

Notice

Our Department is currently collaborating with Samsung R&D Institute Bangladesh (SRBD) on internships and capstone project. To further strengthen this cooperation, SRBD will visit our university on September 24 to provide an introduction session about their organization, project and vision.

- Date & Time: Wednesday, September 24, 14:30–15:30
- Location: S2 301

If you have another class at the same time, check with your professor in advance to see if informing them will allow you to receive attendance credit.

C Programming (W2)



Welcome!!

Please check attendance individually.
(Mobile App)

Things to do today

1. Install VSC, MinGW64 (Google Drive)
2. Create github repositories
3. Make C program

01 | Folder & File / File extension

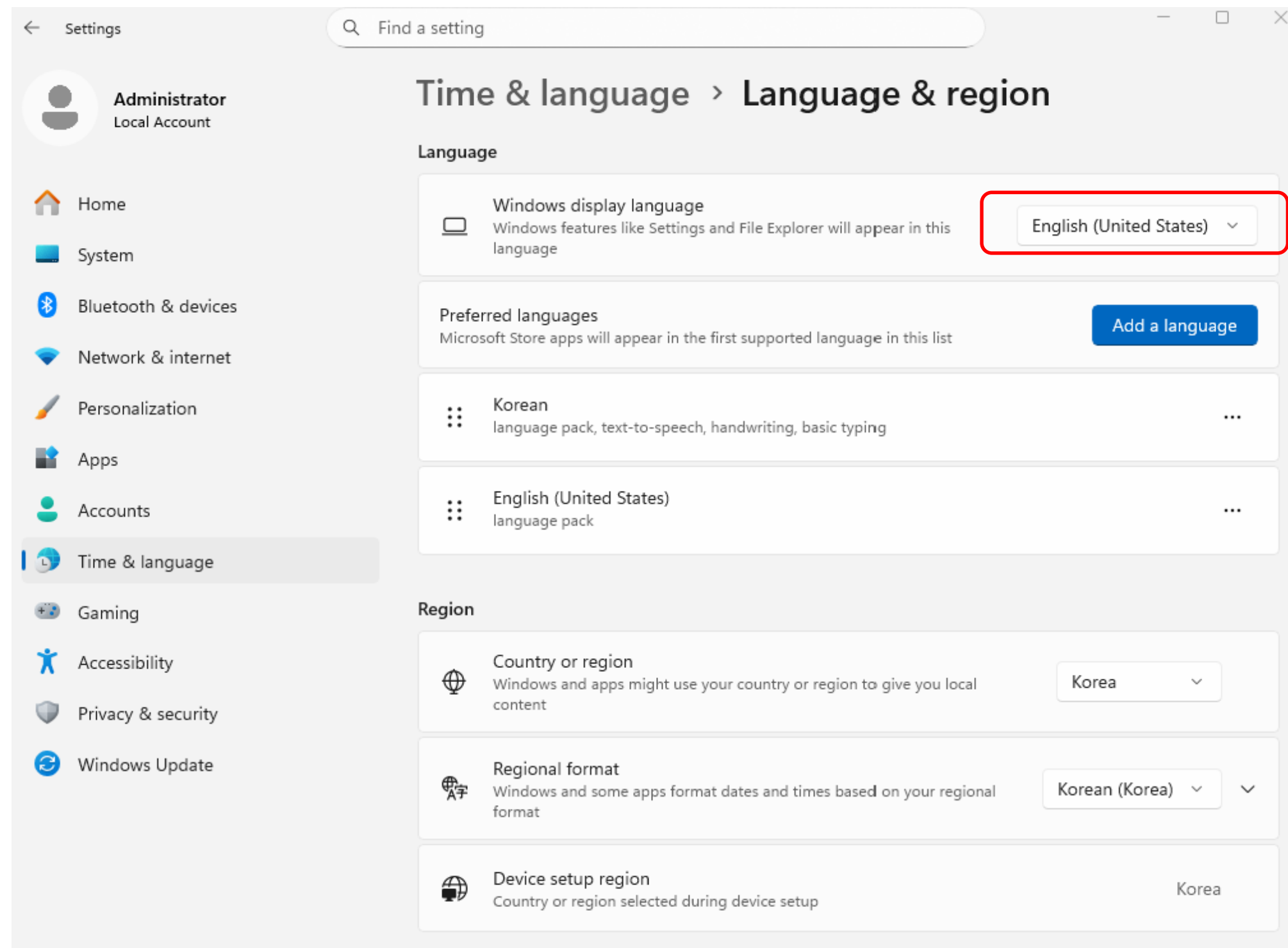
02 | Shell

03 | Homework with github classroom & git

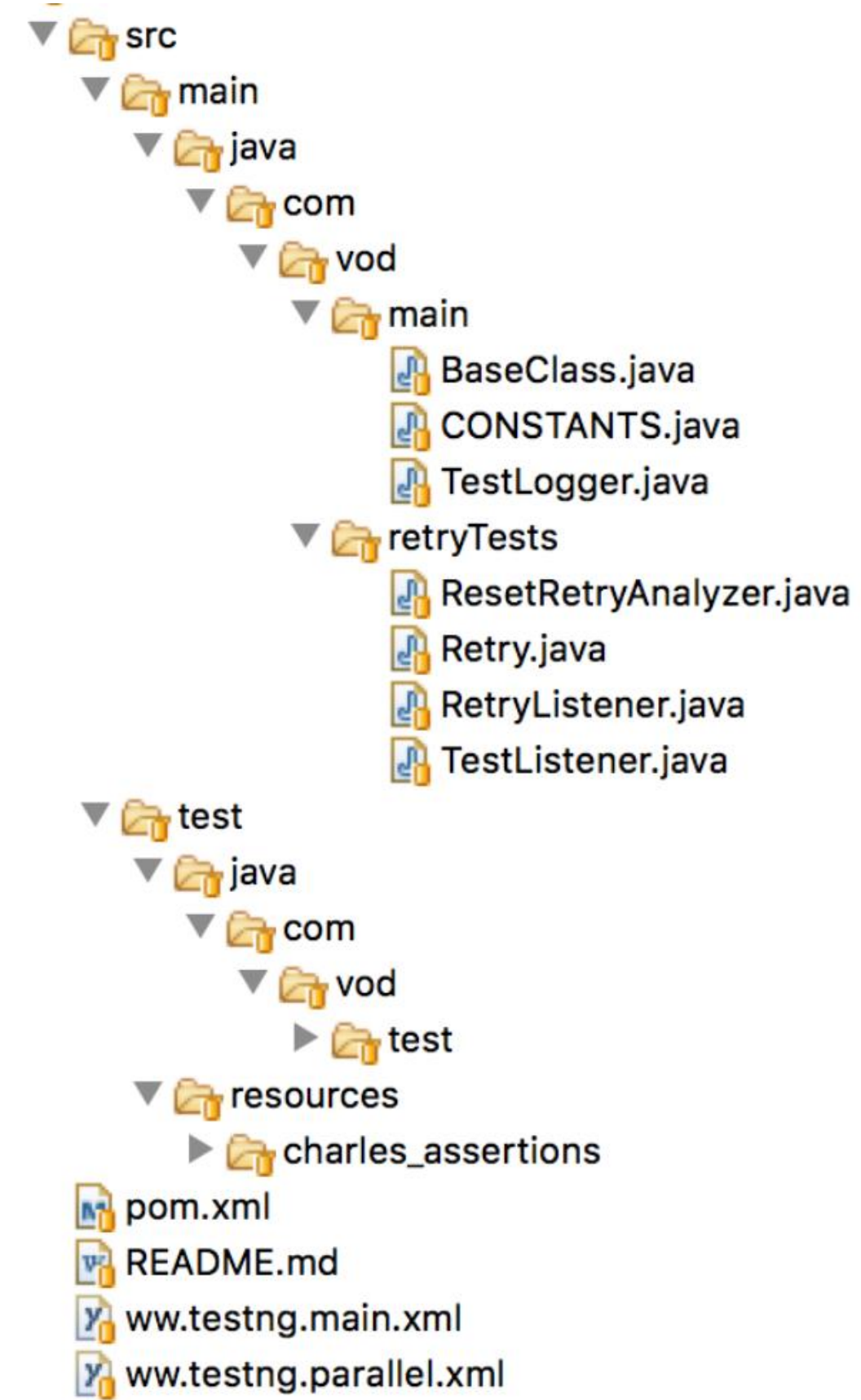
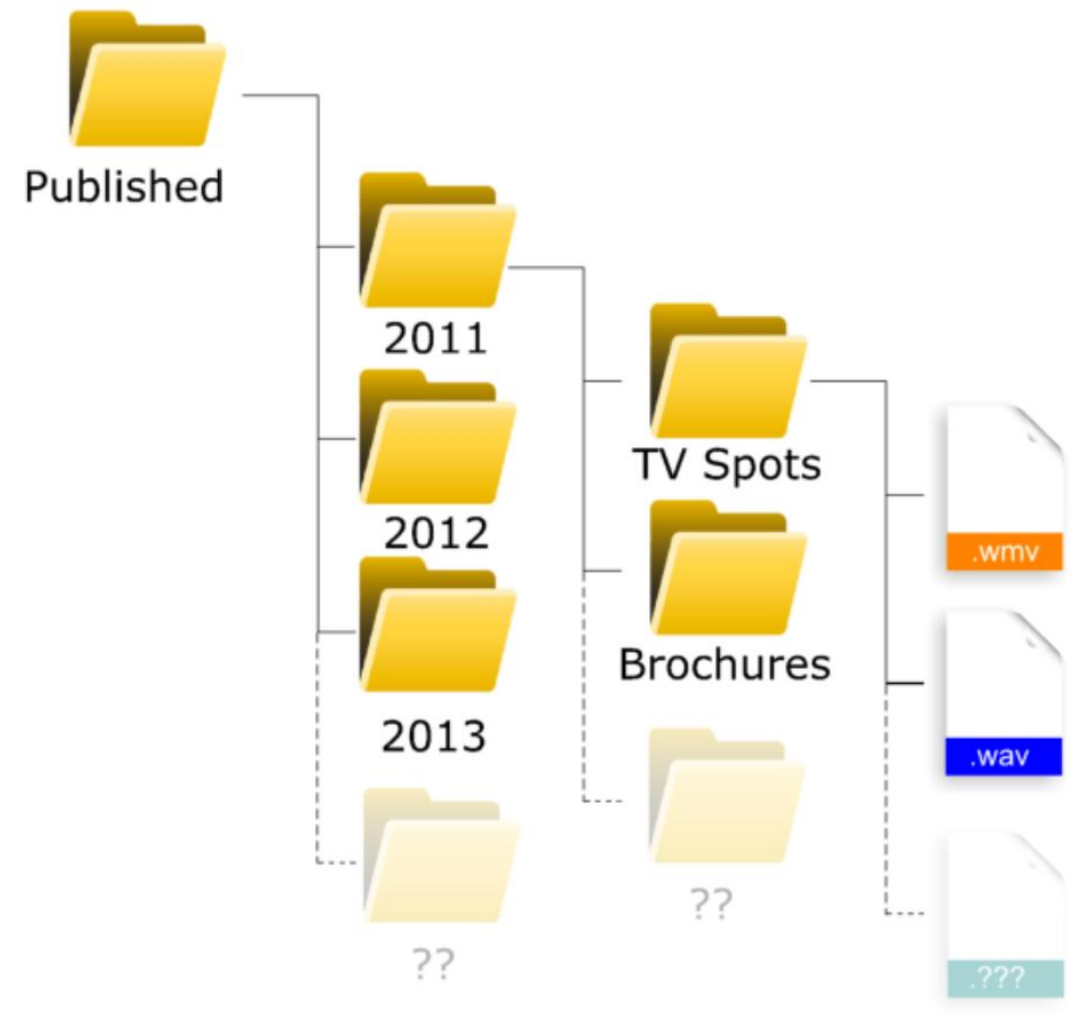
04 | C compile process

