

# Informe del Laboratorio 01

## Tema 1: Git y GitHub

Nota

Estudiante	Escuela	Asignatura
Rodrigo E. Infanzón Acosta <a href="mailto:rinfanzona@ulasalle.edu.pe">rinfanzona@ulasalle.edu.pe</a>	Carrera Profesional de Ingeniería de Software	Lenguaje de Programación 3 <b>Semestre IV</b>

Laboratorio	Tema	Duración
01	<b>Git y GitHub</b>	06 horas académicas

Semestre académico	Fecha de inicio	Fecha de entrega
2024 - A	21/03/24	27/03/24

### Actividades a realizar:

- Elaborar un informe individual, acerca de las plataformas de control de versiones: Git y GitHub.
- Elaborar la solución y explicación de la creación de un tablero M x N, asimismo, la actualización del tablero.cpp usando el primer apellido del estudiante o una "X" en la diagonal.
- Elaborar la solución al problema usando Git y GitHub, seleccionado por el profesor, denominado **"11451. El perrito que quiere un hueso"**, extraído de la plataforma **omegaUp**: <https://omegaup.com/arena/problem/El-perrito-que-quiere-un-hueso/>
- Mencionar **tres aportes** a su adquisición de conocimientos que este laboratorio le proporcionó.
- Utilizar todas las recomendaciones dadas por el docente en clase, tales como:
  - **Antecedentes:** describir antecedentes previos que sean necesarios para desarrollar el laboratorio. Las entregadas por el docente y/o las que se buscaron personalmente.
  - **Commits:** elaborar la lista de envíos que permitirán culminar el laboratorio, previo a la implementación.
  - **Source:** explicar porciones de código fuente importantes, trascendentales que permitieron resolver el laboratorio y que reflejen su particularidad única, sólo en trabajos grupales se permite duplicidad.
  - **Ejecución:** muestra comandos, capturas de pantalla, explicando la forma de replicar y ejecutar el entregable del laboratorio.
- Entregables:
  - URL al directorio específico del laboratorio en su repositorio GitHub privado donde esté todo el código fuente y otros que sean necesarios. Evitar la presencia

de archivos: binarios, objetos, archivos temporales. Incluir archivos de especificación como: packages.json, requirements.txt o README.md.

- No olvidar que el docente debe ser siempre colaborador a su repositorio que debe ser privado. (Usuario del docente: **@rescobedoulasalle**).
- Se debe de describir sólo los commits más importantes que marcaron hitos en su trabajo, adjuntando capturas de pantalla, del commit, porciones de código fuente, evidencia de sus ejecuciones y pruebas.
- En el informe siempre se debe explicar las imágenes (código fuente, capturas de pantalla, commits, ejecuciones, pruebas, etc.) con descripciones puntuales pero precisas.
- Agregar la estructura de directorios y archivos de su laboratorio.

## Recursos y herramientas utilizados:

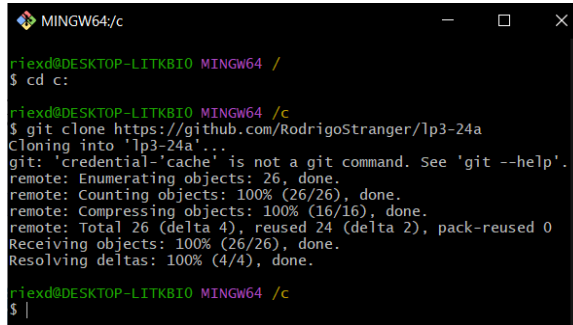
1. Sistema operativo utilizado: Windows 10 pro 22H2 de 64 bits (SO. 19045.4170).
2. Hardware: Ryzen 5 3550H 2.10 GHz, RAM 16 GB DDR4 2400 MHz.
3. Visual Studio Code (versión 1.87.2).
4. Git (versión 2.44.0).
5. Cuenta de GitHub creada con el correo institucional proporcionado por la Universidad La Salle de Arequipa ([rinfanzona@ulasalle.du.pe](mailto:rinfanzona@ulasalle.du.pe)).
6. Conocimientos básicos en Git.
7. Conocimientos básicos sobre programación.

## Información del repositorio en GitHub:

1. Enlace del repositorio en GitHub: <https://github.com/RodrigoStranger/lp3-24a>
2. Enlace del repositorio para el laboratorio 01 en GitHub:  
<https://github.com/RodrigoStranger/lp3-24a/tree/main/lab01>

## Actividades realizadas con el repositorio de GitHub en Local:

1. Como previamente en el laboratorio 15/03/24 realizamos la creación de nuestro repositorio, configurándolo en consola, añadiendo de colaborador a nuestro docente, configurando nuestro token y resolviendo un problema de programación, en la cual pusimos en práctica el uso básico del Git y GitHub, por lo cual, en nuestro equipo local personal, tendríamos que poner en práctica el uso del servidor GitHub, para clonar nuestro repositorio existente en la nube, hacia nuestro equipo personal. Por ende, se realiza de la siguiente manera:
  1. **Seleccionar un sistema de comandos:** utilizo la terminal de comandos establecida por Git, denominada **Git Bash**, con ruta en *C:\ProgramData\Microsoft\Windows\Start Menu\Programs\Git*
  2. **Determinar en qué directorio se va a clonar el repositorio:** debemos especificar en que directorio de nuestro disco local queremos almacenar nuestro repositorio, en mi caso: *C:\*. Elegido el directorio en la consola, se debe ejecutar el comando: **git clone** <https://github.com/RodrigoStranger/lp3-24a> :



