

Trabajos de Algoritmos y Estructura de Datos

Esp. Ing. José María Sola, profesor.

Revisión 4.2.0

2020-11-01

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	8
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	22
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. Enumeraciones	25
11.1. Productos	25
12. Uniones	27
12.1. Productos	27
13. Tipo Color	29
13.1. Objetivos	29
13.2. Temas	29
13.3. Problema	29
13.4. Restricciones	31
13.5. Tareas	32
13.6. Productos	32
14. Geometría — Desarrollo de Tipos	35
14.1. Introducción	35

14.2. Objetivos	35
14.3. Temas	35
14.4. Problema	36
14.5. Restricciones	37
14.6. Tareas	38
14.7. Productos	38
15. Geometría Parte II — Input/Output	41
15.1. Problema	41
15.2. Restricciones	41
15.3. Tareas	42
15.4. Productos	42
16. Geometría Parte III — Renderizar	43
17. Geometría Parte IV — Estructuras Enlazadas	45
18. Diagonal de una Matriz	47
18.1. Objetivos	47
18.2. Restricciones	47
18.3. Productos	47
19. Secuencia Dinámica — Implementación Contigua	49
19.1. Restricciones	49
19.2. Tareas	49
19.3. Productos	49
20. Templates	51
20.1. Objetivos	51
21. Stack — Implementación Contigua	53
21.1. Restricciones	53
21.2. Tareas	53
21.3. Productos	53
22. Queue — Implementación Contigua	55
22.1. Restricciones	55
22.2. Tareas	55
22.3. Productos	55
23. Secuencia Dinámica — Implementación Enlazada	57
23.1. Restricciones	57
23.2. Tareas	57
23.3. Productos	57
24. Stack — Implementación Enlazada	59

24.1. Restricciones	59
24.2. Tareas	59
24.3. Productos	59
25. Queue — Implementación Enlazada	61
25.1. Restricciones	61
25.2. Tareas	61
25.3. Productos	61
26. Árbol de Búsqueda Binaria	63
26.1. Objetivos	63
26.2. Temas	63
26.3. Problema	63
26.4. Restricciones	63
26.5. Tareas	63
26.6. Productos	64
27. Repetición	65
28. Mayor de dos Números	67
28.1. Problema	67
28.2. Productos	67
29. Repetición de Frase	69
29.1. Problema	69
29.2. Restricciones	69
29.3. Productos	69
29.4. Entrega	69
30. ? Trabajo #5 — Especificación del Tipo de Dato Fecha	71
30.1. Tarea	71
30.2. Productos	71
31. Trabajo #9 — Browser	73
31.1. Necesidad	73
31.2. Restricciones sobre la Interacción	73
31.3. Restricciones de solución	74
31.3.1. Mejoras	75
31.4. Productos	77
Bibliografía	79

Lista de figuras

31.1. Líneas de tiempo (BTTF2) para la interacción ejemplo. 74

Lista de tablas

31.1. Ejemplo de interacción 73

Lista de ejemplos

2.1. Nombre de carpeta	5
2.2. Header comments	8

Introducción

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

En la [sección "Trabajos" de la página del curso](#)¹ 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](#)² 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.

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

² <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.

Repositorios público y privado.

```
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.

Repositorio personal para la asignatura.

```
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.

```
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`.¹
- archivos `tests`.¹

Carpeta de resolución de trabajo.

```
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

Ejemplo completo.

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

```
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
```

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/)² se publican cuando es la fecha y hora límite de entrega de cada trabajo.

² <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++ 20** (ó **C++ 17** ó **C++ 14** ó **C++ 11**).
8. Indicar en `readme.md` el compilador seleccionado.
9. Probar el compilador con un programa `hello.cpp` que envíe a `cout` la línea `hello, world!` o similar.
- 10 Ejecutar el programa, y capturar su salida en un archivo `output.txt`.
11. Publicar en el repositorio personal AED la carpeta `00-cppHelloWorld` con `readme.md`, `hello.cpp`, y `output.txt`.
- 12 La última tarea es informar por email a UTNFRBAAED@yahoogroups.com¹ el usuario *GitHub*.