Change language (The school is blocking the installation of language packs.)

Please use English rather than your native language. In particular, when you encounter a C error message, search in English - you'll find much more resources and solutions.

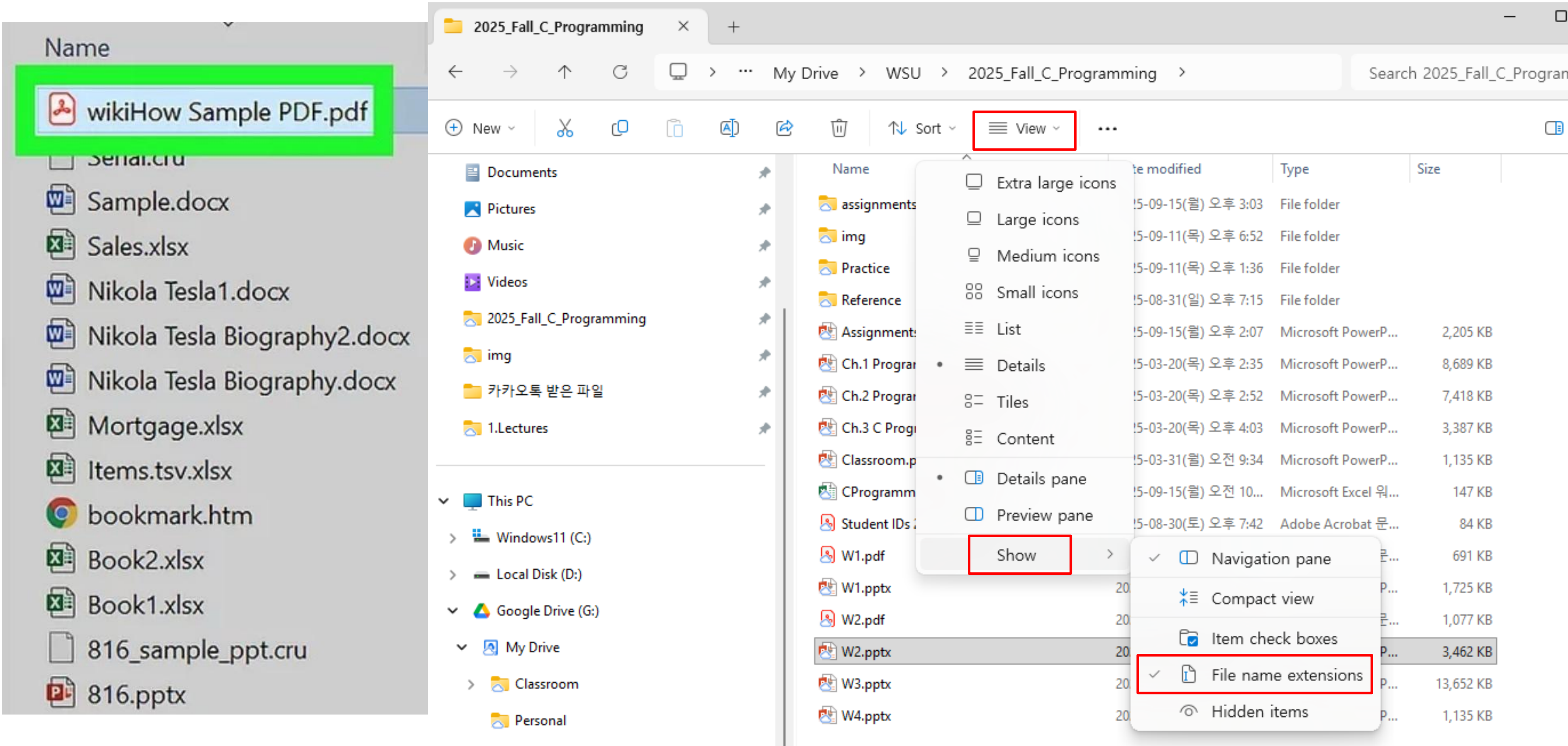


Folder & File



File extension

- *running a file using the metaphor of calling a person's name*



Shell

A **shell** is like a **translator between you and the operating system (OS)**.

It takes the commands you type, sends them to the OS, and then shows you the results.

In a PC, a **shell** is a user interface that allows you to interact with the operating system. It can be **command-line-based (CLI)** or **graphical (GUI)**:

1. Command-Line Shell (CLI) – This is a text-based interface where users enter commands to perform tasks. Examples:

- **Bash** (Linux, macOS)
- **PowerShell** (Windows)
- **Command Prompt (cmd.exe)** (Windows)
- **Zsh, Fish** (Unix-like systems)

2. Graphical Shell (GUI) – This provides a visual interface with icons, windows, and menus. Examples:

- **Windows Explorer** (Windows shell)
- **GNOME/KDE/Xfce** (Linux desktop environments)
- **macOS Finder** (Mac shell)

Terminology

- **Terminal**(room) is the program that opens a window to interact with the system.
 - A **terminal** is the window/app where you type commands and see the text output.
 - It's the interface for interacting with a shell.
 - **Console**: historically the physical I/O; today often used loosely like “terminal.”
- **Shell**(teacher) is the command interpreter that runs inside the terminal and translates user commands into actions performed by the operating system.

“Open windows terminal using Command Prompt shell”

PowerShell vs Command Prompt

Category	Command Prompt (CMD)	PowerShell
Foundation	Legacy Windows CLI (MS-DOS heritage)	.NET-based shell & scripting language
Output type	Text (strings)	Objects
Pipeline	Text stream piping	Object pipeline (property-level operations)
Scripting	.bat/.cmd (limited)	.ps1 (functions/modules/classes)
Extensibility	External programs focused	.NET/WMI/COM/REST APIs
Common commands	dir, echo, ipconfig	Get-ChildItem, Write-Output, Get-NetIPAddress
Environment variable	echo %PATH%	\$env:PATH
Typical use	Simple execution & legacy compatibility	System administration & automation (DevOps)

Examples

List files: CMD → dir | PowerShell → Get-ChildItem (aliases: ls, dir)
Print PATH: CMD → echo %PATH% | PowerShell → \$env:PATH
Filter with pipeline: Get-Process | Where-Object { \$_.CPU -gt 100 }

Common Windows Shell Commands

Command	Description	Example
dir	List files and directories in current folder	dir C:\Users
cd [path]	Change directory (.. = up one level)	cd C:\Windows
mkdir [name]	Create a new folder	mkdir Projects
rmdir [name]	Remove a folder (must be empty)	rmdir Temp
del [file]	Delete a file	del notes.txt
copy [src] [dest]	Copy a file	copy a.txt D:\Backup\
move [src] [dest]	Move or rename a file	move old.txt new.txt

Useful tools

Command	Description	
doskey /history	A tool used in Command Prompt (cmd) to create command macros or to view the history of previously entered commands	
Where [file]	Displays the location of the executable file for a given command or program	