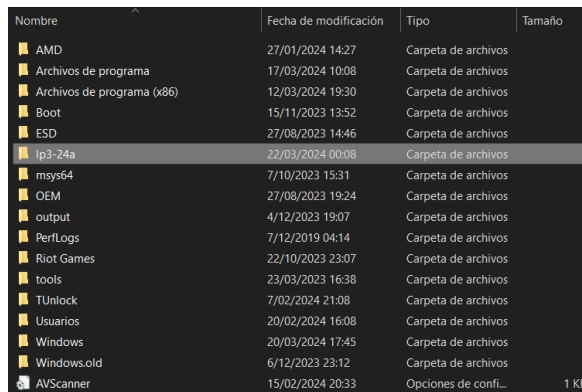
```

MINGW64/c
riexd@DESKTOP-LITKB10 MINGW64 /
$ cd c:
riexd@DESKTOP-LITKB10 MINGW64 /c
$ git clone https://github.com/RodrigoStranger/lp3-24a
Cloning into 'lp3-24a'...
git: 'credential-cache' is not a git command. See 'git --help'.
remote: Enumerating objects: 26, done.
remote: Counting objects: 100% (26/26), done.
remote: Compressing objects: 100% (16/16), done.
remote: Total 26 (delta 4), reused 24 (delta 2), pack-reused 0
Receiving objects: 100% (26/26), done.
Resolving deltas: 100% (4/4), done.
riexd@DESKTOP-LITKB10 MINGW64 /c
$ |

```

Imagen con propiedad del autor ©Rodrigo Infanzon Acosta, ©Git

- Producto del resultado en la consola de **Git Bash**, el repositorio clonado, lp3-24a existe en nuestro equipo local, listo para trabajar según las metas establecidas por nuestro docente:



Nombre	Fecha de modificación	Tipo	Tamaño
AMD	27/01/2024 14:27	Carpeta de archivos	
Archivos de programa	17/03/2024 10:08	Carpeta de archivos	
Archivos de programa (x86)	12/03/2024 19:30	Carpeta de archivos	
Boot	15/11/2023 13:52	Carpeta de archivos	
ESD	27/08/2023 14:46	Carpeta de archivos	
lp3-24a	22/03/2024 00:08	Carpeta de archivos	
msys64	7/10/2023 15:31	Carpeta de archivos	
OEM	27/08/2023 19:24	Carpeta de archivos	
output	4/12/2023 19:07	Carpeta de archivos	
PerfLogs	7/12/2019 04:14	Carpeta de archivos	
Riot Games	22/10/2023 23:07	Carpeta de archivos	
tools	23/03/2023 16:38	Carpeta de archivos	
TUnlock	7/02/2024 21:08	Carpeta de archivos	
Usuarios	20/02/2024 16:08	Carpeta de archivos	
Windows	20/03/2024 17:45	Carpeta de archivos	
Windows.old	6/12/2023 23:12	Carpeta de archivos	
AVScanner	15/02/2024 20:33	Opciones de confi...	1 KB

Imagen con propiedad del autor ©Rodrigo Infanzon Acosta

## Desarrollo de las actividades:

### Informe individual: Git y GitHub

#### Git

El sistema de control de versiones gratuito y de código abierto, Git, fue originalmente creado por Linus Torvalds en 2005. A diferencia de los sistemas centralizados de control de versiones anteriores, como SVN y CVS, Git adopta un enfoque distribuido con determinadas características:

#### Beneficios y Significado:

Git ha tenido un impacto significativo en el desarrollo de software y la cultura de la colaboración. Algunos de sus beneficios más destacados incluyen:

- **Historial completo de cambios:** Git proporciona un registro detallado de todos los cambios realizados en un proyecto, lo que facilita la auditoría y la reversión de cambios si es necesario.
- **Colaboración eficiente:** la naturaleza distribuida de Git permite a los desarrolladores trabajar de manera independiente y luego integrar sus cambios de manera fluida, lo que facilita la colaboración en equipos distribuidos.
- **Experimentación segura:** con ramas ligeras y fáciles de crear, Git fomenta la experimentación y el desarrollo iterativo de nuevas características sin afectar la estabilidad del proyecto principal.

- **Respaldo y restauración:** Git permite a los desarrolladores realizar copias de seguridad regulares del repositorio y restaurar versiones anteriores en caso de pérdida de datos o errores críticos.

Desde su creación en 2005, Git ha pasado de ser una herramienta utilizada principalmente para el desarrollo del kernel de Linux a convertirse en el estándar de facto para el control de versiones en la industria del software. Su impacto en la eficiencia, la colaboración y la gestión de proyectos es innegable, y su legado continuará influyendo en la forma en que se desarrolla el software en el futuro.

## Configuración:

### Configurar nuestra identidad global en Git:

Para configurar tu identidad global en Git, incluyendo tu nombre de usuario y dirección de correo electrónico, sigue estos pasos:

- **git config --global user.name "Tu Nombre"**
- **git config --global user.email tu@email.com**

**Verificar la configuración:** Puedes verificar que la configuración se haya realizado correctamente ejecutando los siguientes comandos:

- **git config --global user.name**
- **git config --global user.email**

### Operaciones básicas de gestión de archivos y carpetas:

Antes de inicializar un repositorio Git, podemos realizar operaciones básicas en tu sistema de archivos. Algunos comandos comunes necesarios antes de crear un repositorio:

- Elección de disco local: **cd (nombre del disco):**
- Entrar a un directorio: **cd (nombre del directorio)**
- Retroceder de directorio: **cd ..**
- Crear una carpeta: **mkdir (nombre de la carpeta)**
- Eliminar una carpeta: **rmdir (nombre de la carpeta)**
- Leer un archivo: **ls**
- Crear un archivo en GitBash: **touch (nombre del archivo.extensión)**

### Inicializar un repositorio Git:

Para comenzar a utilizar Git en un directorio existente, primero debemos inicializar un repositorio:

- **git init**

### Cambiar el nombre de la rama principal:

Este paso es opcional, pero si deseamos cambiar el nombre de la rama principal a "main", puedes usar el siguiente comando:

- **git branch -M main**

## Agregar un control remoto:

A continuación, necesitamos agregar un control remoto a tu repositorio local. Esto generalmente se hace para vincular tu repositorio local con un repositorio remoto, como uno alojado en GitHub. Puedes hacerlo con el comando:

- **git remote add origin** <https://github.com/RodrigoStranger/Trabajos-Universidad>

## Ejemplo:

```
riexd@DESKTOP-LITKB10 MINGW64 /c
$ git config --global user.name "Rodrigo Infanzon Acosta"

riexd@DESKTOP-LITKB10 MINGW64 /c
$ git config --global user.email rinfanzona@ulasalle.edu.pe

riexd@DESKTOP-LITKB10 MINGW64 /c
$ mkdir prueba

riexd@DESKTOP-LITKB10 MINGW64 /c
$ cd prueba

riexd@DESKTOP-LITKB10 MINGW64 /c/prueba
$ git init
Initialized empty Git repository in C:/prueba/.git/

riexd@DESKTOP-LITKB10 MINGW64 /c/prueba (master)
$ git branch -M main

riexd@DESKTOP-LITKB10 MINGW64 /c/prueba (main)
$ git remote add origin https://github.com/RodrigoStranger/Trabajos-Universidad
```

Imagen con propiedad del autor ©Rodrigo Infanzón Acosta, ©Git

## Agregar archivos al área de preparación:

Antes de hacer un commit, necesitamos agregar los archivos que deseas incluir en el commit al área de preparación o staging area. Puedes hacer esto utilizando el comando git add. Por ejemplo, podemos usar:

- Agregar todos los archivos: **git add .**
- Agregar un archivo: **git add (nombre del archivo.extensión)**
- Eliminar un archivo del area de preparación: **git rm --cached (nombre del archivo)**

## Hacer un commit:

Después de agregar los archivos al área de trabajo, estás listo para hacer un commit. Un commit en Git representa un conjunto de cambios que se van a guardar en el historial del repositorio. Puedes hacer un commit con el comando:

- **git commit -m "mensaje de confirmación del commit"**
- Hacer un commit sin hacer un git add: **git commit -a**

## Empujar cambios al repositorio remoto:

Una vez que has agregado el control remoto, puedes empujar tus cambios locales al repositorio remoto usando el comando git push. Para establecer la rama remota de seguimiento y empujar los cambios, puedes usar la opción -u (upstream). Por ejemplo:

- **git push -u origin main**

**Resumen:**

1. Crear nuestro directorio local.
2. Inicializamos nuestro repositorio.
3. Agregamos un control remoto en el servidor GitHub.

**En el desarrollo de trabajos:**

4. Agregamos nuestros archivos al area de preparación.
5. Hacemos uso del commit.
6. Empujamos nuestro desarrollo al servidor de GitHub.

**Actualización de repositorio: del servidor hacia el área de trabajo**

Para actualizar un repositorio local con los cambios más recientes del repositorio remoto, puedes utilizar el comando:

- `git pull`

Este comando extrae los cambios del repositorio remoto y los fusiona con tu rama local.

**Git +****Eliminar un archivo de nuestro repositorio:**

- `rm` (nombre del archivo en el repo)
- `git add` (nombre del archivo en el repo)
- `git commit -m "mensaje que ha eliminado un archivo, explica" -a`

**Cambiar nombre de un archivo**

- `git mv` (nombre del archivo) (nombre del nuevo archivo)
- `git add` (nombre del archivo en el repo)
- `git commit -m "mensaje que ha cambiado el nombre de un archivo, explica" -a`

**Ver estado del Area de Preparación:**

- `git status`
- `git status -s`

**Crear un gitignore:**

- `touch .gitignore`
- Agregar excepciones
- `git add .gitignore`
- `git commit -m "mensaje"`

**Hacer una comparación entre commits:**

- Mostrar identificadores: `git log`
- Mostrar identificadores, en resumen: `git log --oneline`
- Realizar la comparación: `git diff` (identificador1) (identificador2)

## Ramas en Git:

En Git, las ramas son una característica fundamental que permite a los desarrolladores trabajar en diferentes versiones del proyecto de forma independiente. Cada rama representa una línea de desarrollo separada que puede tener su propio conjunto de cambios, historial de commits y archivos.



Imagen con propiedad del autor ©MEDIUM (1)

## Funcionamiento de las ramas:

Cuando creas un repositorio Git, se crea automáticamente una rama principal llamada "master" (o "main", dependiendo de la configuración). Esta rama es la línea principal de desarrollo y generalmente refleja la versión estable del proyecto.

Sin embargo, puedes crear nuevas ramas para trabajar en nuevas características, experimentar o solucionar problemas sin afectar la rama principal. Cuando creas una nueva rama, esta comienza como una copia exacta de la rama desde la cual se creó. A medida que realizas cambios y commits en una rama, esos cambios permanecen en esa rama hasta que decides fusionarlos con otra rama. Aquí comparto los comandos para la creación de ramas:

- Crear una rama: `git branch (nombre de la rama)`
- Trasladarse de una rama a otra: `git switch (nombre de la rama)`
- Fusionar una rama con la rama master: `git merge (nombre de la rama)`

```
PF@PC-MASTER MINGW64 ~/Documents/proyectos_git/proyecto_1 (master)
$ git merge rama_imagenes
Updating fcc21d6..ac8a646
Fast-forward
 img/camping.jpg | Bin 0 -> 52502 bytes
 img/city.jpg    | Bin 0 -> 83171 bytes
 img/metro.jpg   | Bin 0 -> 114286 bytes
 index.html     | 9 ++++++--
 js/test.js      | 1 -
 5 files changed, 8 insertions(+), 2 deletions(-)
 create mode 100644 img/camping.jpg
 create mode 100644 img/city.jpg
 create mode 100644 img/metro.jpg
 delete mode 100644 js/test.js
```

Imagen con propiedad del autor ©PROGRAMACION FÁCIL (2)

## GitHub:

GitHub es una plataforma de desarrollo de software basada en la web que ofrece servicios de alojamiento de repositorios Git, herramientas de control de versiones y una variedad de características colaborativas para desarrolladores. Fue fundada en 2008 por Chris Wanstrath, PJ Hyett y Tom Preston-Werner, desde entonces, se ha convertido en una de las herramientas más populares para el desarrollo de software colaborativo.