4.6. Productos

```

Usuario
|-- AED
    |-- 00-cppHelloWorld
        |-- readme.md
        |-- hello.cpp
        |-- output.txt

```

¹ <mailto:UTNFRBAAED@yahoogroups.com>

5

Resolución de Problemas — Adición

5.1. Objetivos

- Demostrar, mediante un problema simple, el conocimiento de las etapas de resolución de problemas.

5.2. Temas

- Resolución de problemas.
- Entrada de datos.
- Tipos numéricos.
- 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 ???, 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 ¹:
 - Representación visual.
 - Representación textual.

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

5.6. Productos

```
DD-Adición
|-- readme.md
`-- Adición.cpp
```

¹ 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

- Demostrar la aplicación de tipos de datos mediante un programa ejemplo.

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:

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

- Valor de la fracción $\frac{5}{9}$ 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

```
DD-Celsius  
|-- readme.md  
`-- Celsius.cpp
```

Funciones y Operador Condicional

8.1. Objetivos

- Demostrar manejo de funciones y del operador condicional.

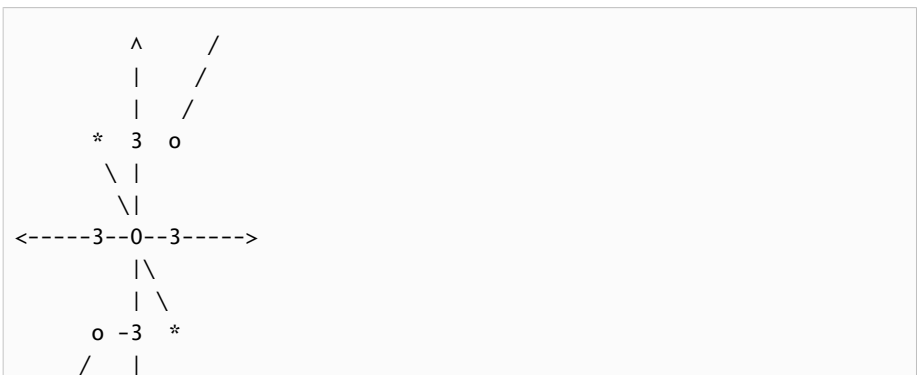
8.2. Temas

- 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:



/	
/	v

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

```
DD-Cond
|-- readme.md
|-- Abs.cpp
|-- Min.cpp
`-- F3.cpp
```


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.
- Funciones.

9.3. Problema

Desarrollar una función que dado un año, determinar si es bisiesto.

9.4. Restricciones

- El nombre de la función debe ser `IsBisiesto` ¹.
- Aplicar operadores booleanos
- No aplicar el operador condicional.
- No aplicar `if` ni `switch`.
- Las pruebas deben realizarse con `assert`.

9.5. Tareas

1. Escribir 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

```
DD-Bisiesto
|-- readme.md
`-- IsBisiesto.cpp
```

¹ 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` [\[PINEIRO\]](#).
3. Factorial: `fact`.



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

4. Fibonacci: `Fib`.



Notar que esta función es doblemente recursiva.

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

```
DD-Recur
|-- readme.md
|-- Div.cpp
|-- Mcd.cpp
|-- Factorial.cpp
`-- Fibonacci.cpp
```

11

Enumeraciones

1. Escriba un programa que declare una variable que pueda almacenar cualquier punto cardinal.
2. Extender el programa de la sección [programa de la sección 1.5. Funciones que Retornan o Reciben Tipos Enum del texto "Enumeraciones"](#)¹ para que contenga una función que dado un día y turno, informe la asignatura que debemos cursar.

11.1. Productos

```
DD-Enum
|-- Cardinal.cpp
`-- SemanaDeCursada.cpp
```

¹ <https://josemariasola.wordpress.com/aed/papers/#Enums>

12

Uniones

1. Escriba un programa ejemplo que opere sobre dos variables:
 - una que almacene tanto enteros (ints) como naturales (unsigneds).
 - y otra que almacene tanto caracteres (chars) como reales (doubles).
2. Extender el programa [Caninos del texto "Uniones"](#)¹ para que incluya las siguientes variables:
 - a. [Santas](#)²
 - b. [wileE](#)³
 - i. ¿El cambio es simplemente agregar una variable?
 - c. [snowball2](#)⁴ y [simba](#)⁵
 - i. ¿El cambio es simplemente agregar dos variables?
 - ii. ¿Deberían existir en el programa conceptos como Mamífero ó carnívoro?

