

**Evaluación Parcial
Producto Académico n.º 2
Actividad Grupal**

RAMIREZ LAGOS ROXANA

CODIGO 40956229@CONTINENTAL.EDU.PE

INGENIERIA DE SISTEMAS

2025

1. Consideraciones:

Nombre de la Actividad Evaluativa: “Modelado de Componentes y Patrones Arquitectónicos”

Descripción de la Actividad:

Los estudiantes conformarán equipos de 3 a 5 integrantes para desarrollar:

- El Modelo de Componentes con sus interfaces e interacciones del proyecto a desarrollar.
- Integrar un patrón arquitectónico adecuado al proyecto a desarrollar

Indicaciones para el Desarrollo:

1. Formación de Equipos: Organizarse en grupos de 3 a 5 integrantes, asignando un líder del equipo para la coordinación del proyecto.
2. Definición del Proyecto: Seleccionar un problema a resolver con software y definir los objetivos del proyecto.
3. Creación del Repositorio: Configurar un repositorio en GitHub con una estructura inicial adecuada.
4. Aplicación de GitFlow: Implementar GitFlow asegurando la correcta administración de ramas (main, develop, feature, hotfix, release), utilizando pull requests para integrar cambios.
5. Gestión de Commits: Realizar commits frecuentes, bien documentados y siguiendo una convención clara de mensajes.
6. Automatización con CI/CD: Configurar herramientas de integración y entrega continua (CI/CD) para la automatización de pruebas y despliegue.
7. Resolución de Conflictos: Manejar conflictos de manera efectiva mediante revisiones de código y pruebas antes de la fusión de ramas.
8. Documentación y Reporte: Elaborar un informe técnico describiendo el uso de Git y GitHub, la aplicación de GitFlow, la automatización de procesos y la gestión de conflictos.
9. Entrega del Proyecto: Subir el código fuente en GitHub y enviar el informe técnico en formato .doc, respetando el formato de presentación indicado.

- **PLANTEAMIENTO Y DESCRIPCIÓN DETALLADA DE LA ACTIVIDAD:**
 - Los equipos desarrollarán un proyecto de software con control de versiones implementado mediante Git y GitHub.
 - Durante el desarrollo deberán aplicar buenas prácticas de gestión de código, utilizar GitFlow para la estructuración del trabajo en equipo, automatizar procesos con herramientas de CI/CD y documentar adecuadamente el proceso de desarrollo.
- **Cada equipo deberá presentar:**
 1. Un repositorio en GitHub con el código fuente gestionado adecuadamente.
 2. Un informe técnico que describa la aplicación de GitFlow, la automatización de procesos, el manejo de conflictos y las mejores prácticas implementadas.

DESARROLLO:

Informe Técnico: Gestión de Versiones y Automatización en Proyecto de Registro de Socios

1. Introducción

Este informe documenta el uso de herramientas de control de versiones (Git y GitHub), la aplicación de la metodología GitFlow, la automatización de procesos y la gestión de conflictos en el desarrollo de una aplicación en C++ para el registro y reporte de pagos de socios en una organización deportiva.

El código fuente implementa un sistema interactivo que permite registrar pagos, calcular montos con IGV, y generar reportes estadísticos según tipo de socio, edad, forma de pago y número de hijos.

2. Uso de Git y GitHub

Repositorio:

El proyecto se gestiona en GitHub, permitiendo colaboración remota, seguimiento de cambios y control de versiones.

Comandos utilizados:

```
git init
git clone <url-del-repositorio>
git add .
git commit -m "Implementación de lógica de registro de pagos"
git push origin develop
```

Ventajas:

- Historial completo de cambios.
- Revisión colaborativa mediante Pull Requests.
- Integración con herramientas de automatización.

3. Aplicación de GitFlow

Estructura de ramas:

- main: versión estable del sistema.
- develop: rama de integración de nuevas funcionalidades.
- feature/registro-pagos: desarrollo de la lógica de registro.
- feature/reportes: implementación de reportes estadísticos.
- hotfix/validaciones: corrección de errores en validaciones de entrada.

