

Universidad de Costa Rica
Facultad de Ingeniería
Escuela de Ingeniería Eléctrica
IE0117: Programación Bajo Plataformas Abiertas
I ciclo 2024

Reporte
Laboratorio 3: Introducción a C

César Alvarado Castro, B60306
Grupo 01

Profesora: Mariela Hernández Chacón

30 de abril de 2024

Índice

1. Parte 1 – Resolución de conflictos en Git	1
1.1. Fork en Git	1
1.2. Propósito principal	1
1.3. Utilidades	1
1.4. Creación	1
1.5. Solución de conflicto en GitHub	1
1.5.1. Configuración	1
1.5.2. Branching o ramas	2
2. Parte 2 – Calculadora en C	11

Índice de figuras

1.	Creación de Fork [1].	1
2.	Clonado del repositorio.	2
3.	Ramas.	2
4.	Función añadida.	3
5.	Llamado de la función.	3
6.	Declaración de la función.	3
7.	Se añade el commit.	4
8.	Comprobación.	4
9.	Cambio de branch.	4
10.	Función.	5
11.	Llamada de la función.	5
12.	Declaración de función.	6
13.	Creación del commit.	6
14.	Se hace git push	6
15.	Comprobación.	7
16.	Conflicto.	7
17.	Marcadores de conflicto en la definición de funciones.	8
18.	Marcadores de conflicto en la llamada a funciones.	8
19.	Solución del conflicto.	9
20.	Marcadores de conflicto en la declaración de funciones.	9
21.	Verificación de solución del conflicto.	10
22.	Se actualiza la rama al repositorio remoto.	10
23.	Verificación del programa.	10
24.	Verificación del programa.	11

1. Parte 1 – Resolución de conflictos en Git

1.1. Fork en Git

Un Fork (bifurcación por su traducción del inglés) es un nuevo repositorio que copia el código, archivos y configuración de visibilidad del repositorio original (ascendente) [1].

1.2. Propósito principal

El propósito general de la creación de un Fork es que se pueda trabajar en el desarrollo de un proyecto sin afectar el repositorio original (también llamado, upstream). Sin embargo los cambios que sufra el repositorio original pueden ser actualizados al Fork y viceversa, se pueden sugerir cambios al repositorio original desde el Fork mediante un pull request [1].

1.3. Utilidades

- Los Forks mantienen las restricciones de sus repositorios originales [1].
- Los Forks cuentan con sus propias wikis, proyectos, acciones, discusiones, solicitudes de extracción, propuestas, políticas, etiquetas, ramas y miembros [1].
- Se puede actualizar el Fork con el repositorio original [1].
- Se pueden hacer cambios al repositorio original mediante un pull request [1].

1.4. Creación

Se debe acceder al repositorio al que se desea hacer Fork y luego click en el botón Fork arriba a la derecha como se muestra en Figura 1.

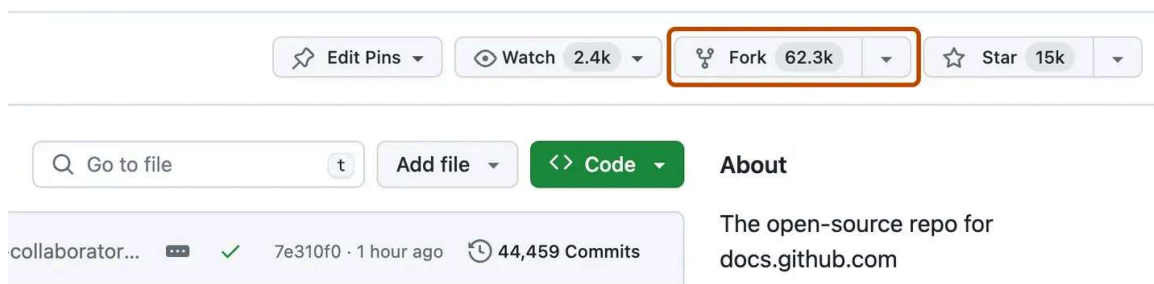


Figura 1: Creación de Fork [1].

Se puede hacer fork de repositorios públicos y también a privados si se tiene acceso a ellos y además el propietario habilitó la opción de poder realizar Forks [1].