System & Info

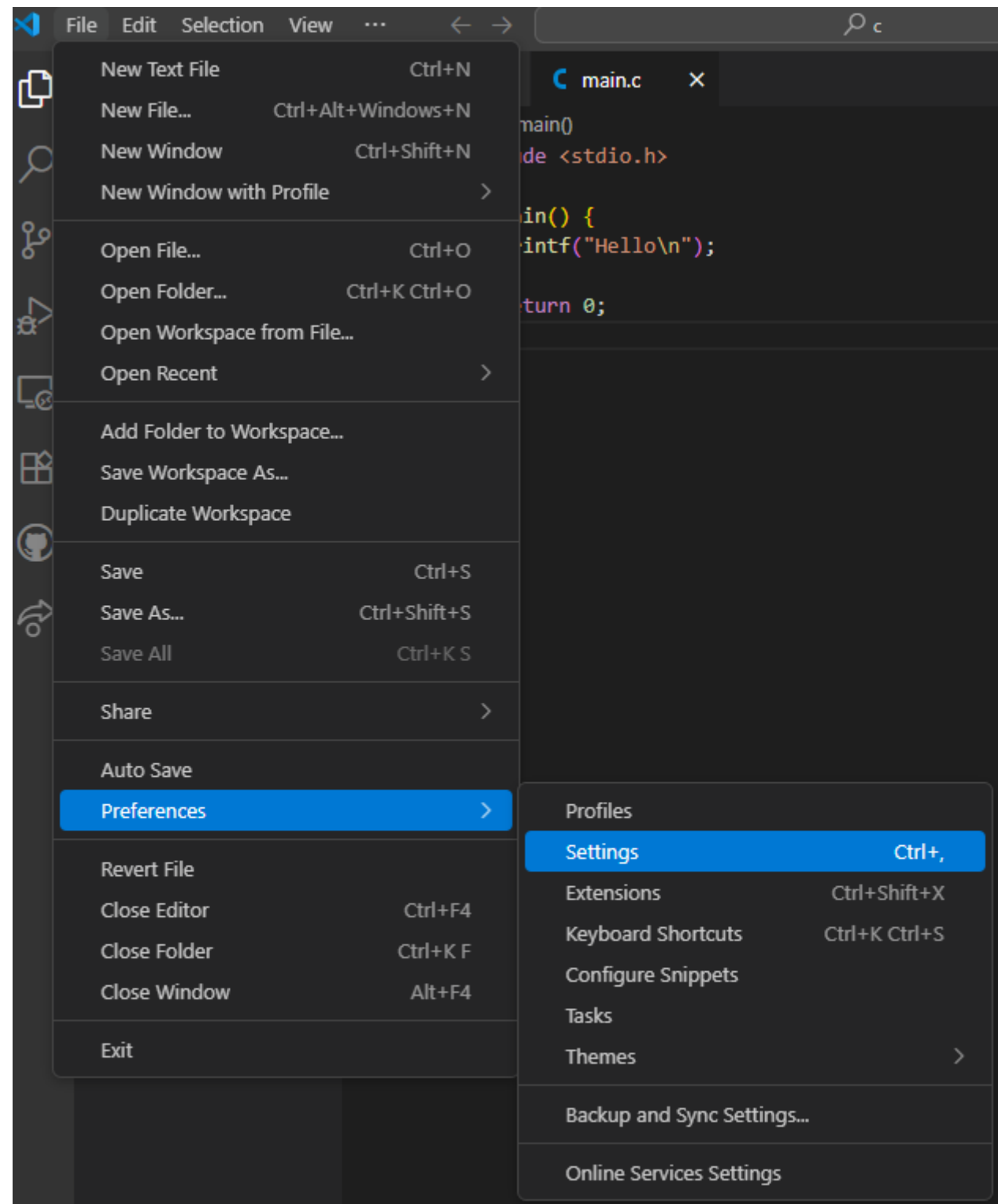
Command	Description	Example
cls	Clear screen	cls
echo [text]	Print text	echo Hello World
systeminfo	Show system info	systeminfo
hostname	Show computer name	hostname
ipconfig	Show IP/network info	ipconfig /all
ping [address]	Test network connection	ping google.com
tasklist	Show running processes	tasklist
taskkill /PID [id] /F	Kill process by ID	taskkill /PID 1234 /F

