

# Trabajos de Algoritmos y Estructura de Datos

Esp. Ing. José María Sola, profesor.

Revisión 3.2.0

2018-05-13

---

---

---

# Tabla de contenidos

1. Introducción .....	1
2. Requisitos Generales para las Entregas de las Resoluciones .....	3
2.1. Requisitos de Forma .....	3
2.1.1. Repositorios .....	3
2.1.2. Lenguaje de Programación .....	7
2.1.3. Header Comments (Comentarios Encabezado) .....	7
2.2. Requisitos de Tiempo .....	7
3. Problemas y Soluciones .....	9
4. "Hello, World!" en C++ .....	11
4.1. Objetivos .....	11
4.2. Temas .....	11
4.3. Problema .....	11
4.4. Restricciones .....	11
4.5. Tareas .....	11
4.6. Productos .....	12
5. Resolución de Problemas — Adición .....	13
5.1. Objetivos .....	13
5.2. Temas .....	13
5.3. Problema .....	13
5.4. Restricciones .....	13
5.5. Tareas .....	14
5.6. Productos .....	14
6. Ejemplos de Valores y Operaciones de Tipos de Datos .....	15
6.1. Objetivos .....	15
6.2. Temas .....	15
6.3. Problema .....	15
6.4. Restricciones .....	15
6.5. Tareas .....	15
6.6. Productos .....	16
7. Funciones y Comparación de Valores en Punto Flotante — Celsius .....	17
7.1. Objetivos .....	17
7.2. Temas .....	17
7.3. Problema .....	17
7.4. Restricciones .....	18

7.5. Tareas .....	18
7.6. Productos .....	18
8. Funciones y Operador Condicional .....	19
8.1. Objetivos .....	19
8.2. Temas .....	19
8.3. Problema .....	19
8.4. Restricciones .....	20
8.5. Tareas .....	20
8.6. Productos .....	20
9. Precedencia de Operadores — Bisiesto .....	21
9.1. Objetivos .....	21
9.2. Temas .....	21
9.3. Problema .....	21
9.4. Restricciones .....	21
9.5. Tareas .....	22
9.6. Productos .....	22
10. Funciones Recursivas con Operador Condicional .....	23
10.1. Objetivos .....	23
10.2. Temas .....	23
10.3. Problema .....	23
10.4. Restricciones .....	24
10.5. Tareas .....	24
10.6. Productos .....	24
11. Repetición .....	25
12. Mayor de dos Números .....	27
12.1. Problema .....	27
12.2. Productos .....	27
13. Repetición de Frase .....	29
13.1. Problema .....	29
13.2. Restricciones .....	29
13.3. Productos .....	29
13.4. Entrega .....	29
14. ? Trabajo #5 — Especificación del Tipo de Dato Fecha .....	31
14.1. Tarea .....	31
14.2. Productos .....	31
15. Trabajo #9 — Browser .....	33

15.1. Necesidad .....	33
15.2. Restricciones sobre la Interacción .....	33
15.3. Restricciones de solución .....	34
15.3.1. Mejoras .....	35
15.4. Productos .....	37
Bibliografía .....	39



# Introducción

---

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

En la [sección "Trabajos" de la página del curso](#)<sup>1</sup> se indican cuales de los trabajos acá definidos que son **obligatorios** y cuales **opcionales**, como así también si se deben resolver **individualmente** o en **equipo**.

En el [sección "Calendario" de la página del curso](#)<sup>2</sup> se establece cuando es la **fecha y hora límite de entrega**,

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. **Problema:** *Descripción* del problema a resolver, la *definición completa y sin ambigüedades* es parte del trabajo.
4. **Tareas:** Plan de tareas a realizar.
5. **Restricciones:** Restricciones que deben cumplirse.
6. **Productos:** Productos que se deben entregar para la resolución del trabajo.

---

<sup>1</sup> <https://josemariasola.wordpress.com/aed/assignments/>

<sup>2</sup> <https://josemariasola.wordpress.com/aed/calendar/>





---

# 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 descritos, 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 repositorios *GitHub*. Uno público y personal y otro privado para del equipo.

```
□ Usuario
  └─ □ Repositorio público personal para la asignatura
     □ Repositorio privado 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:

- Algoritmos y Estructuras de Datos
- Curso.
- Año de cursada, y cuatrimestre si corresponde.
- Legajo.
- Apellido.
- Nombre.