1.5. Solución de conflicto en GitHub

1.5.1. Configuración

Se hace fork del repositorio **c-examples**.

Se clona el nuevo repositorio creado en el espacio de trabajo como se muestra en la Figura

```
Documents Music Public snap Videos
cesar@cesar:~$ cd repositories/
cesar@cesar:~/repositories$ git clone <https://github.com/cesar-21/c-examples.git>
bash: syntax error near unexpected token `newline'
cesar@cesar:~/repositories$ git clone https://github.com/cesar-21/c-examples.git
Cloning into 'c-examples'...
remote: Enumerating objects: 36, done.
remote: Counting objects: 100% (13/13), done.
remote: Compressing objects: 100% (3/3), done.
remote: Total 36 (delta 10), reused 10 (delta 10), pack-reused 23
Receiving objects: 100% (36/36), 14.28 KiB | 860.00 KiB/s, done.
Resolving deltas: 100% (15/15), done.
cesar@cesar:~/repositories$ ls
c-examples
cesar@cesar:~/repositories$ cd c-examples/
cesar@cesar:~/repositories/c-examples$ ls
funciones intro_c secuencias
cesar@cesar:~/repositories/c-examples$
```

Figura 2: Clonado del repositorio.

1.5.2. Branching o ramas

Se crean las ramas **feature-branch** y **main** (o master) como se muestra en la Figura 3. Se observa como la rama **main** ya existía por ser predeterminada.

```
cesar@cesar:~/repositories/c-examples$ git checkout -b main
fatal: A branch named 'main' already exists.
cesar@cesar:~/repositories/c-examples$ git checkout -b feature-branch
Switched to a new branch 'feature-branch'
cesar@cesar:~/repositories/c-examples$
```

Figura 3: Ramas.

Cambios diferentes en cada una de las ramas

Cambios en main

Se abre el archivo **/funciones/9-encabezados.c** y se agrega una función llamada **funcion4()** que reste como se muestra en la Figura 4 y se añade la llamada de esta función en **int main()** como se muestra en la Figura 5.

```

cesar@cesar: ~/repositories/c-examples
// funcion3(): Imprime un mensaje. Retorna el valor que recibió como argumento
// incrementado en 2
int funcion3(int x) {
    printf("En funcion3()...\n");
    int y = x + 2;
    return y;
}

// funcion4(): Imprime un mensaje. Retorna el valor que recibió como argumento
// disminuido en 2
int funcion4(int x) {
    printf("En funcion4()...\n");
    int y = x - 2;
    return y;
}

```

Figura 4: Función añadida.

```

cesar@cesar: ~/repositories/c-examples/funciones
cesar@cesar: ~/repositories/c-examples/funciones

int v;

// Llama a las 3 funciones
funcion1();
funcion2(5, 2.67);
v = funcion3(10);

// Imprime el valor que retornó funcion3()
printf("v = %d\n", v);

v = funcion4(10);

// Imprime el valor que retornó funcion4()
printf("v = %d\n", v);

return 0;

```

Figura 5: Llamado de la función.

Se abre el archivo `/funciones/9-encabezados.h` y se agrega la declaración de la `funcion4()` al final del archivo. Como se muestra en la Figura 6.

```

cesar@cesar: ~/repositories/c-examples/funciones

/* Prototipos o encabezados de las funciones */
void funcion1(void);
void funcion2(int, float);
int funcion3(int);
int funcion4(int);

```

Figura 6: Declaración de la función.

Se hace un commit en el Branch con los cambios y luego un push como se muestra en las Figuras 7 y 8.

```
cesar@cesar: ~/repositories/c-examples/funciones
cesar@cesar:~/repositories/c-examples/funciones$ git add 9-encabezados.c
cesar@cesar:~/repositories/c-examples/funciones$ git add 9-encabezados.h
cesar@cesar:~/repositories/c-examples/funciones$ 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)
        modified:   9-encabezados.c
        modified:   9-encabezados.h

cesar@cesar:~/repositories/c-examples/funciones$ git commit -m "Cambios en main"
[main 040b821] Cambios en main
2 files changed, 33 insertions(+)
```