### Características Principales:

- **Repositorios Git:** GitHub proporciona alojamiento de repositorios Git, lo que permite a los desarrolladores almacenar, gestionar y colaborar en proyectos de software utilizando el sistema de control de versiones Git. Cada proyecto alojado en GitHub tiene su propio repositorio, que contiene el historial completo de cambios realizados en el código.
- **Control de Versiones:** GitHub facilita el seguimiento de las diferentes versiones de un proyecto de software a lo largo del tiempo. Los desarrolladores pueden realizar commits para guardar cambios en el repositorio, crear ramas para trabajar en características o correcciones de errores específicas, y fusionar ramas para combinar los cambios de forma segura.
- **Colaboración:** GitHub ofrece una variedad de características diseñadas para facilitar la colaboración entre desarrolladores. Esto incluye la capacidad de revisar y comentar el código, realizar solicitudes de extracción (pull requests) para proponer cambios en el código base, y discutir problemas y solicitudes de funciones en un tablero de problemas integrado.
- **Integraciones y Herramientas:** GitHub se integra con una amplia gama de herramientas y servicios, lo que facilita la integración continua, la entrega continua y otras prácticas de desarrollo ágil. También ofrece características como wikis, páginas de proyectos y un sistema de seguimiento de problemas para ayudar a los equipos a organizar y documentar su trabajo.
- **Impacto y Popularidad:** GitHub ha tenido un impacto significativo en la forma en que se desarrolla el software en la actualidad. Ha democratizado el proceso de desarrollo de software al proporcionar herramientas poderosas y accesibles para que los desarrolladores trabajen juntos de manera colaborativa, independientemente de su ubicación geográfica.

GitHub se ha convertido en la plataforma de elección para millones de desarrolladores y organizaciones de todo el mundo, incluidas algunas de las empresas más grandes y exitosas del sector tecnológico. Su popularidad se debe en parte a su facilidad de uso, su amplia gama de características y su sólida comunidad de desarrolladores.

### Solución y explicación de la creación de un tablero M x N

Durante el primer laboratorio de programación impartido por el docente, se nos solicitó desarrollar un programa en cualquier lenguaje de programación, en este caso se usó C++, dicho programa debe ser capaz de imprimir un tablero en la consola, con la particularidad que se parezca a un tablero de ajedrez.

#### Lógica:

Para cumplir con el requerimiento del profesor, me sumergí en la lógica detrás de la creación de una matriz bidimensional que representaría nuestro tablero. Con el conocimiento adquirido en clase, empleé dos bucles “for” anidados para inicializar y recorrer esta matriz.

#### Mi lógica posee:

- Uso de una función.
- Uso de vectores.
- Uso de bucles anidados “for”.
- Programación básica.



**Codificación:**

```

1 #include <iostream>
2 #include <vector>
3 using namespace std;
4 void ImprimirTablero(int filas, int columnas) {
5     vector<vector<char>> > tablero(filas, vector<char>(columnas, ' '));
6     for (int i = 0; i < filas; ++i) {
7         for (int k = 0; k < columnas; ++k) {
8             cout << "----";
9         }
10        cout << "-" << endl;
11        for (int j = 0; j < columnas; ++j) {
12            cout << "|" << tablero[i][j] << " ";
13        }
14        cout << "|" << endl;
15    }
16    for (int k = 0; k < columnas; ++k) {
17        cout << "----";
18    }
19    cout << "-" << endl;
20 }
21 int main() {
22     ImprimirTablero(5, 5);
23     return 0;
24 }

```

Imagen con propiedad del autor ©Rodrigo Infanzón Acosta, ©GDB Online Compiler

**Explicación:**

1. Defino una función llamada **ImprimirTablero** que toma dos parámetros: **filas** y **columnas**, que representan las dimensiones del tablero a imprimir.
2. Creo una matriz bidimensional llamada **tablero**, utilizando un vector de vectores de caracteres (**vector<vector<char>>**). Cada celda de la matriz se inicializa con un espacio en blanco (' ').
3. Utilizo un bucle for externo para iterar sobre las filas del tablero. Dentro de este bucle, hay dos bucles anidados: uno para imprimir las líneas horizontales que separan las filas (---) y otro para imprimir las celdas del tablero.
4. Dentro del bucle externo, después de imprimir las líneas horizontales, hay otro bucle for interno que se encarga de imprimir las celdas del tablero. Cada celda se imprime dentro de un borde vertical (|), seguido del contenido de la celda y otro borde vertical (|). El contenido de cada celda se toma de la matriz tablero.
5. Después de imprimir todas las celdas en una fila, se imprime un borde vertical (|) al final de la fila y se inserta un salto de línea (endl) para pasar a la siguiente fila.
6. Finalmente, después de imprimir todas las filas, se imprime otra línea horizontal (---) para representar el borde inferior del tablero, finalizando con un salto de línea.
7. Ejecuto el programa en **el int main**, llamando a la función **ImprimirTablero(n,m)**.

### Ejecución de ejemplo ImprimirTablero (5,5):

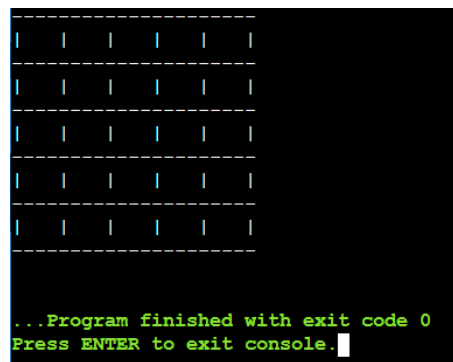


Imagen con propiedad del autor ©Rodrigo Infanzón Acosta. ©GDB Online Compiler

Esta función imprime un tablero de tamaño especificado en la consola, utilizando caracteres para representar las celdas del tablero y líneas horizontales y verticales para los bordes.