```
└─ Usuario
  └─ Repositorio público personal para la asignatura
    └─ readme.md // Front page del usuario
```

## ***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:

- Algoritmos y Estructuras de Datos
- 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.

```
└─ Repositorio privado del equipo
   └─ readme.md // Front page del equipo
```

## ***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

En los enunciados de cada trabajo, el número de trabajo para utilizar en el nombre de la carpeta está generalizado con "DD", se debe reemplazar por los dos dígitos del trabajo establecidos en el curso.

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 trabajo.
- 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`.<sup>1</sup>
- archivos tests.<sup>1</sup>

```
└─ 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 GitHub
  └─ Asignatura // Repositorio personal público para a la asignatura
    └─ readme.md // Front page del usuario
    └─ 00-Hello // Carpeta 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
    └─ readme.md // Front page del equipo
    └─ 04-Stack // Carpeta de resolución de trabajo
      └─ .gitignore
      └─ readme.md // Front page de la resolución
      └─ Makefile
      └─ StackTest.cpp
      └─ Stack.h
      └─ Stack.cpp
      └─ StackApp.cpp
      └─ 01-Otro-trabajo
```

---

<sup>1</sup>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 tiene una **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.

En el [calendario del curso](https://josemariasola.wordpress.com/aed/calendar/)<sup>2</sup> se publican cuando es la fecha y hora límite de entrega de cada trabajo.

---

<sup>2</sup> <https://josemariasola.wordpress.com/aed/calendar/>



---

# 3

## Problemas y Soluciones

---

Todos los archivos `readme.md` que actúan como *Front Page* de la resolución, deben contener el *Análisis del problema* y el *Diseño de la solución*.

- **Etapla #1: Análisis del Problema.**
  - Transcripción del problema.
  - Refinamiento del problema e hipótesis de trabajo.
  - *Modelo IPO* con:
    - Entradas: nombres y tipos de datos.
    - Proceso: nombre descriptivo.
    - Salidas: nombres y tipos de datos.
- **Etapla #2: Diseño de la solución.** 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 ó 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

## "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. Problema

Adquirir y preparar los recursos necesarios para resolver los trabajos del curso.

### 4.4. Restricciones

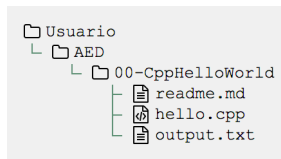
- Ninguna.

### 4.5. 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](mailto:UTNFRBAAED@yahoogroups.com)<sup>1</sup> el usuario usuario GitHub.

## 4.6. Productos



---

<sup>1</sup> <mailto:UTNFRBAAED@yahoogroups.com>

---

# 5

## Resolución de Problemas — Adición

---

### 5.1. Objetivos

- Completar todas las etapas de la resolución de problemas para un problema simple: la adición de dos números.

### 5.2. Temas

- Resolución de problemas.
- Entrada de datos.
- Enteros.
- Adición.
- Léxico.
- Representación de algoritmos.

### 5.3. Problema

Obtener del usuario dos números y mostrarle la suma.

### 5.4. Restricciones

- Ninguna.

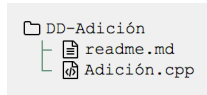
## 5.5. Tareas

1. Escribir el archivo `readme.md` que actúa como *front page* de la resolución que contenga lo solicitado en la sección “[Carpetas para cada Resolución](#)”, y en particular, el *Análisis del Problema* y el *Diseño de la Solución*:

- Etapa #1: Análisis del problema:
  - Transcripción del problema.
  - Refinamiento del problema e Hipótesis de trabajo.
  - Modelo IPO.
- Etapa #2 Diseño de la Solución:
  - Léxico del Algoritmo.
  - Representación del Algoritmo <sup>1</sup>:
    - Representación visual.
    - Representación textual.

2. Escribir, compilar, ejecutar, y probar `Adición.cpp`.

## 5.6. Productos



---

<sup>1</sup> En este trabajo en particular es necesario presentar ambas representaciones, en el resto de los trabajos se puede optar por una u otra.

---

# 6

## Ejemplos de Valores y Operaciones de Tipos de Datos

---

### 6.1. Objetivos

- Mediante un ejemplo, demostrar la aplicación de tipos de datos.

### 6.2. Temas

- Tipos de datos.
- Declaraciones.
- Variables.
- Valores.

### 6.3. Problema

Diseñar un programa C++ que ejemplifique la aplicación de los tipos de datos vistos en clases.

### 6.4. Restricciones

- No extraer valores de `cin`, usar valores literales (constantes).

### 6.5. Tareas

- Este es un *trabajo no estructurado*, que 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`.



### Crédito Extra

¿Son esos realmente todos los tipos que vimos en clase?  
Justifique.



### Crédito Extra

No utilice `cout` y sí utilice `assert` para las pruebas.

## 6.6. Productos