Figura 7: Se añade el commit.

```
cesar@cesar: ~/repositories/c-examples/funciones
cesar@cesar:~/repositories/c-examples/funciones$ git push
Username for 'https://github.com': cesar-21
Password for 'https://cesar-21@github.com':
Enumerating objects: 9, done.
Counting objects: 100% (9/9), done.
Delta compression using up to 2 threads
Compressing objects: 100% (5/5), done.
Writing objects: 100% (5/5), 567 bytes | 567.00 KiB/s, done.
Total 5 (delta 3), reused 0 (delta 0), pack-reused 0
remote: Resolving deltas: 100% (3/3), completed with 3 local objects.
To https://github.com/cesar-21/c-examples.git
   5716b99..040b821  main -> main
cesar@cesar:~/repositories/c-examples/funciones$ git log
commit 040b8217620bd13ef4384d9dfb984a985e20e4d2 (HEAD -> main, origin/main, origin/HEAD)
Author: cesar-21 <cesar.alvaradocastro@ucr.ac.cr>
Date: Thu May 2 23:03:01 2024 -0600

    Cambios en main
```

Figura 8: Comprobación.

Cambios en feature-branch

Se cambia de branch como se muestra en la Figura 9.

```
cesar@cesar: ~/repositories/c-examples/funciones
cesar@cesar:~/repositories/c-examples$ git checkout feature-branch
Switched to branch 'feature-branch'
cesar@cesar:~/repositories/c-examples$ cd funciones/
cesar@cesar:~/repositories/c-examples/funciones$ vim 9-encabezados.c
cesar@cesar:~/repositories/c-examples/funciones$ vim 9-encabezados.h
cesar@cesar:~/repositories/c-examples/funciones$
```

Figura 9: Cambio de branch.

Se abre el archivo `/funciones/9-encabezados.c` y se agrega una función llamada `funcion5()` que suma 3 como se muestra en la Figura 10 y se añade la llamada de esta función en `int main()` como se muestra en la Figura 11.

```

cesar@cesar: ~/repositories/c-examples/funciones
printf("En funcion2()...\n");
printf("a = %d, b = %.2f\n", a, b);
}

// funcion3(): Imprime un mensaje. Retorna el valor que recibió como argumento
// incrementado en 2
int funcion3(int x) {
    printf("En funcion3()...\n");
    int y = x + 2;
    return y;
}

// funcion5(): Imprime un mensaje. Retorna el valor que recibió como argumento
// incrementado en 3
int funcion5(int x) {
    printf("En funcion5()...\n");
    int y = x + 3;
    return y;
}

```

Figura 10: Función.

```

cesar@cesar: ~/repositories/c-examples/funciones
int main() {

    int v;

    // Llama a las 3 funciones
    funcion1();
    funcion2(5, 2.67);
    v = funcion3(10);

    // Imprime el valor que retornó funcion3()
    printf("v = %d\n", v);

    v = funcion5(10);

    // Imprime el valor que retornó funcion5()
    printf("v = %d\n", v);

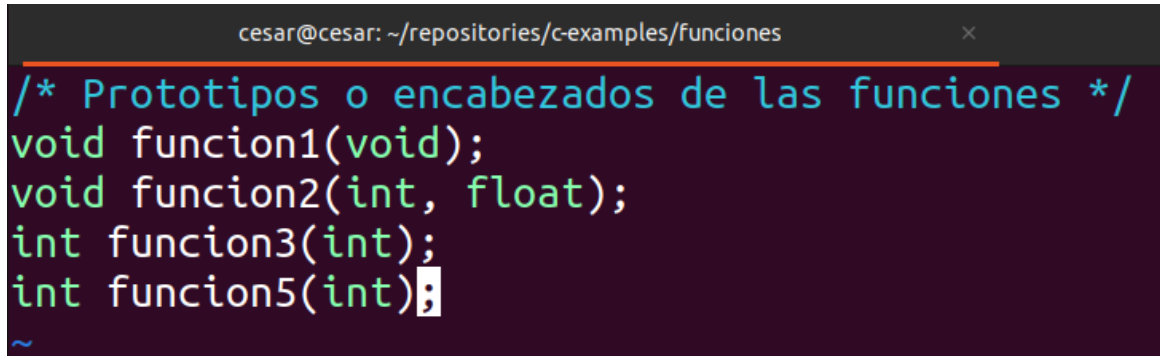
    return 0;
}

```

Figura 11: Llamada de la función.