Networking

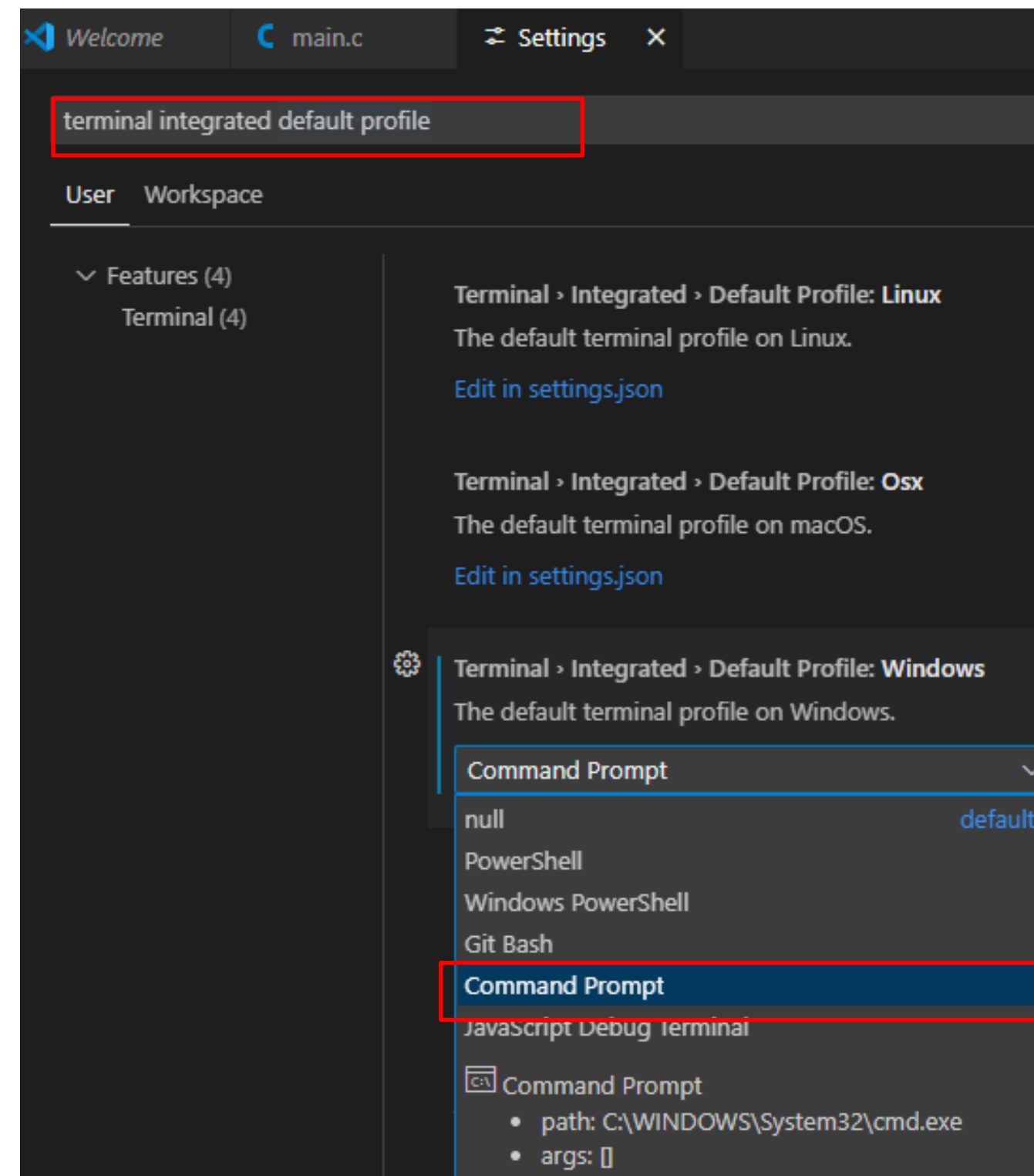
Command	Description	Example
netstat	Show active connections	netstat -an
tracert [addr]	Trace route	tracert google.com
nslookup [domain]	DNS lookup	nslookup openai.com

How to change default shell in VSC

. Setting > Terminal > default profile: Windows > Command Prompt



TYPE: terminal integrated default profile



Homework & Project with github classroom

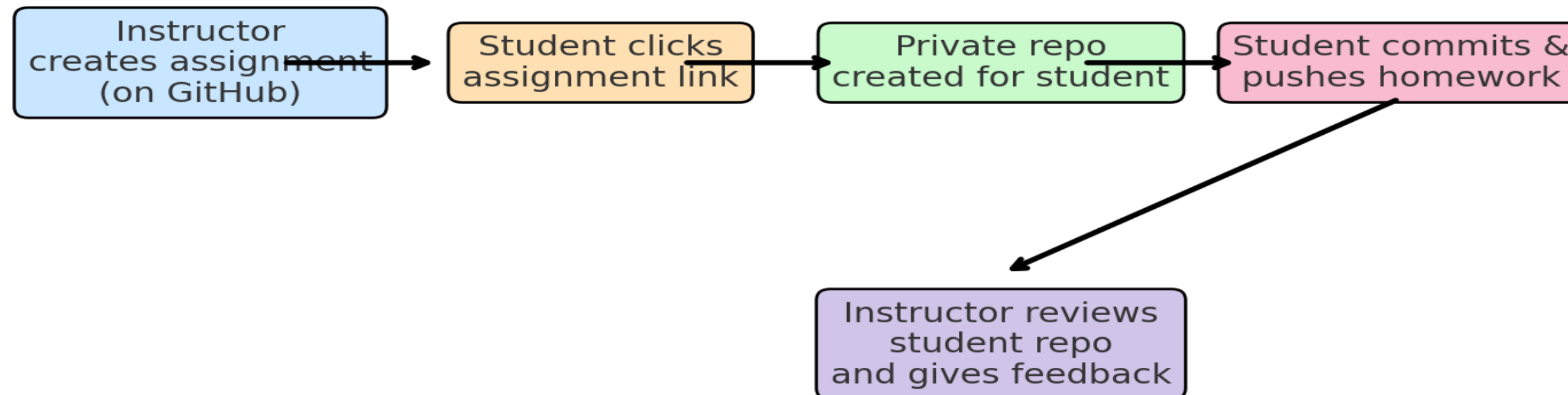
[Principle]

The **instructor** prepares the assignment on GitHub.

Each **student** clicks a special link and automatically gets their **own private repository**.

Students write code and upload it to GitHub, and the instructor can review it easily.