### Observaciones:

- No se puede evidenciar el uso de Git (agregar al area de preparación, hacer un commit, y empujar al servidor de GitHub) ya que esos procedimientos se realizaron en el primer laboratorio y fue calificado. Sin embargo, los commits ya están establecidos en el repositorio lp3-24a.

## Actualización del tablero elaborado: crear una diagonal usando el primer apellido del estudiante

Después de elaborar, añadir al area de preparación, hacer el commit y empujar el programa elaborado “tablero.cpp”, el docente nos solicitó una modificación al programa, en este caso me tocó crear una diagonal usando mi primero apellido “INFANZON”, por lo cual lo elabore de la siguiente manera:

### Lógica:

Para cumplir con el nuevo requerimiento del profesor, actualicé la función **ImprimirTablero**, empleando en uso del tipo de dato “char”, con la finalidad de representar cada letra de mi apellido “INFANZON”. Y finalmente utilizar bucles para imprimir las letras, espacios en blanco y la estructura final del tablero.

### Mi lógica posee:

- Uso de una función.
- Uso de vectores.
- Uso de bucles anidados “for”.
- Programación básica.

### Codificación:

```
1 #include <iostream>
2 #include <vector>
3 using namespace std;
4 void ImprimirTablero() {
5     char l1 = 'I', l2 = 'N', l3 = 'F', l4 = 'A', l5 = 'N', l6 = 'Z', l7 = 'O', l8 = 'N';
6     char tablero[8][8] = {
7         {l1, l2, l3, l4, l5, l6, l7, l8},
8         {l1, l2, l3, l4, l5, l6, l7, l8},
9         {l1, l2, l3, l4, l5, l6, l7, l8},
10        {l1, l2, l3, l4, l5, l6, l7, l8},
11        {l1, l2, l3, l4, l5, l6, l7, l8},
12        {l1, l2, l3, l4, l5, l6, l7, l8},
13        {l1, l2, l3, l4, l5, l6, l7, l8},
14        {l1, l2, l3, l4, l5, l6, l7, l8}
15    };
16    cout << "-----" << endl;
17    for (int i = 0; i < 8; i++) {
18        cout << "| ";
19        for (int j = 0; j < 8; j++) {
20            cout << tablero[i][j] << " | ";
21        }
22        cout << endl << "-----" << endl;
23    }
24 }
25
26 int main() {
27     ImprimirTablero();
28     return 0;
29 }
```

Imagen con propiedad del autor ©Rodrigo Infanzón Acosta, ©GDB Online Compiler

### Explicación:

1. Defino la función llamada **ImprimirTablero** que no toma ningún argumento y no devuelve ningún valor.
2. Declaro las variables de tipo “char”, en las cuales se almacena cada letra de mi apellido.
3. Creo una matriz **tablero** de 8x8 donde cada elemento es una letra de mi apellido “INFANZON” en la diagonal principal y espacios en blanco en el resto de la matriz.
4. Imprimo una línea horizontal para representar el borde superior del tablero.

5. Utilizo un bucle anidado para imprimir cada elemento de la matriz **tablero**. Cada elemento se imprime dentro de un borde vertical "|", seguido de un espacio. Al final de cada fila, se imprime una línea horizontal para representar los bordes inferiores de las celdas del tablero.
6. Ejecuto el programa en el **int main**, llamando a la función **ImprimirTablero()**.

### Ejecución de ejemplo ImprimirTablero ():

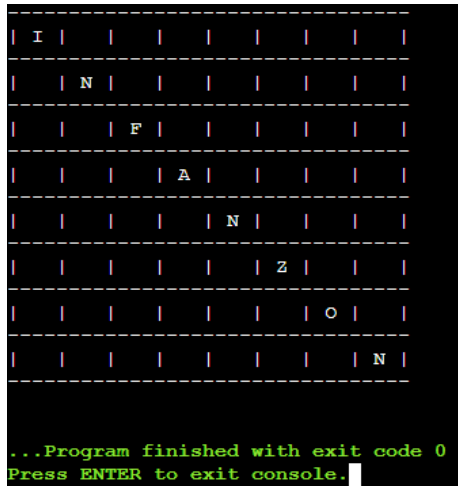


Imagen con propiedad del autor ©Rodrigo Infanzón Acosta, ©GDB Online Compiler

Esta función imprime un tablero 8x8, utilizando cada "char" declarado en el programa usado en la diagonal del tablero, utilizando caracteres para representar las celdas del tablero y líneas horizontales y verticales para los bordes.

### Observaciones:

- No se puede evidenciar el uso de Git (agregar al área de preparación, hacer un commit, y empujar al servidor de GitHub) ya que esos procedimientos se realizaron en el primer laboratorio y fue calificado. Sin embargo, los commits ya están establecidos en el repositorio lp3-24a.

## Solución y explicación del problema en OmegaUp: "El perrito que quiere un hueso" usando Git y GitHub

Estás paseando a tu perrito y acaban de pasar enfrente de una tienda de comida para perros. En la vitrina hay dos huesos de olor y tamaño distintos. Tu perro mentalmente le asignó una calificación del 1 al 10 al olor de cada hueso y una calificación del 1 al 10 al tamaño de cada hueso. Por supuesto, tu perro preferiría que le compraras el hueso que es simultáneamente el más grande y el que huele mejor. ¿Puedes ayudarlo a determinar qué hueso comprar?

**Entrada:** cuatro enteros: **L1, T1, L2, T2**, que son el olor y el tamaño de los dos huesos, respectivamente. Cada hueso se da en su respectiva línea.

**Salida:** El mensaje: "**Hueso 1**" o "**Hueso 2**" dependiendo de qué hueso es el que es simultáneamente el más grande y el que huele mejor. En caso de que ninguno de los dos huesos cumpla esta propiedad, imprimir el mensaje: "**Perrito confundido :(**".