Se abre el archivo `/funciones/9-encabezados.h` y se agrega la declaración de la **fun-**

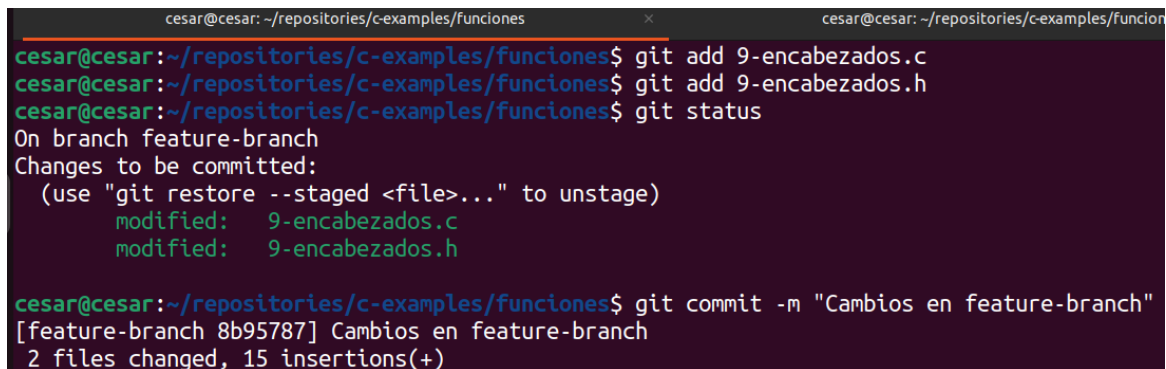
cion5() al final del archivo. Como se muestra en la Figura 12.



```
cesar@cesar: ~/repositories/c-examples/funciones
/* Prototipos o encabezados de las funciones */
void funcion1(void);
void funcion2(int, float);
int funcion3(int);
int funcion5(int);
~
```

Figura 12: Declaración de función.

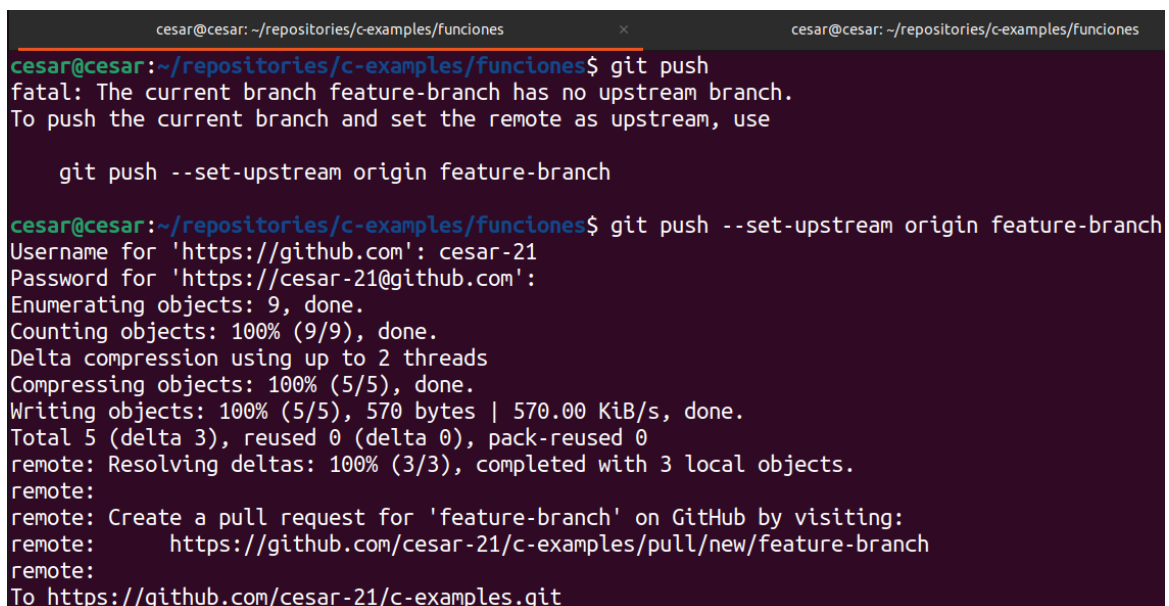
Se hace un commit en el Branch con los cambios y luego un push como se muestra en las Figuras 13, 14 y 15.



```
cesar@cesar: ~/repositories/c-examples/funciones
cesar@cesar:~/repositories/c-examples/funciones$ git add 9-encabezados.c
cesar@cesar:~/repositories/c-examples/funciones$ git add 9-encabezados.h
cesar@cesar:~/repositories/c-examples/funciones$ git status
On branch feature-branch
Changes to be committed:
  (use "git restore --staged <file>..." to unstage)
        modified:   9-encabezados.c
        modified:   9-encabezados.h

cesar@cesar:~/repositories/c-examples/funciones$ git commit -m "Cambios en feature-branch"
[feature-branch 8b95787] Cambios en feature-branch
 2 files changed, 15 insertions(+)
```