GitHub Classroom Workflow (Assignment Process)



Homework & Project with github classroom


[Method]

1. Click the assignment link and confirm that your **personal repository** has been created.
2. On the repository page,
 - Use “**Add file → Create new file**” to create a new file.
 - To upload files, choose “**Add file → Upload files**” and drag-and-drop your code.
 - To edit existing files, open the file and click the **pencil icon (Edit this file)**.
3. When saving, enter a **commit message** and click **Commit changes**.
4. All changes are automatically saved in the repository and visible to the instructor.
5. Students can complete assignments **entirely on GitHub's website** without cloning.

<https://classroom.github.com/a/hUI5vJ20>

GitHub Classroom


GitHub Education




You're ready to go!

You accepted the assignment, **Program with test**.


Your assignment repository has been created:

 <https://github.com/2025-Fall-C-Program/program01-tdkweon>

We've configured the repository associated with this assignment.

 Your assignment is due by **Sep 5, 2025, 18:51 UTC**

Note: You may receive an email invitation to join [2025-Fall-C-Program](#) on your behalf. No further action is necessary.



Join the GitHub Student Developer Pack

Verified students receive free GitHub Pro plus thousands of dollars worth of the best real-world tools and training from GitHub Education partners — for free. For more information, visit [GitHub Student Developer Pack](#).

Apply

2025-Fall-C-Program / program01-tdkweon

Code Issues Pull requests Actions Projects Security Insights

program01-tdkweon

Private

Watch 0

forked from 2025-Fall-C-Program/2025-fall-c-program-classroom-000-program-with-test-c-assignment-template-01

main

Go to file

Code

This branch is 3 commits ahead of 2025-Fall-C-Program/2025-fall-c-program-classroom-000-program-with-test-c-assignment-template-01:main

Contribute

Sync fork

tdkweon

Update main.c

e5d5afc · yesterday

.github/workflows	GitHub Classroom Autograding Workflow	yesterday
Makefile	Initial commit	yesterday
README.md	add deadline	yesterday
first_program.c	Initial commit	yesterday
main.c	Update main.c	yesterday

README

Review the assignment due date

c-assignment-template-01

Write the first my program

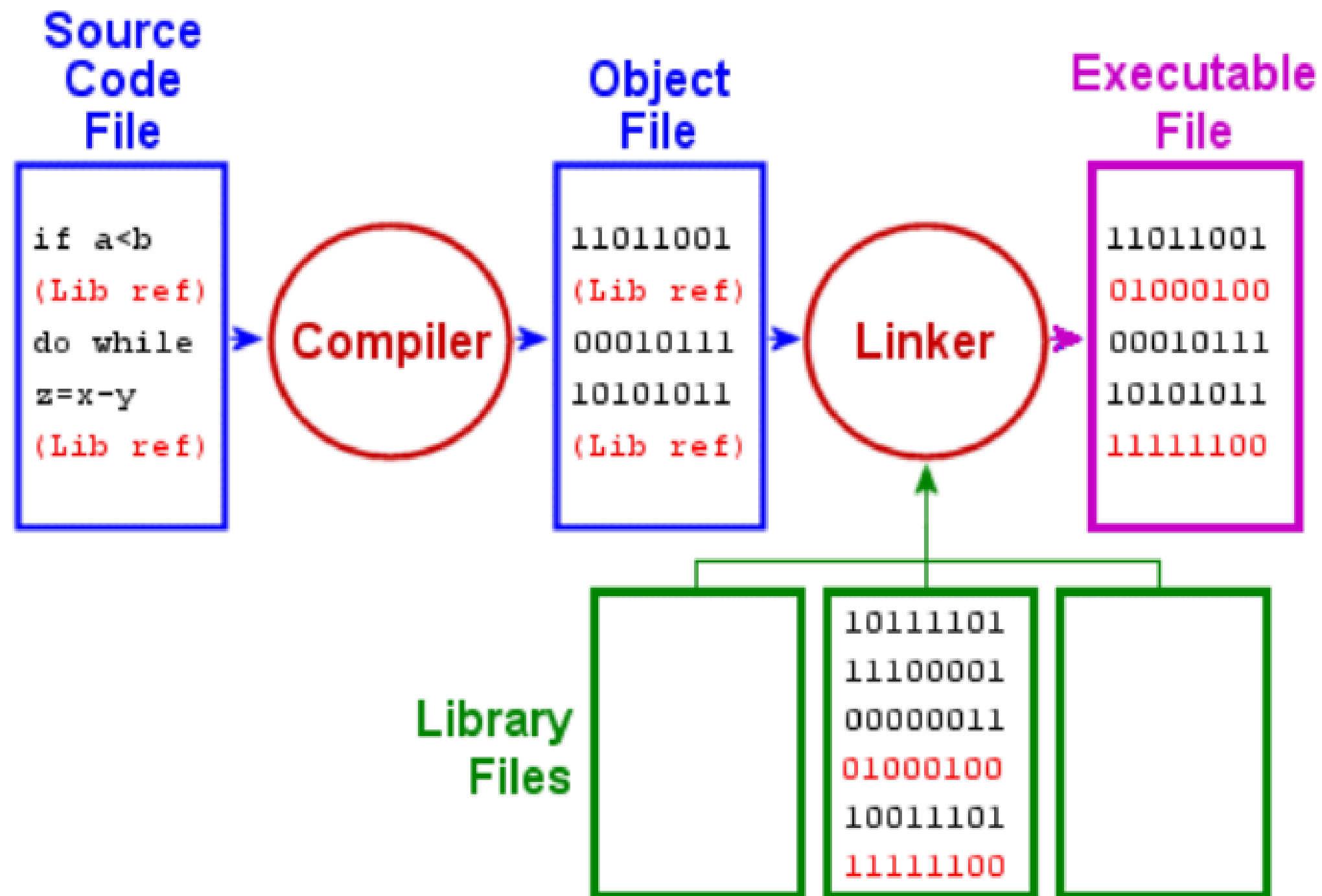
main.c

```
#include <stdio.h>
```

```
int main() {  
    printf("Hello world!\n");  
    return 0;  
}
```