### Ejemplo:

Entrada	Salida
8 9 5 8	Hueso 1
3 6 7 10	Hueso 2
6 9 8 4	Perrito confundido :(

Imagen con propiedad del autor ©omegaUp

### Solución:

Analizando el problema, se observa que el perrito en el primer caso elige el hueso 1 porque tanto el olor y su tamaño son superiores numéricamente al hueso 2. De la misma manera, en el segundo caso, el hueso 2 es superior numéricamente en olor y tamaño al hueso 1, por lo que el perrito termina eligiéndolo. Sin embargo, en el tercer caso se observa que el olor del hueso 2 es superior al olor del hueso 1 y el tamaño del hueso 1 es superior al hueso 2, por ende, el perro está confundido.

### Conclusión:

Como observamos, el perrito prefiere un hueso que posea el máximo valor en olor como en tamaño, si alguno de estos valores no es superior en la comparación, el perrito se confunde.

### Diseño del algoritmo:

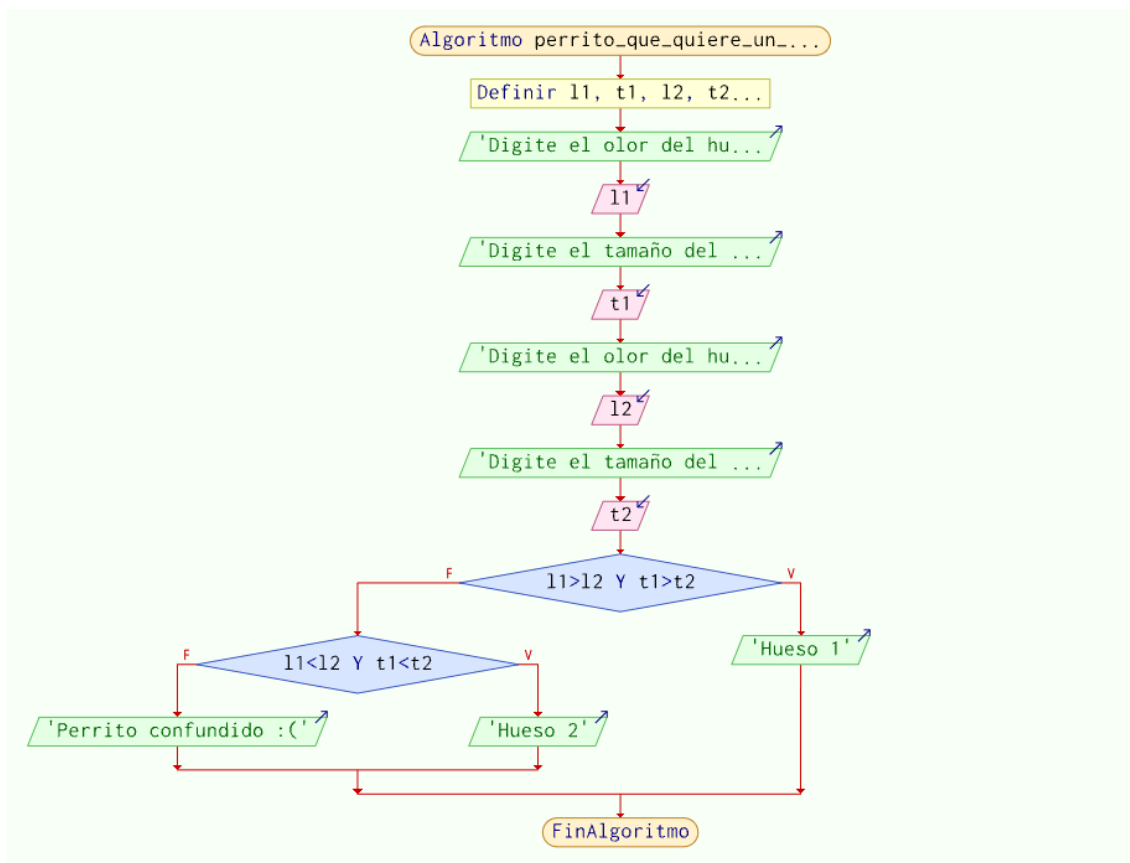


Imagen con propiedad del autor ©Rodrigo Infanzón Acosta, ©PSeInt

## Explicación:

1. Primeramente, declaramos las variables l1, t1, l2, t2 como tipo de dato entero.
2. Creamos una condicional anidada de otra, con el propósito de realizar una comparación lineal de dos huesos, con su respectivo olor y tamaño.
3. En la primer condicional, si el olor y tamaño del hueso 1 supera al olor y tamaño del hueso 2, entonces, el perrito elegirá el hueso 1
4. Si esta premisa no se cumple, entra hacia la segunda condicional.
5. En la segunda condicional, si el olor y tamaño del hueso 2 supera al olor y tamaño del hueso 1, entonces, el perrito elegirá el hueso 2.
6. Si esta premisa no se cumple, significa que el olor del hueso 2 es mayor al hueso 1, pero el tamaño del hueso 2 es inferior al tamaño del hueso 1 o viceversa. Por lo tanto, el perrito esta confundido.

## Codificación:

Una vez diseñado nuestra lógica resolviendo el problema, podemos iniciar con la codificación del programa. Para ello, debo ubicarme en mi directorio: c:/lp3-24a/lab01/exercises/ y crear un archivo denominado "hueso.cpp", en dicho archivo se debe de codificar el programa para posteriormente:

- Añadirlo al área de preparación.
- Hacer un commit del archivo.
- Empujar el archivo al servidor GitHub.