Figura 13: Creación del commit.



```
cesar@cesar: ~/repositories/c-examples/funciones
cesar@cesar:~/repositories/c-examples/funciones$ git push
fatal: The current branch feature-branch has no upstream branch.
To push the current branch and set the remote as upstream, use

    git push --set-upstream origin feature-branch

cesar@cesar:~/repositories/c-examples/funciones$ git push --set-upstream origin feature-branch
Username for 'https://github.com': cesar-21
Password for 'https://cesar-21@github.com':
Enumerating objects: 9, done.
Counting objects: 100% (9/9), done.
Delta compression using up to 2 threads
Compressing objects: 100% (5/5), done.
Writing objects: 100% (5/5), 570 bytes | 570.00 KiB/s, done.
Total 5 (delta 3), reused 0 (delta 0), pack-reused 0
remote: Resolving deltas: 100% (3/3), completed with 3 local objects.
remote:
remote: Create a pull request for 'feature-branch' on GitHub by visiting:
remote:   https://github.com/cesar-21/c-examples/pull/new/feature-branch
remote:
To https://github.com/cesar-21/c-examples.git
```

Figura 14: Se hace **git push**.

```
cesar@cesar:~/repositories/c-examples/funciones$ git log
commit 8b95787c39c4b9696eab3c1e2529b079b1622b56 (HEAD -> feature-branch, origin/feature-branch)
Author: cesar-21 <cesar.alvaradocastro@ucr.ac.cr>
Date: Thu May 2 23:50:46 2024 -0600

Cambios en feature-branch
```

Figura 15: Comprobación.

Conflicto Merge

Se intenta fusionar la rama **feature-branch** en Fusione la rama feature-branch en main (usando el comando **git merge**) como se muestra en la Figura 16, donde se puede apreciar que se indica el conflicto por diferencias en los archivos **9-encabezados.c** y **9-encabezados.h** entre las 2 ramas.

```
cesar@cesar:~/repositories/c-examples/funciones$ git merge main
Auto-merging funciones/9-encabezados.c
CONFLICT (content): Merge conflict in funciones/9-encabezados.c
Auto-merging funciones/9-encabezados.h
CONFLICT (content): Merge conflict in funciones/9-encabezados.h
Automatic merge failed; fix conflicts and then commit the result.
cesar@cesar:~/repositories/c-examples/funciones$
```

Figura 16: Conflicto.