12.1. Productos

```
DD-Union
|-- EjemploDeUniones.cpp
```

¹ <https://josemariasola.wordpress.com/aed/papers/#Unions>

² https://en.wikipedia.org/wiki/Santa%27s_Little_Helper

³ https://en.wikipedia.org/wiki/Wile_E._Coyote_and_the_Road_Runner

⁴ https://en.wikipedia.org/wiki/Simpson_family#Snowball_II

⁵ <https://en.wikipedia.org/wiki/Simba>

```
-- Animados.cpp
```

13

Tipo Color

13.1. Objetivos

- Demostrar capacidad de construcción de tipos compuestos basados en tipos existentes y simples, es decir, no compuestos.

13.2. Temas

- Tipo de dato definido por el usuario (programador).
- Tipo Abstracto de Datos.
- Especificación.
- Implementación.
- Definición de conjunto de valores con `struct`.
- Tipos enteros de ancho fijo.
- Variables externas.
- Variables `const`.

13.3. Problema

Diseñar un tipo Color basado en el [modelo RGB¹](https://en.wikipedia.org/wiki/RGB_color_model), con tres canales de 8 bits. Todo color está compuesto por tres componentes: intensidad de *red* (rojo), de *green* (verde), y de *blue* (azul). Cada intensidad está en el rango [0, 255]. Definir los valores para rojo, azul, verde, cyan, magenta, amarillo, negro, y blanco. Dos

¹ https://en.wikipedia.org/wiki/RGB_color_model

colores se pueden mezclar, lo cual produce un nuevo color que tiene el promedio de intensidad para cada componente.



Crédito Extra

La operación *Mezclar* mezcla en partes iguales; desarrollar una variante de la operación que permita indicar las proporciones de las partes.



Crédito Extra

Desarrollar la operaciones *Sumar* y *Restar* que dados dos colores suma o resta la intensidad de cada canal, siempre dando resultados en el rango [0, 255]. Utilizá estas operaciones para inicializar los colores secundarios, blanco, y negro.



Crédito Extra

Desarrollar la operación *GetComplementario* que dado un color obtiene el complementario u opuesto. Por ejemplo, el complementario de rojo es cyan.



Crédito Extra

Desarrollar la operación *GetHtmlHex* que genera un string con la representación hexadecimal para HTML de un color. Por ejemplo, `assert("#0000ff" == GetHtmlHex(azu1));`



Crédito Extra

Desarrollar la operación *GetHtmlRgb* que genera un string con la representación rgb para HTML de un color. Por ejemplo `assert("rgb(0,0,255)" == GetHtmlRgb(azu1));`



Crédito Extra

Codificar la función *CrearSvgConTextoEscritoEnAltoContraste* que dado un nombre archivo sin extensión, un texto, y un color de letra

genera un archivo **SVG**² con el texto en un color y fondo en su complementario.

Por ejemplo
`CrearSvgConTextoEscritoEnAltoContraste("Mensaje", "¡Hola, Mundo!", cyan)` genera el archivo `Mensaje.svg` con el siguiente contenido:

```
<svg xmlns="http://www.w3.org/2000/svg">
  <rect x="0" y="0" height="30" width="120"
    style="fill: #ff0000"/>
  <text x="5" y="18" style="fill:
    rgb(0,255,255);background-color: #ff0000">
    ¡Hola, Mundo!
  </text>
</svg>
```

Que se visualiza así:



Notar que el fondo tiene el color complementario del texto y que, tan solo por fines ilustrativos, el color de fondo se establece en notación hexadecimal, y el color del texto en notación rgb.

13.4. Restricciones

- Las operaciones de proyección para *red*, *green*, y *blue* se implementan con acceso directo a los componentes, no es necesario definir *getters* especiales. Por la misma razón, los *setters* no son necesarios.
- Utilizar el tipo `uint8_t` de `cstdint`, si no es posible, usar `unsigned char`.
- Los colores primarios, secundarios, negro y blanco deben implementarse como ocho variables declaradas fuera de `main` y de toda función, con el calificador `const` para que no puedan modificarse.

² https://en.wikipedia.org/wiki/Scalable_Vector_Graphics

- Implementar la operación `is Igual` que retorna `true` si un color es igual a otro, si no, `false`.