Ejemplo de flujo:

```
git checkout develop git checkout -b feature/reportes # Desarrollo... git add . git commit -m "Agrega generación de reportes estadísticos" git checkout develop git merge feature/reportes git push origin develop
```

Beneficio:

GitFlow permite una organización clara del ciclo de vida del desarrollo, facilitando la colaboración y el mantenimiento del código.

4. Automatización de Procesos

Herramientas:

- GitHub Actions para integración continua.
- Scripts de validación (pre-commit) para asegurar formato y compilación.

Ejemplo de workflow automatizado:

```
name: C++ Build and Test on: [push, pull_request] jobs: build: runs-on: ubuntu-latest steps: - uses: actions/checkout@v2 - name: Compilar código run: g++ -o registro registro.cpp - name: Ejecutar pruebas run: ./registro < datos_prueba.txt
```

Automatización aplicada:

- Compilación automática en cada push.
- Validación de entradas y salidas esperadas.
- Alertas en caso de errores de compilación o lógica.

5. Gestión de Conflictos

Conflictos comunes:

- Edición simultánea del bloque switch(opcion) por varios desarrolladores.
- Diferencias en validaciones de entrada (do-while).

Resolución:

- Uso de git merge y revisión manual de conflictos.
- Identificación de marcadores:

```
<<<<<<< HEAD // Versión local ===== // Versión remota >>>>>>> develop
```

Buenas prácticas:

- Comunicación clara entre colaboradores.
- Revisión de cambios antes del merge.
- Uso de herramientas como GitHub Desktop o VS Code para visualizar conflictos.

6. Pruebas antes de la fusión

Checklist de validación:

- El código compila sin errores.
- Las entradas inválidas son correctamente gestionadas.
- Los cálculos de IGV y montos son precisos.
- Los reportes reflejan correctamente los datos acumulados.

Prueba de funcionalidad:

- Registro de socio tipo 'A' con edad > 50 y sin hijos.
- Validación de cálculo de montoTotalVipMayores50 y montoAcumuladoSinHijos.

Comando para comparar ramas antes del merge:

git diff develop feature/reportes

Estructura del Proyecto en Dev-C++

1. **Archivo principal:**
Guarda el código que compartiste como registro.cpp.
2. **Carpeta del proyecto:**
Crea una carpeta llamada RegistroSocios y dentro coloca:
3. RegistroSocios/
 4. |— registro.cpp
 5. |— README.md
 6. |— .gitignore
 7. |— docs/
 8. |— informe-tecnico.md
9. **Archivo .gitignore** (para excluir archivos temporales de Dev-C++):
 10. *.o
 11. *.exe
 12. *.log
 13. *.obj

Configuración de Git y GitHub con GitFlow

1. **Inicializa el repositorio:**
2. git init

3. `git remote add origin https://github.com/tuusuario/RegistroSocios.git`
4. **Aplica GitFlow:**
5. `git checkout -b develop`
6. `git checkout -b feature/registro-pagos`
7. **Sube tu código:**
8. `git add registro.cpp`
9. `git commit -m "Implementación inicial del sistema de registro de pagos"`
10. `git push origin feature/registro-pagos`
11. **Fusión a develop:**
12. `git checkout develop`
13. `git merge feature/registro-pagos`
14. `git push origin develop`

GitFlow aplicado

- main: versión estable
- develop: integración
- feature/registro-pagos: lógica de pagos
- feature/reportes: generación de reportes

7. Conclusión

El uso de Git y GitHub ha permitido una gestión eficiente del código fuente, mientras que GitFlow ha facilitado la organización del desarrollo colaborativo. La automatización mediante GitHub Actions ha mejorado la calidad del software, reduciendo errores humanos. La gestión proactiva de conflictos y la validación previa a la fusión de ramas aseguran la estabilidad del sistema.

Este enfoque técnico garantiza un desarrollo sostenible, escalable y alineado con buenas prácticas de ingeniería de software.

```
git checkout develop
git checkout -b feature/reportes
# Desarrollo...
git add .
git commit -m "Agrega generación de reportes estadísticos"
git checkout develop
git merge feature/reportes
git push origin develop
```

Beneficio:

GitFlow permite una organización clara del ciclo de vida del desarrollo, facilitando la colaboración y el mantenimiento del código.

