## Task 1:

function get\_GCD(a, b)

// Base case: if b is 0, return a

if b = 0 then

return a

end if

// recursively call the function again: get\_GCD(b, remainder)

return get\_GCD(b, a mod b)

end function

## Task 5:

To keep track of commits and the differences between each commit, we can use various Git commands such as:

1. **git init**: This command initialises a new Git repository in the current directory.
2. **git add**: This command adds changes to the staging area, which is a place where Git tracks the changes you want to include in the next commit. You can use git add . to add all changes in the current directory or git add filename to add a specific file.
3. **git commit**: This command creates a new commit with the changes that have been added to the staging area. You can use git commit -m "Commit message" to create a commit with a descriptive message explaining the changes made.

To keep track of your commits and the differences between each commit, you can follow these steps:

1. As you make changes to your codebase, use git add to stage the changes you want to include in the next commit.
2. Use git commit -m "The commit message" to create a new commit with your staged changes.
3. If you want to see the differences between a specific commit and the current state of your codebase, use git diff [The hashes of the commits you want to compare].

By using these Git commands, you can effectively keep track of your commits, view the changes made in each commit, and compare the differences between different commits. This can be extremely useful for code review, debugging, and tracking project history. Additionally, Git provides various tools and interfaces, such as Git GUI clients and web-based Git hosting services like GitHub, which can make it easier to visualise and explore commit histories and differences between commits.

GitHub Project Link: <https://github.com/Munaikh/FC723-Assignment-3>

## Task 6:

The traditional Euclidean Algorithm finds the GCD of two numbers, but we can extend it to handle multiple numbers by repeatedly applying the algorithm to pairs of numbers.

### Pseudocode:

// create new function with a list as an input

function gcdMultiple(numbers)

if length(numbers) == 0

return 0 // No numbers provided

else if length(numbers) == 1

return numbers[0] // Only one number provided

else

gcd = gcd(numbers[0], numbers[1]) // Calculate GCD of the first two numbers

for i = 2 to length(numbers)

gcd = gcd(gcd, numbers[i]) // Calculate GCD of the current GCD and the next number

end for

return gcd

end if

end function