Crédito Extra

Responder en `readme.md` porqué se debe usar `uint8_t`. Si tu compilador no te permite usar `uint8_t`, indicar porqué es correcto usar `unsigned char` pero no `char`.

13.5. Tareas

1. Especificar matemáticamente el tipo en `color.md`:
 - a. Especificar el conjunto de operaciones.
 - b. Especificar el conjunto de valores.
2. Diseñar y codificar las pruebas en `main`.
3. Declarar los prototipos de las operaciones arriba de `main`.
4. Declarar `color` antes de los prototipos las operaciones.
5. Compilar: Luego de finalizar tareas anteriores, estamos en condiciones de compilar. Deberíamos obtener error de *linkeo* (i.e., vinculación) pero no de compilación.
6. Codificar las definiciones de las operaciones, debajo de `main`.
7. Probar: Luego de las definiciones, deberíamos poder realizar el proceso de traducción completo (i.e., compilación y linkeo) sin errores. Una vez obtenido el programa ejecutable, deberíamos poder ejecutarlo sin errores.

13.6. Productos

```
DD-Color
|-- readme.md
|-- color.md      // Especificación
`-- color.cpp     // Implementación y pruebas
```



Crédito Extra

Estructurar la solución con separación física en archivos de:

- pruebas,
- de parte pública de la implementación, y
- de parte privada de la implementación.

Escribir un `makefile` que construya y pruebe la solución.
Estos temas están desarrollados en [\[Interfaces-Make\]](#)

```
DD-Color
|-- readme.md
|-- Makefile
|-- Color.md      // Especificación
|-- Color.h       // Implementación: Parte Pública
|-- ColorTest.cpp // Pruebas
`-- Color.cpp     // Implementación: Parte Privada
```


Geometría — Desarrollo de Tipos

14.1. Introducción

Este trabajo se hace uso del [Capítulo 13, Tipo Color](#) y es el primero de una secuencia de trabajos que aplican tipos para solucionar problemas de geometría.

Este trabajo tiene como tema central la construcción de tipos mediante producto cartesiano; el tema se desarrolla en [\[Structs-Arrays\]](#).

14.2. Objetivos

- Demostrar capacidad de construcción de tipos compuestos basados en otros tipos, simples o compuestos, existentes o nuevos.

14.3. Temas

- Tipo de dato definido por el usuario (programador).
- Tipo Abstracto de Datos.
- Especificación.
- Implementación.
- Definición de conjunto de valores con `struct`.
- Definición de conjunto de operaciones con funciones y pasaje de argumentos por referencia (i.e., variable).
- Estructura dinámica con capacidad máxima.

14.4. Problema

Construir el tipo ??? con color. Un polígono tiene una cantidad dinámica de vértices, y el tipo debe incluir las operaciones para agregar, remover, acceder y modificar esos vértices.



Crédito Extra

Los tipos y operaciones marcados como opcionales son crédito extra. También puedes agregar las operaciones que quieras.

Tipo	Valores	Operaciones
Punto	Representa un punto en el plano con coordenadas cartesianas.	<ul style="list-style-type: none">• IsIgual• GetDistancia• GetDistanciaAlOrigen• GetRho (opcional)• GetPhi (opcional)• GetCuadrante (opcional)• GetEje (opcional)• GetSemiplano (opcional)• Mover (opcional)
Círculo (opcional)	Representa un círculo con color en el plano.	<ul style="list-style-type: none">• GetCircunferencia• GetÁrea• Mover
Triángulo (opcional)	Representa triángulos con color en el plano, se lo describe por tres puntos y su color.	<ul style="list-style-type: none">• GetPerímetro• GetÁrea• IsEscaleno• IsEquilátero• IsIsósceles• GetTipo (opcional)• GetCentro (opcional)

Tipo	Valores	Operaciones
Rectángulo (<i>opcional</i>)	Representa rectángulos con color en el plano, con lados paralelos a los ejes.	<ul style="list-style-type: none"> • GetBase • GetAltura • GetPerímetro • GetÁrea • GetLongitudDiagonal • IsCuadrado • GetVértice (<i>opcional</i>): Retorna el punto correspondiente a cada uno de los cuatro vértices, es decir, <i>SuperiorIzquierdo</i>, <i>SuperiorDerecho</i>, <i>InferiorIzquierdo</i>, e <i>InferiorDerecho</i>.
Polígono	Representa polígonos con color en el plano.	<ul style="list-style-type: none"> • AddVértice • GetVértice • SetVértice • RemoveVértice • GetCantidadLados • GetPerímetro