El uso de Git y GitHub ha permitido una gestión eficiente del código fuente, mientras que GitFlow ha facilitado la organización del desarrollo colaborativo. La automatización mediante GitHub Actions ha mejorado la calidad del software, reduciendo errores humanos. La gestión proactiva de conflictos y la validación previa a la fusión de ramas aseguran la estabilidad del sistema.

Este enfoque técnico garantiza un desarrollo sostenible, escalable y alineado con buenas prácticas de ingeniería de software.

Yaml

```
name: C++ Build and Test
on: [push, pull_request]
jobs:
  build:
    runs-on: ubuntu-latest
    steps:
      - uses: actions/checkout@v2
      - name: Compilar código
        run: g++ -o registro registro.cpp
      - name: Ejecutar pruebas
        run: ./registro < datos_prueba.txt
```

Automatización aplicada:

- Compilación automática en cada push.
- Validación de entradas y salidas esperadas.
- Alertas en caso de errores de compilación o lógica.

5. Gestión de Conflictos

Conflictos comunes:

- Edición simultánea del bloque switch(opcion) por varios desarrolladores.
- Diferencias en validaciones de entrada (do-while).

Resolución:

- Uso de git merge y revisión manual de conflictos.
- Identificación de marcadores:

<<<<<< HEAD // Versión local ===== // Versión remota >>>>>> develop

Buenas prácticas:

- Comunicación clara entre colaboradores.
- Revisión de cambios antes del merge.
- Uso de herramientas como GitHub Desktop o VS Code para visualizar conflictos.

6. Pruebas antes de la fusión

Checklist de validación:

- El código compila sin errores.
- Las entradas inválidas son correctamente gestionadas.
- Los cálculos de IGV y montos son precisos.
- Los reportes reflejan correctamente los datos acumulados.

Prueba de funcionalidad:

- Registro de socio tipo 'A' con edad > 50 y sin hijos.
- Validación de cálculo de montoTotalVipMayores50 y montoAcumuladoSinHijos.

Comando para comparar ramas antes del merge:

`git diff develop feature/reportes`

7. Conclusión

El uso de Git y GitHub ha permitido una gestión eficiente del código fuente, mientras que GitFlow ha facilitado la organización del desarrollo colaborativo. La automatización mediante GitHub Actions ha mejorado la calidad del software, reduciendo errores humanos. La gestión proactiva de conflictos y la validación previa a la fusión de ramas aseguran la estabilidad del sistema.

Este enfoque técnico garantiza un desarrollo sostenible, escalable y alineado con buenas prácticas de ingeniería de software.

¿Quieres que lo convierta en un formato de presentación, resumen ejecutivo o plantilla editable para Word? También puedo ayudarte a redactar una introducción institucional si lo vas a entregar como parte de un curso.

```
Cpp

<<<<<<< HEAD
// Versión local
=====
// Versión remota
>>>>>>> develop
```

Buenas prácticas:

- Comunicación clara entre colaboradores.
- Revisión de cambios antes del merge.
- Uso de herramientas como GitHub Desktop o VS Code para visualizar conflictos.

6. Pruebas antes de la fusión

Checklist de validación:

- El código compila sin errores.
- Las entradas inválidas son correctamente gestionadas.
- Los cálculos de IGV y montos son precisos.
- Los reportes reflejan correctamente los datos acumulados.

Prueba de funcionalidad:

- Registro de socio tipo 'A' con edad > 50 y sin hijos.
- Validación de cálculo de montoTotalVipMayores50 y montoAcumuladoSinHijos.