```
└─ DD-EjemploTipos
   └─ [readme.md]
      └─ [EjemploTipos.cpp]
```

---

## Funciones y Comparación de Valores en Punto Flotante — Celsius

---

### 7.1. Objetivos

- Demostrar el manejo de funciones y valores punto flotante.

### 7.2. Temas

- Funciones.
- Tipo `double`.
- División entera y flotante.
- Pruebas con `assert`.
- Argumentos con valor por defecto.

### 7.3. Problema

Desarrollar una función que, dada una magnitud en Fahrenheit, calcule la equivalente en Celsius:

$$\text{celsius}: \mathbb{R} \rightarrow \mathbb{R} / \text{celsius}(f) = \frac{5}{9}(f - 32)$$

Hay dos sub-problemas que se requieren solucionar antes de poder probar e implementar la función `celsius`:

- Valor de la fracción versus la división entera de la expresión `5/9` en C++.

- Representación no precisa de los tipos flotantes.

Una solución al primer problema es realizar división entre flotantes. Para el segundo problema, debemos incorporar la comparación con *tolerancia*, para eso debemos diseñar una función `bool` que reciba dos flotantes a comparar y un flotante que represente la tolerancia.

## 7.4. Restricciones

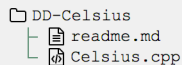
- Las pruebas deben realizarse con `assert`.
- Los prototipos deben ser:

```
double Celsius(double);  
bool AreNear(double, double, double = 0.001);
```

## 7.5. Tareas

1. Escribir el léxico, es decir, la definición matemática de la función.
2. Escribir las pruebas.
3. Escribir los prototipos.
4. Escribir las definiciones.

## 7.6. Productos



```
graph TD  
  DD-Celsius[DD-Celsius] --> readme.md[readme.md]  
  DD-Celsius --> Celsius.cpp[Celsius.cpp]
```



# Funciones y Operador Condicional

## 8.1. Objetivos

- Demostrar manejo de funciones y del operador condicional.

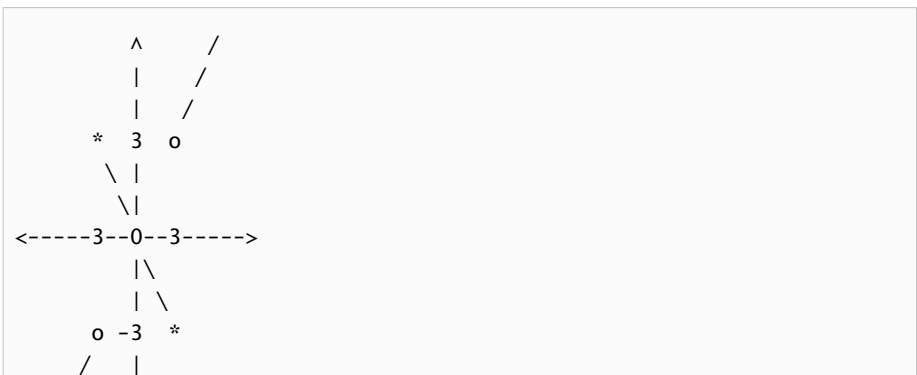
## 8.2. Temas

- Operador condicional.
- Funciones.

## 8.3. Problema

Desarrollar las siguientes funciones:

1. Valor absoluto.
2. Valor mínimo entre dos valores.
3. Función  $f_3$ , definida por:





## 8.4. Restricciones

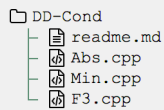
- Las pruebas deben realizarse con `assert`.
- Cada función debe aplicar el operador condicional.

## 8.5. Tareas

Por cada función:

1. Escribir el léxico, es decir, la definición matemática de la función.
2. Escribir las pruebas.
3. Escribir los prototipos.
4. Escribir las definiciones.

## 8.6. Productos



## Precedencia de Operadores — Bisiesto

---

### 9.1. Objetivos

- Demostrar el uso de operadores booleanos y expresiones complejas.

### 9.2. Temas

- Expresiones.
- Operadores booleanos: and, or, y not.
- Operador resto: %.
- Asociatividad de Operadores: ID ó DI.
- Precedencia de Operadores.
- Orden de evaluación de Operandos.
- Efecto de lado de una expresión.

### 9.3. Problema

Dado un año, determinar si es bisiesto.

### 9.4. Restricciones

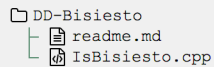
- Desarrollar la lógica en una función.

- El nombre de la función debe ser `IsBisiesto`<sup>1</sup>.
- Aplicar operadores booleanos
- No aplicar el operador condicional.
- No aplicar `if` ni `switch`.

## 9.5. Tareas

1. Escribir el léxico, es decir, la definición matemática de la función.
2. Escribir las pruebas.
3. Escribir el prototipo.
4. Escribir la definición.
5. Incluir en `readme.md` el *árbol de expresión* asociado a la expresión de retorno de la función.

## 9.6. Productos