14.5. Restricciones

- Se debe usar el [Capítulo 13, Tipo Color](#) construido previamente.
- Los vértices deben ser del tipo *Punto*.
- La secuencia dinámica de vértices debe implementarse con un array que contenga los elementos y un unsigned que indique cuantos vértices tiene realmente. Ese unsigned es menor o igual al tamaño del array.
- Los vértices deben ser del tipo *Punto*.
- Las pruebas deben realizarse con assert, sin usar cin ni cout.

14.6. Tareas

Por cada tipo de dato:

1. Especificar el tipo matemáticamente.
2. Diseñar y codificar las pruebas en main.
3. Implementar el tipo.

14.7. Productos

```
DD-Geometría
|-- readme.md
|-- Geometría.md      // Especificación todos los tipos
`-- Geometría.cpp     // Implementación y pruebas de todos los tipos
```



Crédito Extra

Estructurar la solución con separación física en archivos de pruebas, de implementación parte privada, y de implementación parte pública.

Escribir un `makefile` que construya y pruebe la solución. Estos temas están desarrollados en [\[Interfaces-Make\]](#)

```
DD-Geometría
|-- readme.md
|-- Makefile
|-- Color.md          // Especificación
|-- Color.h           // Implmntcn Parte Pública
|-- ColorTest.cpp     // Pruebas
|-- Color.cpp         // Implmntcn Parte Privada
|-- Punto.md          // Especificación
|-- Punto.h           // Implmntcn Parte Pública
|-- PuntoTest.cpp     // Pruebas
|-- Punto.cpp         // Implmntcn Parte Privada
|-- Círculo.md        // Especificación
|-- Círculo.h         // Implmntcn Parte Pública
|-- CírculoTest.cpp   // Pruebas
|-- Círculo.cpp       // Implmntcn Parte Privada
|-- Triángulo.md      // Especificación
|-- Triángulo.h       // Implmntcn Parte Pública
```

```
|-- TriánguloTest.cpp    // Pruebas
|-- Triángulo.cpp        // Implmntcn Parte Privada
|-- Rectángulo.md        // Especificación
|-- Rectángulo.h         // Implmntcn Parte Pública
|-- RectánguloTest.cpp   // Pruebas
|-- Rectángulo.cpp        // Implmntcn Parte Privada
|-- Polígono.md          // Especificación
|-- Polígono.h           // Implmntcn Parte Pública
|-- PolígonoTest.cpp     // Pruebas
|-- Polígono.cpp         // Implmntcn Parte Privada
```


Geometría Parte II — Input/Output

15.1. Problema

Dado un archivo con polígonos, copiar a un nuevo archivos los polígonos que tienen un perímetro menor a un valor x .

15.2. Restricciones

- La conexión a los archivos debe ser mediante `streams`.
- Si fuese necesario utilizar `in.clear()` para limpiar el estado erróneo y volver a leer de un `stream`.
- A los tipos deben agregarse operaciones de extracción e inserción según los siguientes prototipos, donde T es el nombre del tipo:

```
bool ExtraerT(istream& in, T& v);
```

```
bool InsertarT(ostream& out, const T& v);
```



Crédito Extra

Utilizar *interfaz fluida* [\[FLUENT\]](#) con estos prototipos para extracción e inserción respectivamente:

```
istream& ExtraerT(istream& in, T& v);
```

```
ostream& InsertarT(ostream& out, const T& v);
```

- La solución debe desarrollarse en una función:

```
void CopiarPolígonosConPerímetrosMayoresA(double x, string  
nombreArchivoIn, string nombreArchivoOut);
```

15.3. Tareas

1. Diseñar la representación que cada tipo va a tener en los flujos.
2. (Opcional) Especificar matemáticamente la operación inserción para cada tipo, la especificación de la operación extracción es simplemente: *"La operación Extraer debe poder extraer un valor insertado por la operación Insertar"*.
3. Agregar las pruebas del par de operaciones para cada tipo.