Comando para comparar ramas antes del merge:

```
Bash

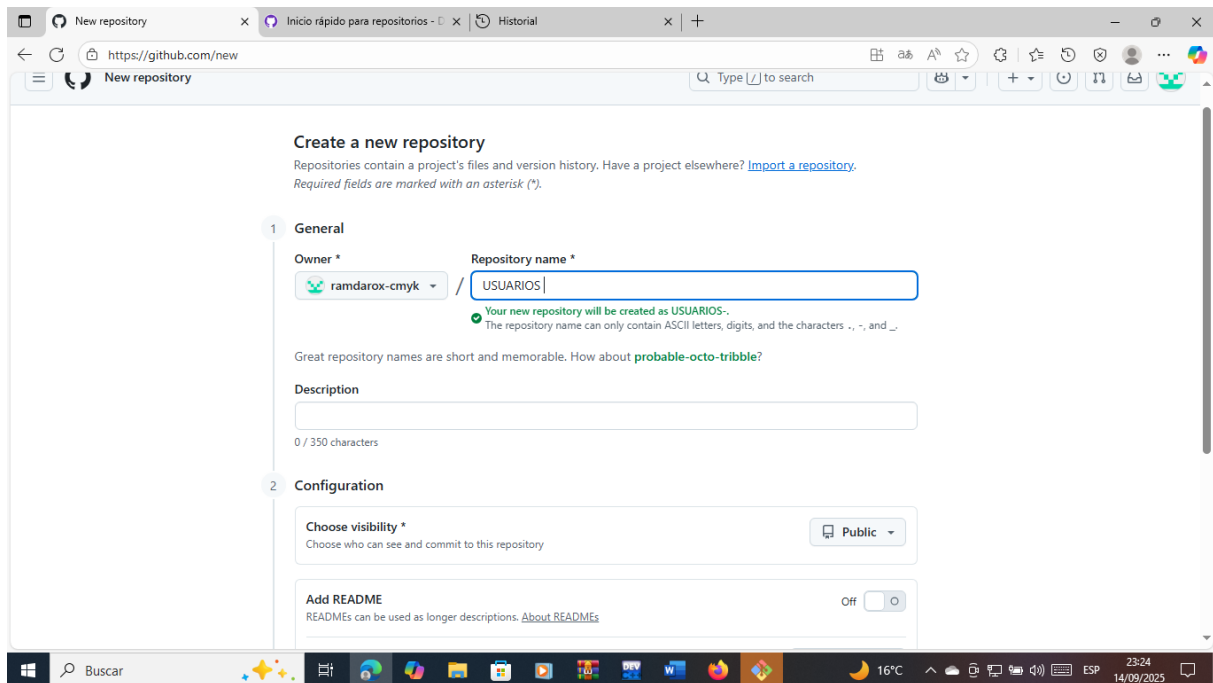
git diff develop feature/reportes
```

7. Conclusión

El uso de Git y GitHub ha permitido una gestión eficiente del código fuente, mientras que GitFlow ha facilitado la organización del desarrollo colaborativo. La automatización mediante GitHub Actions ha mejorado la calidad del software,

reduciendo errores humanos. La gestión proactiva de conflictos y la validación previa a la fusión de ramas aseguran la estabilidad del sistema.

Este enfoque técnico garantiza un desarrollo sostenible, escalable y alineado con buenas prácticas de ingeniería de software.



Create a new repository

Repositories contain a project's files and version history. Have a project elsewhere? [Import a repository.](#)
Required fields are marked with an asterisk (*).

1 General

Owner * ramdarox-cmyk / Repository name * USUARIOS

✓ Your new repository will be created as USUARIOS-
The repository name can only contain ASCII letters, digits, and the characters -, ., and _

Great repository names are short and memorable. How about [probable-octo-tribble](#)?