```
graph TD;
  DD-Bisiesto[DD-Bisiesto] --> readme.md[readme.md];
  DD-Bisiesto --> IsBisiesto.cpp[IsBisiesto.cpp];
```

---

<sup>1</sup> Es una práctica común utilizar el prefijo `Is` para predicados, es decir, funciones que retornan un valor lógico.

---

# 10

## Funciones Recursivas con Operador Condicional

---

### 10.1. Objetivos

- Demostrar manejo de funciones definidas recursivamente e implementadas con el operador condicional.

### 10.2. Temas

- Funciones recursivas.
- Operador condicional.

### 10.3. Problema

Desarrollar las siguientes funciones:

1. División entera de naturales: `div`.
2. MCD (Máximo Común Denominador): `mcd`.
3. Factorial: `fact`.



Un número factorial puede ser muy grande, por eso hay que elegir el tipo de la función correctamente.

4. Fibonacci: `Fib`.

## 10.4. Restricciones

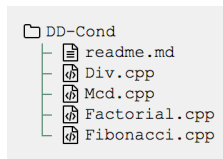
- Las pruebas deben realizarse con `assert`.
- Cada función debe aplicar el operador condicional.

## 10.5. Tareas

Por cada función:

1. Escribir el léxico, es decir, la definición matemática de la función.
2. Escribir las pruebas.
3. Escribir los prototipos.
4. Escribir las definiciones.

## 10.6. Productos



---

# 11

## Repetición

---





---

# 12

## Mayor de dos Números

---

### 12.1. Problema

Dado dos números informar cuál es el mayor.

### 12.2. Productos

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



---

# 13

## Repetición de Frase

---

### 13.1. Problema

Enviar una frase a la salida estándar muchas veces.

### 13.2. Restricciones

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

- Salto condicional.
- Iterativa estructurada.

### 13.3. Productos

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

### 13.4. Entrega

- Abr 27, 13hs.



## ? Trabajo #5 — Especificación del Tipo de Dato Fecha

---

### 14.1. Tarea

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

### 14.2. Productos

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



## Trabajo #9 — Browser

---

### 15.1. Necesidad

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

### 15.2. 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 15.1. 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

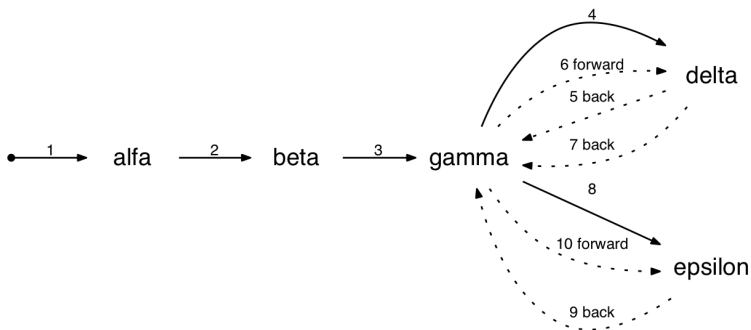


Figura 15.1. Líneas de tiempo (BTTF2) para la interacción ejemplo.

### 15.3. 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 siguiente URL.

### **15.3.1. 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 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

## 15.4. 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>