En la Figura 17 GitHub resalta con marcadores de conflicto cuales cambios fueron hechos en cada rama durante la definición de las funciones. Y en la Figura 18 se muestra lo mismo pero durante las llamadas a las funciones en **main**. Se puede observar también que la línea **printf("v = %d\n", v);** solo aparece una vez, ya GitHub resuelve automáticamente que como es lo mismo, solo debe ir una vez. Sin embargo, esto no es lo que se quiere en el programa, ya que solo se imprimiría en terminal solamente una de las funciones, por lo que se arregla esto y se ordena otra vez el código, y por último se eliminan los marcadores de conflicto como se muestra en la Figura 19.

```

cesar@cesar: ~/repositories/c-examples

<<<<<<< HEAD
// funcion5(): Imprime un mensaje. Retorna el valor que recibió como argumento
// incrementado en 3
int funcion5(int x) {
    printf("En funcion5()...\n");
    int y = x + 3;
    return y;
}

=====

// funcion4(): Imprime un mensaje. Retorna el valor que recibió como argumento
// disminuido en 2
int funcion4(int x) {
    printf("En funcion4()...\n");
    int y = x - 2;
    return y;
}

```

Figura 17: Marcadores de conflicto en la definición de funciones.

```

cesar@cesar: ~/repositories/c-examples/funciones

// Llama a las 3 funciones
funcion1();
funcion2(5, 2.67);
v = funcion3(10);

// Imprime el valor que retornó funcion3()
printf("v = %d\n", v);

<<<<<<< HEAD
v = funcion5(10);

// Imprime el valor que retornó funcion5()
=====
v = funcion4(10);

// Imprime el valor que retornó funcion4()
>>>>>>> main
printf("v = %d\n", v);

return 0;
-- INSERT --

```

Figura 18: Marcadores de conflicto en la llamada a funciones.

```

cesar@cesar: ~/repositories/c-examples/funciones
int main() {

    int v;

    // Llama a las 3 funciones
    funcion1();
    funcion2(5, 2.67);
    v = funcion3(10);

    // Imprime el valor que retornó funcion3()
    printf("v = %d\n", v);

    v = funcion4(10);
    // Imprime el valor que retornó funcion4()
    printf("v = %d\n", v);

    v = funcion5(10);

    // Imprime el valor que retornó funcion5()
    printf("v = %d\n", v);

    return 0;
}
-- INSERT --

```

Figura 19: Solución del conflicto.

En la Figura 20 GitHub resalta cuales cambios fueron hechos en cada rama durante declaración de las funciones en el archivo **9-encabezados.h**. Una vez más se eliminan las indicaciones de GitHub y se ordena el código.

```

cesar@cesar: ~/repositories/c-examples/funciones
/* Prototipos o encabezados de las funciones */
void funcion1(void);
void funcion2(int, float);
int funcion3(int);
<<<<<<< HEAD
int funcion5(int);
=====
int funcion4(int);
>>>>>>> main

```

Figura 20: Marcadores de conflicto en la declaración de funciones.

En la Figura 21 se continúa con el comando **git merge --continue** ya que se resolvió el conflicto. Y se termina de subir al repositorio como se muestra en la Figura 22

```
cesar@cesar: ~/repositories/c-examples/funciones
cesar@cesar:~/repositories/c-examples/funciones$ vim 9-encabezados.c
cesar@cesar:~/repositories/c-examples/funciones$ vim 9-encabezados.c
cesar@cesar:~/repositories/c-examples/funciones$ vim 9-encabezados.h
cesar@cesar:~/repositories/c-examples/funciones$ git add 9-encabezados.c
cesar@cesar:~/repositories/c-examples/funciones$ git add 9-encabezados.h
cesar@cesar:~/repositories/c-examples/funciones$ git merge --continue
[feature-branch 1043515] Merge branch 'main' into feature-branch
```

Figura 21: Verificación de solución del conflicto.

```
cesar@cesar: ~/repositories/c-examples/funciones
cesar@cesar:~/repositories/c-examples/funciones$ git push
Username for 'https://github.com': cesar-21
Password for 'https://cesar-21@github.com':
Enumerating objects: 13, done.
Counting objects: 100% (13/13), done.
Delta compression using up to 2 threads
Compressing objects: 100% (5/5), done.
Writing objects: 100% (5/5), 610 bytes | 610.00 KiB/s, done.
Total 5 (delta 3), reused 0 (delta 0), pack-reused 0
remote: Resolving deltas: 100% (3/3), completed with 3 local objects.
To https://github.com/cesar-21/c-examples.git
   8b95787..1043515 feature-branch -> feature-branch
cesar@cesar:~/repositories/c-examples/funciones$ git log
commit 1043515fd0a986a9fc544b824e63412d949a47d8 (HEAD -> feature-branch, origin/feature-branch)
Merge: 8b95787 040b821
Author: cesar-21 <cesar.alvaradocastro@ucr.ac.cr>
Date: Fri May 3 00:33:20 2024 -0600

    Merge branch 'main' into feature-branch

commit 8b95787c39c4b9696eab3c1e2529b079b1622b56
Author: cesar-21 <cesar.alvaradocastro@ucr.ac.cr>
Date: Thu May 2 23:50:46 2024 -0600

    Cambios en feature-branch
```