Description

0 / 350 characters

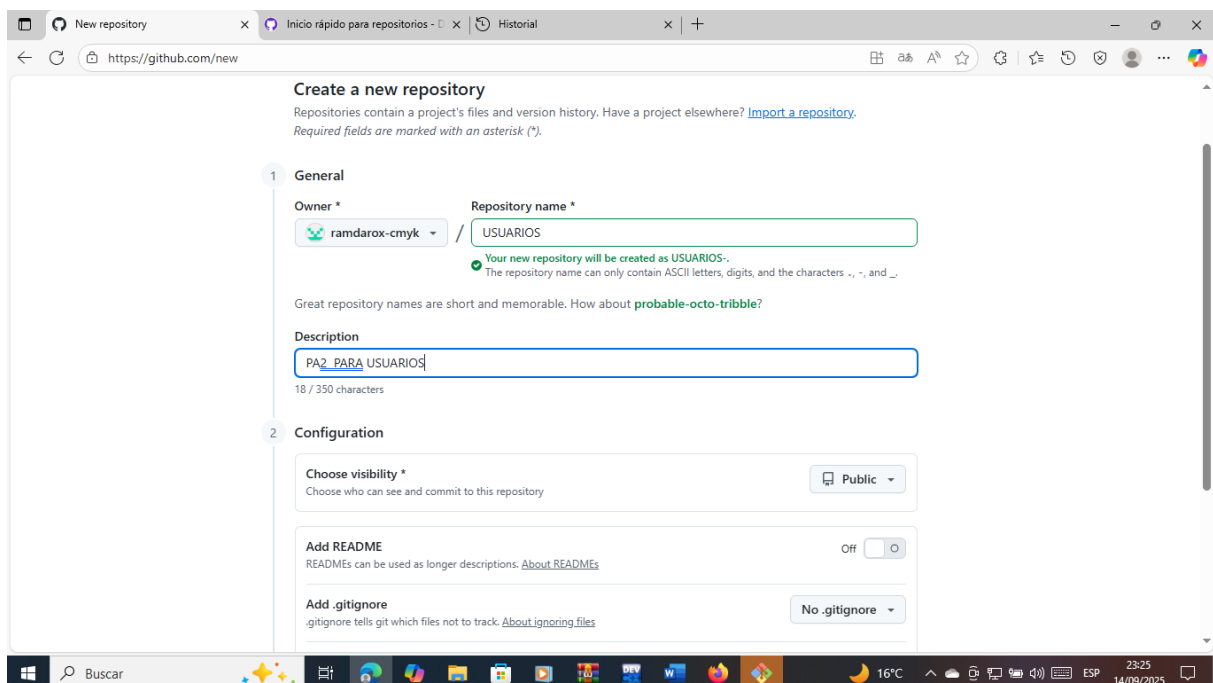
2 Configuration

Choose visibility * Public

Choose who can see and commit to this repository

Add README Off

READMEs can be used as longer descriptions. [About READMEs](#)



Create a new repository

Repositories contain a project's files and version history. Have a project elsewhere? [Import a repository.](#)
Required fields are marked with an asterisk (*).

1 General

Owner * ramdarox-cmyk / Repository name * USUARIOS

✓ Your new repository will be created as USUARIOS-
The repository name can only contain ASCII letters, digits, and the characters -, ., and _

Great repository names are short and memorable. How about [probable-octo-tribble](#)?

Description

PAZ PARA USUARIOS

18 / 350 characters

2 Configuration

Choose visibility * Public

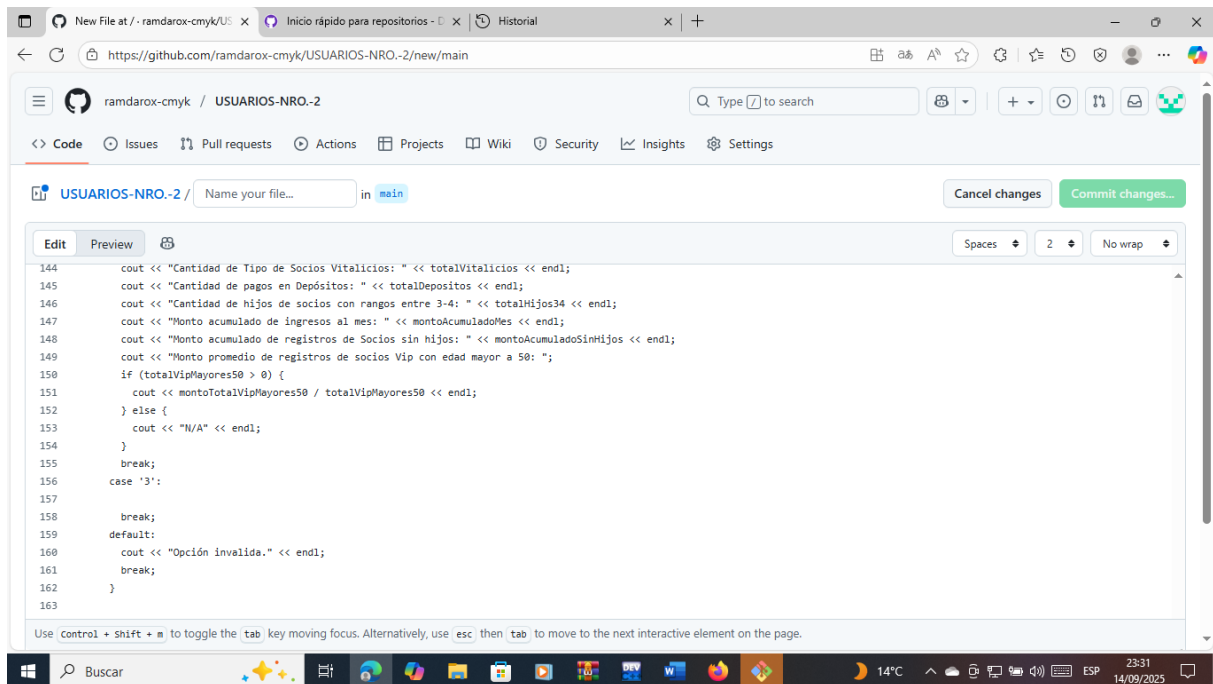
Choose who can see and commit to this repository