15.4. Productos

Este trabajo modifica los productos del trabajo anterior.

Si decidiste hacer un solo archivo `Geometría.cpp` con su `main` invocá a `CopiarPolígonosConPerímetrosMayoresA` al final del `main`.

Si decidiste separar los archivos, creá un nuevo archivo llamado `Filtrar.cpp` con un `main` que invoque a la función `CopiarPolígonosConPerímetrosMayoresA` y con la implementación de esa función.

16

Geometría Parte III — Renderizar

17

Geometría Parte IV — Estructuras Enlazadas

18

Diagonal de una Matriz

18.1. Objetivos

- Escribir un programa que determine la suma de la diagonal de una matriz.

18.2. Restricciones

- La suma la debe calcular una función que tenga como parámetro *in* una matriz.

18.3. Productos

```
DD-DiagonalMatriz
|-- readme.md
`-- DiagonalMatriz.cpp
```

19

Secuencia Dinámica — Implementación Contigua

19.1. Restricciones

- La implementación debe basarse en array, por lo tanto tienen una capacidad máxima.

19.2. Tareas

- Especificar tipo.
- Diseñar pruebas.
- Implementar parte pública.
- Implementar parte privada.
- Probar.
- Diseñar un programa de aplicación.

19.3. Productos

```
DD-SecDinCont
|-- readme.md
|-- SecDin.md // Especificación.
|-- SecDinTest.cpp
|-- SecDin.h
|-- SecDinCont.cpp
`-- SecDinApp.cpp
```

20

Templates

20.1. Objetivos

- Matriz con cantidad y tipo de elemento parametrizado.
- Secuencia Dinámica Contigua con cantidad y tipo de elemento parametrizado.

21

Stack — Implementación Contigua

21.1. Restricciones

- La implementación debe basarse en array, por lo tanto tienen una capacidad máxima.

21.2. Tareas

- Especificar tipo.
- Diseñar pruebas.
- Implementar parte pública.
- Implementar parte privada.
- Probar.
- Diseñar un programa de aplicación.

21.3. Productos

```
DD-StackCont
|-- readme.md
|-- Stack.md // Especificación.
|-- StackTest.cpp
|-- Stack.h
|-- StackCont.cpp
`-- StackApp.cpp
```

22

Queue — Implementación Contigua

22.1. Restricciones

- La implementación basarse en array, por lo tanto tienen una capacidad máxima.
- El array debe utilizarse como un array circular con aritmética módulo N.

22.2. Tareas

- Especificar tipo.
- Diseñar pruebas.
- Implementar parte pública.
- Implementar parte privada.
- Probar.
- Diseñar un programa de aplicación.

22.3. Productos

```
DD-QueueCont
|-- readme.md
|-- Queue.md // Especificación.
|-- QueueTest.cpp
|-- Queue.h
|-- QueueCont.cpp
`-- QueueApp.cpp
```

23

Secuencia Dinámica — Implementación Enlazada

23.1. Restricciones

- La implementación deben basarse en una `struct` con un puntero al primer nodo.
- La reserva de memoria para los nodos debe realizarse dinámicamente con el operador `new`.

23.2. Tareas

- Especificar tipo.
- Diseñar pruebas.
- Implementar parte pública.
- Implementar parte privada.
- Probar.
- Diseñar un programa de aplicación.

23.3. Productos

```
DD-SecDinLink
|-- readme.md
|-- SecDin.md // Especificación.
|-- SecDinTest.cpp
|-- SecDin.h
```

```
|-- SecDinLink.cpp  
|-- SecDinApp.cpp
```

24

Stack — Implementación Enlazada

24.1. Restricciones

- La implementación basarse en un struct con un puntero al nodo de la cima.
- La reserva de memoria para los nodos debe realizarse dinámicamente con el operador new.

24.2. Tareas

- Especificar tipo.
- Diseñar pruebas.
- Implementar parte pública.
- Implementar parte privada.
- Probar.
- Diseñar un programa de aplicación.

24.3. Productos

```
DD-StackCont
|-- readme.md
|-- stack.md // Especificación.
|-- StackTest.cpp
|-- stack.h
|-- StackLink.cpp
`-- StackApp.cpp
```

25

Queue — Implementación Enlazada

25.1. Restricciones

- La implementación basarse en una struct con un puntero al primer nodo y otro al último.
- La reserva de memoria para los nodos debe realizarse dinámicamente con el operador new.