Figura 22: Se actualiza la rama al repositorio remoto.

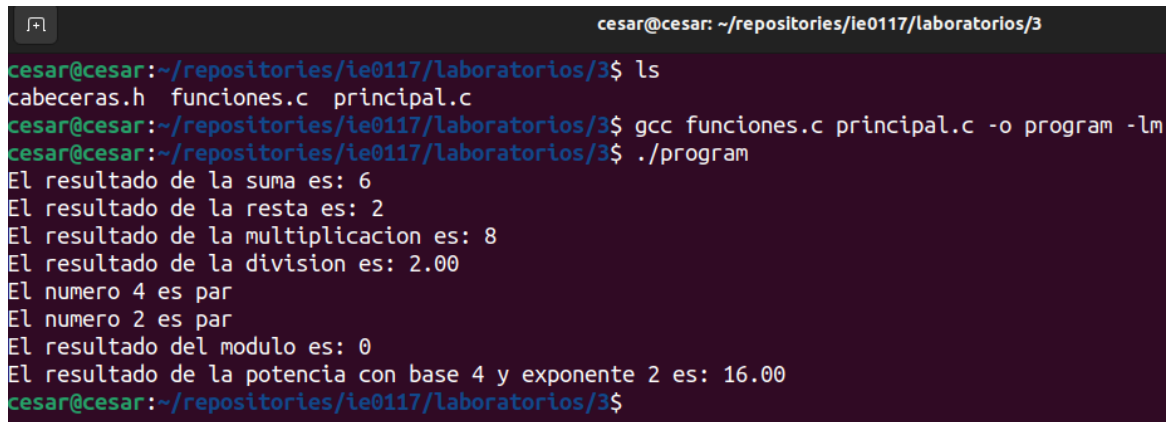
En la Figura 23 se muestra finalmente la compilación y se corre el programa.

```
cesar@cesar: ~/repositories/c-examples/funciones
cesar@cesar:~/repositories/c-examples/funciones$ gcc 9-encabezados.c -o encabezados
cesar@cesar:~/repositories/c-examples/funciones$ ./encabezados
En funcion1()...
En funcion2()...
a = 5, b = 2.67
En funcion3()...
v = 12
En funcion4()...
v = 8
En funcion5()...
v = 13
cesar@cesar:~/repositories/c-examples/funciones$
```

Figura 23: Verificación del programa.

2. Parte 2 – Calculadora en C

En la Figura 24 se observan los 3 archivos pedidos. También se muestra la forma de compilarlo, agregando **-lm** al final para poder usar las funciones de la librería **math.h** correctamente.



```
cesar@cesar: ~/repositories/ie0117/laboratorios/3
cesar@cesar:~/repositories/ie0117/laboratorios/3$ ls
cabeceras.h  funciones.c  principal.c
cesar@cesar:~/repositories/ie0117/laboratorios/3$ gcc funciones.c principal.c -o program -lm
cesar@cesar:~/repositories/ie0117/laboratorios/3$ ./program
El resultado de la suma es: 6
El resultado de la resta es: 2
El resultado de la multiplicacion es: 8
El resultado de la division es: 2.00
El numero 4 es par
El numero 2 es par
El resultado del modulo es: 0
El resultado de la potencia con base 4 y exponente 2 es: 16.00
cesar@cesar:~/repositories/ie0117/laboratorios/3$
```

Figura 24: Verificación del programa.

Referencias

- [1] About forks. GitHub Docs. [Online]. Available: <https://docs.github.com/es/pull-requests/collaborating-with-pull-requests/working-with-forks/about-forks>