Add README Off

READMEs can be used as longer descriptions. [About READMEs](#)

Add .gitignore No .gitignore

.gitignore tells git which files not to track. [About ignoring files](#)

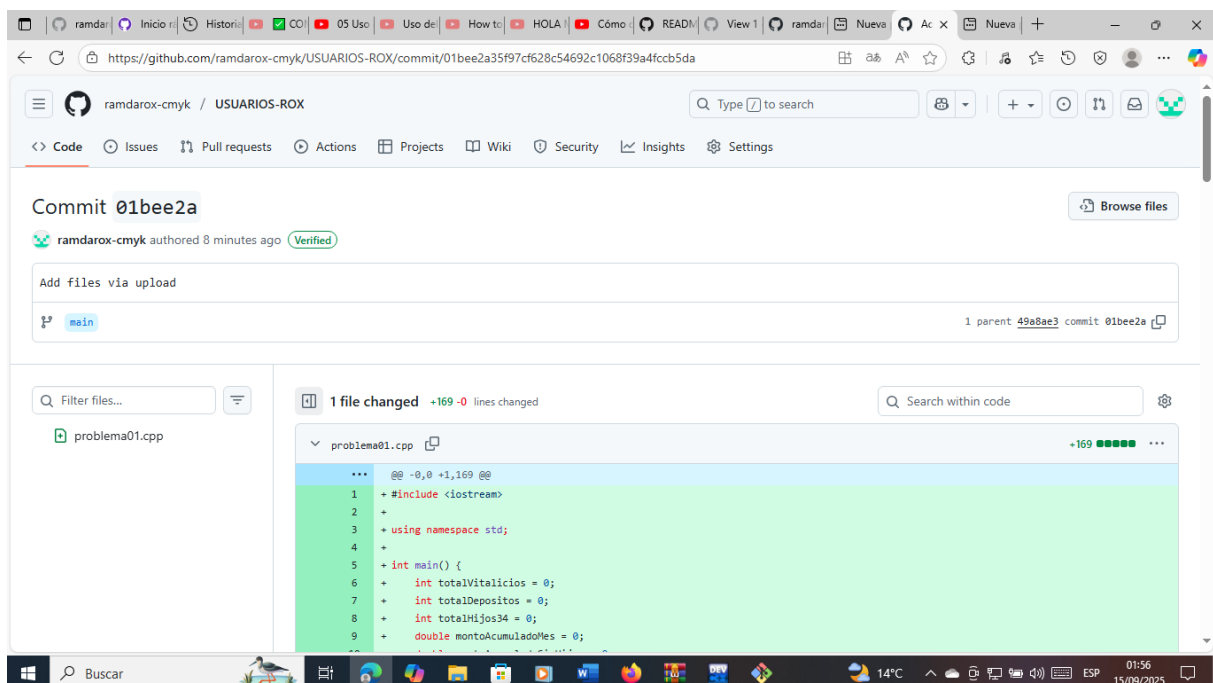


```

144     cout << "Cantidad de Tipo de Socios Vitalicios: " << totalVitalicios << endl;
145     cout << "Cantidad de pagos en Depósitos: " << totalDepositos << endl;
146     cout << "Cantidad de hijos de socios con rangos entre 3-4: " << totalHijos34 << endl;
147     cout << "Monto acumulado de ingresos al mes: " << montoAcumuladoMes << endl;
148     cout << "Monto acumulado de registros de Socios sin hijos: " << montoAcumuladoSinHijos << endl;
149     cout << "Monto promedio de registros de socios Vip con edad mayor a 50: ";
150     if (totalVipMayores50 > 0) {
151         cout << montoTotalVipMayores50 / totalVipMayores50 << endl;
152     } else {
153         cout << "N/A" << endl;
154     }
155     break;
156     case '3':
157
158     break;
159     default:
160         cout << "Opción inválida." << endl;
161         break;
162     }
163

