	`B`
`Forward`	`F`
`Refresh`	`R`
`PrintHistory`	`PH`
`ClearHistory`	`CH`
`PrintThisTimeLine`	`PTL`

Comando	Alias
PrintAllTimeLines	PATL

10.4. Productos

- BrowserSimple/browse.cpp
- BrowserMásComandos/browse.cpp
- BrowserMejorIntérprete/browse.cpp
- BrowserValidadorEstructurado/browse.cpp
- BrowserValidadorAutómata/browse.cpp
- BrowserValidadorRegex/browse.cpp

Bibliografía

[DOT] Gansner, Emden R., Eleftherios Koutsofios, and Stephen North. "Drawing graphs with dot." (2015). http://graphviz.org/doc/dotguide.pdf