25.2. Tareas

- Especificar tipo.
- Diseñar pruebas.
- Implementar parte pública.
- Implementar parte privada.
- Probar.
- Diseñar un programa de aplicación.

25.3. Productos

```
DD-QueueLink
|-- readme.md
|-- Queue.md // Especificación.
|-- QueueTest.cpp
|-- Queue.h
|-- QueueLink.cpp
`-- QueueApp.cpp
```

26

Árbol de Búsqueda Binaria

26.1. Objetivos

- Objetivo.
- Objetivo.
- Objetivo.

26.2. Temas

- Tema.
- Tema.
- Tema.

26.3. Problema

Problema

26.4. Restricciones

- Restricción.
- Restricción.
- Restricción.

26.5. Tareas

1. Tarea.

2. Tarea.

3. Tarea.

26.6. Productos

27

Repetición

28

Mayor de dos Números

28.1. Problema

Dado dos números informar cuál es el mayor.

28.2. Productos

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

29

Repetición de Frase

29.1. Problema

Enviar una frase a la salida estándar muchas veces.

29.2. Restricciones

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

- Salto condicional.
- Iterativa estructurada.

29.3. Productos

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

29.4. Entrega

- Abr 27, 13hs.

? Trabajo #5 — Especificación del Tipo de Dato Fecha

30.1. Tarea

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

30.2. Productos

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

Trabajo #9 — Browser

31.1. Necesidad

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

31.2. 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 31.1. 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

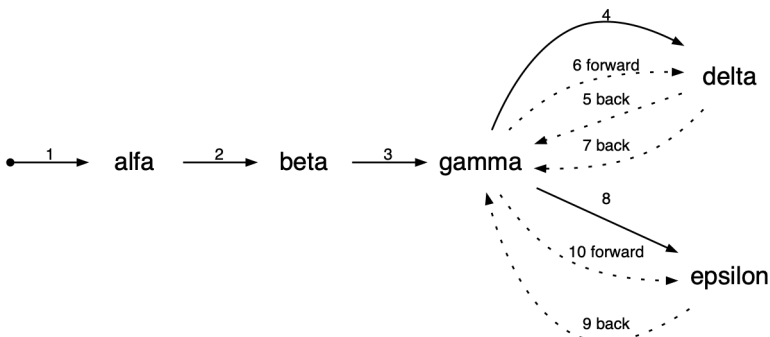


Figura 31.1. Líneas de tiempo (BTTF2) para la interacción ejemplo.

31.3. 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]; // 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.

31.3.1. 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

31.4. Productos

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

Bibliografía

Emden R. Gansner and Eleftherios Koutsofios and Stephen North.
Drawing graphs with dot (2015) Retrived 2018-06-19 from [https://
www.graphviz.org/pdf/dotguide.pdf](https://www.graphviz.org/pdf/dotguide.pdf)

Interfaz Fluida https://en.wikipedia.org/wiki/Fluent_interface

Git 101 <https://josemariasola.wordpress.com/papers#Git101>

Compiladores, Editores y Entornos de Desarrollo: Instalación, Configuración y Prueba [https://josemariasola.wordpress.com/papers/
#CompiladoresInstalacion](https://josemariasola.wordpress.com/papers/#CompiladoresInstalacion)

José María Sola. *Interfaces & Make* (2017) [https://josemariasola.wordpress.com/
ssl/papers#Interfaces-Make](https://josemariasola.wordpress.com/ssl/papers#Interfaces-Make)

Brian W. Kernighan and Dennis Ritchie. *The C Programming Language, 2nd Edition* (1988)

Jorge Muchnik y Ana María Díaz Bott. *SSL, 2da Edición* (tres volúmenes) (2012)

María Alicia Piñeiro. *Matemática Discreta Unidad 3 Divisibilidad en \mathbb{Z}* (2019) <https://josemariasola.wordpress.com/aed/reference#gcd>

José María Sola. *Tuplas & Secuencias y Structs & Arrays: Construcción de Tipo por Producto Cartesiano* (2018) [https://josemariasola.wordpress.com/
aed/papers#Structs-Arrays](https://josemariasola.wordpress.com/aed/papers#Structs-Arrays)