```

¿CÓMO SE ABRE EL HIT BUSH ?



Commit 01bee2a

ramdarox-cmyk authored 8 minutes ago (Verified)

Add files via upload

1 parent 49a8ae3 commit 01bee2a

1 file changed +169 -0 lines changed

problema01.cpp

```

*** @@ -0,0 +1,169 @@
1 + #include <iostream>
2 +
3 + using namespace std;
4 +
5 + int main() {
6 +     int totalVitalicios = 0;
7 +     int totalDepositos = 0;
8 +     int totalHijos34 = 0;
9 +     double montoAcumuladoMes = 0;

```

<https://github.com/ramdarox-cmyk/USUARIOS-ROX.git>

Rúbrica de evaluación:

A continuación, se presenta la escala de valoración, con base a la cual se evaluará.

Criterios	Sobresaliente (5 puntos)	Suficiente (3 puntos)	En proceso (1 puntos)	En inicio (0 puntos)
Uso de Git y GitHub en la gestión del código	Gestionan el código de manera organizada en GitHub con commits bien documentados, ramas estructuradas y un historial claro de cambios.	Utilizan Git y GitHub correctamente, pero con documentación limitada o commits poco descriptivos.	Presentan uso básico de Git, con un historial de cambios poco estructurado.	No utilizan Git y GitHub adecuadamente o el repositorio está incompleto.
Implementación del flujo de trabajo GitFlow	Aplican correctamente GitFlow con ramas bien definidas, gestión adecuada de pull requests y colaboración eficiente entre los miembros del equipo.	Siguen parcialmente GitFlow, pero con errores en la administración de ramas o en la integración del código.	Implementan GitFlow de manera básica, con ramas poco organizadas o sin pull requests adecuados.	No implementan GitFlow en la gestión del proyecto.
Automatización de procesos en la construcción del software	Configuran correctamente herramientas de CI/CD para automatizar la construcción, pruebas e integración del software.	Implementan algún nivel de automatización, pero sin aprovechar todas las capacidades de CI/CD.	Presentan una automatización limitada o sin impacto en la calidad del software.	No aplican ninguna herramienta de automatización en la construcción del software.
Seguimiento de commits, ramas y pull requests	Realizan commits frecuentes y bien documentados, con ramas organizadas y pull requests explicativos.	Hacen commits regularmente, pero sin mantener una estructura clara de las ramas y pull requests.	Presentan commits desordenados o sin descripciones útiles, con ramas mal organizadas.	No mantienen un seguimiento adecuado del desarrollo con commits y ramas desorganizadas.
Manejo de conflictos y fusión de código	Gestionan correctamente la resolución de conflictos y realizan fusiones eficientes sin afectar la estabilidad del código.	Resuelven conflictos de manera funcional, aunque con algunas inconsistencias en la integración del código.	Presentan problemas en la resolución de conflictos y en la integración de cambios.	No manejan conflictos ni realizan fusiones de código adecuadas.
Total parcial				
Nota	Promedio = (Nota parcial x 20)/25			