Posteriormente abrir el archivo y codificar:

```
riexd@DESKTOP-LITKB10 MINGW64 /
$ cd c:

riexd@DESKTOP-LITKB10 MINGW64 /c
$ cd lp3-24a

riexd@DESKTOP-LITKB10 MINGW64 /c/lp3-24a (main)
$ cd lab01

riexd@DESKTOP-LITKB10 MINGW64 /c/lp3-24a/lab01 (main)
$ cd exercises

riexd@DESKTOP-LITKB10 MINGW64 /c/lp3-24a/lab01/exercises (main)
$ touch hueso.cpp

riexd@DESKTOP-LITKB10 MINGW64 /c/lp3-24a/lab01/exercises (main)
$ notepad hueso.cpp
```

Imagen con propiedad del autor ©Rodrigo Infanzón Acosta, ©Git

## En código:

```
1 #include <iostream>
2
3 using namespace std;
4
5 int main() {
6     int l1,t1,l2,t2 = 0;
7     cout<<"Digite el olor del hueso 1: ";
8     cin>>l1;
9     cout<<"Digite el tamaño del hueso 1: ";
10    cin>>t1;
11    cout<<"Digite el olor del hueso 2: ";
12    cin>>l2;
13    cout<<"Digite el tamaño del hueso 2: ";
14    cin>>t2;
15    if(l1>l2 && t1>t2) {
16        cout<<"Hueso 1";
17    } else if(l1<l2 && t1<t2) {
18        cout<<"Hueso 2";
19    } else {
20        cout<<"Perrito confundido :(";
21    }
22    return 0;
23 }
```

Imagen con propiedad del autor ©Rodrigo Infanzón Acosta, ©GDB Online Compiler

## Ejecución del código caso 1:

```
Digite el olor del hueso 1: 8
Digite el tamaño del hueso 1: 9
Digite el olor del hueso 2: 5
Digite el tamaño del hueso 2: 8
Hueso 1

...Program finished with exit code 0
Press ENTER to exit console.
```

Imagen con propiedad del autor ©Rodrigo Infanzón Acosta, ©GDB Online Compiler

## Ejecución del código caso 2:

```
Digite el olor del hueso 1: 3
Digite el tamaño del hueso 1: 6
Digite el olor del hueso 2: 7
Digite el tamaño del hueso 2: 10
Hueso 2

...Program finished with exit code 0
Press ENTER to exit console.
```

Imagen con propiedad del autor ©Rodrigo Infanzón Acosta, ©GDB Online Compiler

## Ejecución del código caso 3:

```
Digite el olor del hueso 1: 6
Digite el tamaño del hueso 1: 9
Digite el olor del hueso 2: 8
Digite el tamaño del hueso 2: 4
Perrito confundido :(

...Program finished with exit code 0
Press ENTER to exit console.
```

Imagen con propiedad del autor ©Rodrigo Infanzón Acosta, ©GDB Online Compiler

## Añadiendo el archivo a nuestro repositorio: local y servidor

1. Una vez codificado, guardamos el archivo "hueso.cpp".
2. En el GitBash o Consola, nos ubicamos en nuestro directorio c:/lp3-24a/lab01/exercises/
3. Usamos el comando git add hueso.cpp
4. Hacemos el commit respectivo: git commit -m "mensaje"
5. Empujamos nuestro archivo al servidor: git push origin main

```
filedBESKTOP-LITK810 MINGW64 /c/lp3-24a/lab01/exercises (main)
$ git add hueso.cpp

filedBESKTOP-LITK810 MINGW64 /c/lp3-24a/lab01/exercises (main)
$ git status
On branch main
Your branch is up to date with 'origin/main'.

Changes to be committed:
  (use "git restore --staged <file>..." to unstage)
    new file:   hueso.cpp

Changes not staged for commit:
  (use "git add <file>..." to update what will be committed)
  (use "git restore <file>..." to discard changes in working directory)
    modified:   ../report/report01.docx

Untracked files:
  (use "git add <file>..." to include in what will be committed)
    ../report/-perito.docx
    ../report/-ABL0468.tmp

filedBESKTOP-LITK810 MINGW64 /c/lp3-24a/lab01/exercises (main)
$ git commit -m "Agregando el codigo del problema del perrito"
[main 5a6d734] Agregando el codigo del problema del perrito
1 file changed, 23 insertions(+)
 create mode 100644 lab01/exercises/hueso.cpp

filedBESKTOP-LITK810 MINGW64 /c/lp3-24a/lab01/exercises (main)
$ git push origin main
git: 'credential-cache' is not a git command. See 'git --help'.
Enumerating objects: 8, done.
Counting objects: 100% (8/8), done.
Delta compression using up to 8 threads
Compressing objects: 100% (5/5), done.
Writing objects: 100% (5/5), 636 bytes | 328.00 KiB/s, done.
Total 5 (delta 1), reused 0 (delta 0), pack-reused 0 (from 0)
remote: Resolving deltas: 100% (1/1), completed with 1 local object.
To https://github.com/RodrigoStranger/lp3-24a
 4a0ce3f..5a6d734 main -> main
```

Imagen con propiedad del autor ©Rodrigo Infanzón Acosta, ©Git

## Observaciones:

1. Se puede optimizar el código elaborado implementándolo en una función, para ello tenemos que: modificar el código fuente, añadirlo al área de preparación, hacer un commit y empujar al servidor de GitHub:

## Modificación:

```
1 #include <iostream>
2
3 using namespace std;
4
5 string decisionPerrito(int o1, int t1, int o2, int t2) {
6     if (o1>o2 && t1>t2) {
7         return "Hueso 1";
8     } else if (o1<o2 && t1<t2) {
9         return "Hueso 2";
10    } else {
11        return "Perrito confundido :(";
12    }
13 }
14
15 int main() {
16     int l1, t1, l2, t2;
17     cout<<"Digite el olor del hueso 1: ";
18     cin>>l1;
19     cout<<"Digite el tamaño del hueso 1: ";
20     cin>>t1;
21     cout<<"Digite el olor del hueso 2: ";
22     cin>>l2;
23     cout<<"Digite el tamaño del hueso 2: ";
24     cin>>t2;
25     cout<<"El perrito eligió: "<<decisionPerrito(l1, t1, l2, t2)<<endl;
26     return 0;
27 }
```

Imagen con propiedad del autor ©Rodrigo Infanzón Acosta, ©GDB Online Compiler

## Agregando la modificación al servidor GitHub:

1. Una vez modificado, guardamos el archivo "hueso.cpp".
2. En el GitBash o Consola, nos ubicamos en nuestro directorio c:/lp3-24a/lab01/exercises/
3. Usamos el comando git add hueso.cpp
4. Hacemos el commit respectivo: git commit -m "mensaje de edición"
5. Empujamos nuestro archivo al servidor: git push origin main

```
C:\lp3-24a\lab01\exercises>git add hueso.cpp
C:\lp3-24a\lab01\exercises>git commit -m "Modificando hueso.cpp, agregando funcion"
[main 92f2d94] Modificando hueso.cpp, agregando funcion
1 file changed, 21 insertions(+), 17 deletions(-)
C:\lp3-24a\lab01\exercises>git push origin main
git: 'credential-000cache' is not a git command. See 'git --help'.
Enumerating objects: 9, done.
Counting objects: 100% (9/9), done.
Delta compression using up to 8 threads
Compressing objects: 100% (5/5), done.
Writing objects: 100% (5/5), 703 bytes | 234.00 KiB/s, done.
Total 5 (delta 1), reused 0 (delta 0), pack-reused 0 (from 0)
remote: Resolving deltas: 100% (1/1), completed with 1 local object.
To https://github.com/RodrigoStranger/lp3-24a
 5a6d734..92f2d94  main -> main
C:\lp3-24a\lab01\exercises>
```

Imagen con propiedad del autor ©Rodrigo Infanzón Acosta, ©Windows

## Estructura del laboratorio 01:

```
|--- lab01 [DIRECTORIO]
|   |--- exercises [DIRECTORIO]
|       |--- hueso.cpp
|       |--- tablero.cpp
|       |--- report [DIRECTORIO]
|           |--- report01.dox
|           |--- report01.pdf
```

3 directorios, 4 archivos

Imagen con propiedad del autor ©Rodrigo Infanzón Acosta, ©Notepad



## Aportes del laboratorio hacia el estudiante:

1. **Aplicación práctica de conceptos teóricos:** durante el laboratorio 01, pude aplicar y consolidar los conceptos teóricos aprendidos en clase. La oportunidad de trabajar directamente con el código y resolver problemas en tiempo real me permitió entender mejor los conceptos abstractos y ver cómo se aplican en situaciones reales.
2. **Dominio de control de versiones:** el laboratorio 01 me proporcionó una comprensión profunda de los conceptos fundamentales de control de versiones, especialmente en el contexto de Git. En este contexto, aprendí cómo usar Git para gestionar eficazmente el historial de cambios en proyectos de software, lo que incluye la creación de repositorios, la realización de commits, la creación de ramas y la fusión de cambios.
3. **Adquisición de un contexto histórico acerca de la programación:** el laboratorio 01, tuve la oportunidad de adquirir un contexto histórico sobre la programación, lo cual fue fundamental para comprender cómo ha evolucionado esta disciplina a lo largo del tiempo.

## Referencias:

1. [https://www.onlinegdb.com/online\\_c++\\_compiler](https://www.onlinegdb.com/online_c++_compiler)
2. <https://github.com/>
3. <https://pseint.sourceforge.net/>
4. <https://omegaup.com/>
5. <https://git-scm.com/>

## Referencias Bibliográficas:

1. Carrillo, A. (2020). *Git Branch y Merge de cero a crack*. Medium. Recuperado de: [https://medium.com/@alfonsocarrillo\\_dev/git-branch-y-merge-de-cero-a-crack-80d653b05f80](https://medium.com/@alfonsocarrillo_dev/git-branch-y-merge-de-cero-a-crack-80d653b05f80)
2. Programación Fácil (2024). *git merge - Fusión de ramas*. Recuperado de: [https://www.programacionfacil.org/cursos/git\\_github/capitulo\\_7\\_git\\_merge\\_fusion\\_de\\_ramas.html](https://www.programacionfacil.org/cursos/git_github/capitulo_7_git_merge_fusion_de_ramas.html)
3. Pérez, J. (2023). *Informe de laboratorio 01*. Recuperado de: [https://drive.google.com/drive/folders/14FJERSs59gl756stFEtry\\_vY-1AgAxAO](https://drive.google.com/drive/folders/14FJERSs59gl756stFEtry_vY-1AgAxAO)
4. Scott, C. & Straub, B. (2009). *Pro Git*. Apress. Recuperado de: <https://drive.google.com/file/d/1LCdXfgUYjdSiCB15KJJNl7dobdFj9DGz/view>
5. Soy Dalto. (4 de febrero del 2024). *Curso de GIT desde CERO (Completo)* [Video]. YouTube. [https://www.youtube.com/watch?v=9Zj-K-zk\\_Go&t=11598s](https://www.youtube.com/watch?v=9Zj-K-zk_Go&t=11598s)

## Calificación:

### Rúbrica para el contenido de informes y evidencias

Contenido y demostración		Puntos	Checklist	Estudiante	Profesor
<b>GitHub</b>	El repositorio se pudo clonar y se evidencia la estructura adecuada para revisar los entregables. (Se descontará puntos por error o observación)	4	X	4	
<b>Commits</b>	Hay porciones de código fuente asociado a los commits planificados con explicaciones detalladas. (El profesor puede preguntar para refrendar calificación).	4	X	4	
<b>Ejecución</b>	Se incluyen comandos para ejecuciones y pruebas del código fuente explicadas gradualmente que permitirán replicar el proyecto. (Se descontará puntos por cada omisión)	4	X	4	
<b>Pregunta</b>	Se responde con completitud a la pregunta formulada en la tarea. (El profesor puede preguntar para refrendar calificación).	2	X	2	
<b>Ortografía</b>	El documento no muestra errores ortográficos. (Se descontará puntos por error encontrado)	2	X	2	
<b>Madurez</b>	El informe muestra de manera general una evolución de la madurez del código fuente con explicaciones puntuales pero precisas, agregando diagramas generados a partir del código fuente y refleja un acabado impecable. (El profesor puede preguntar para refrendar calificación).	4	X	4	
Total		20		20	