What is compile?

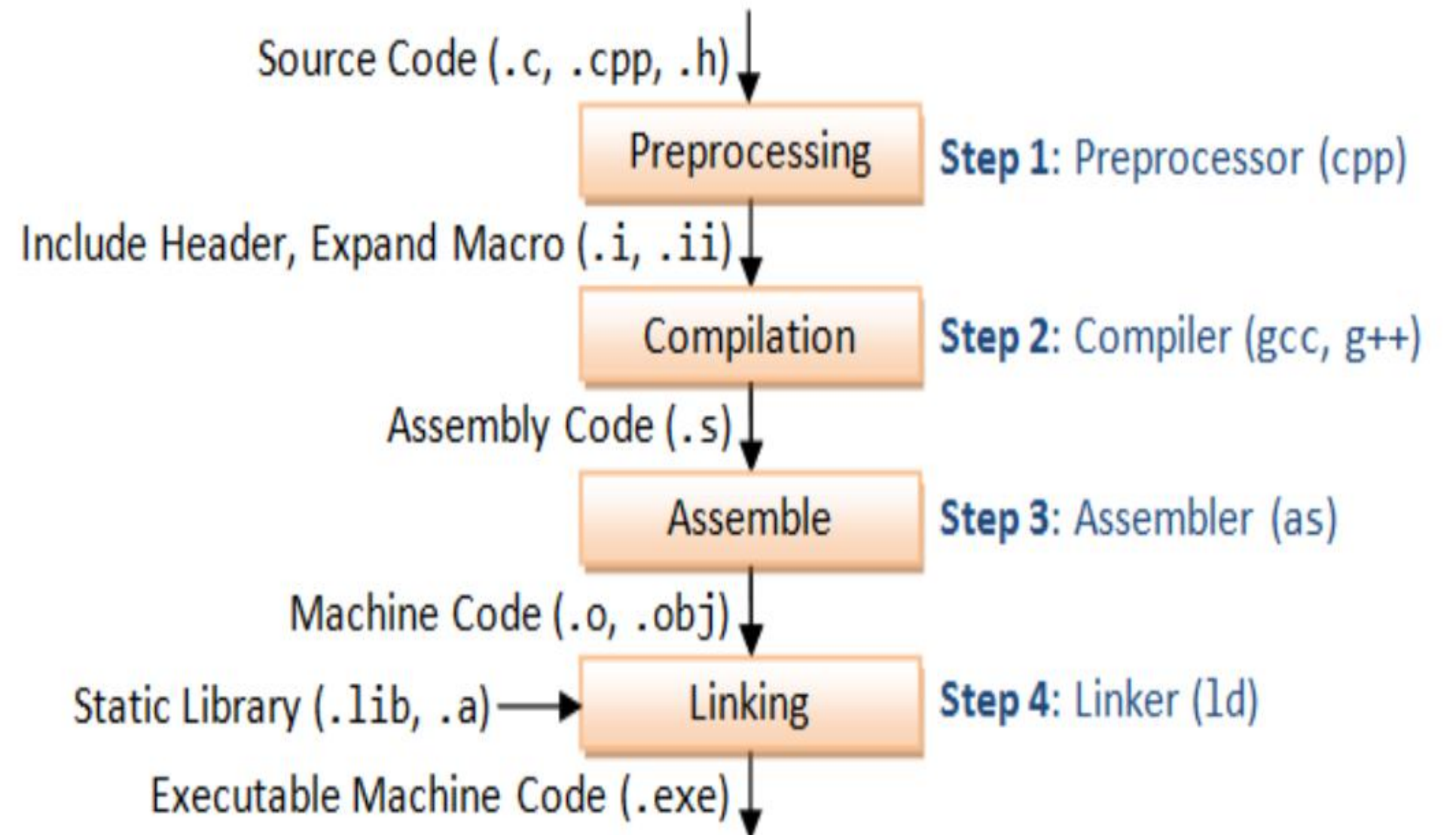
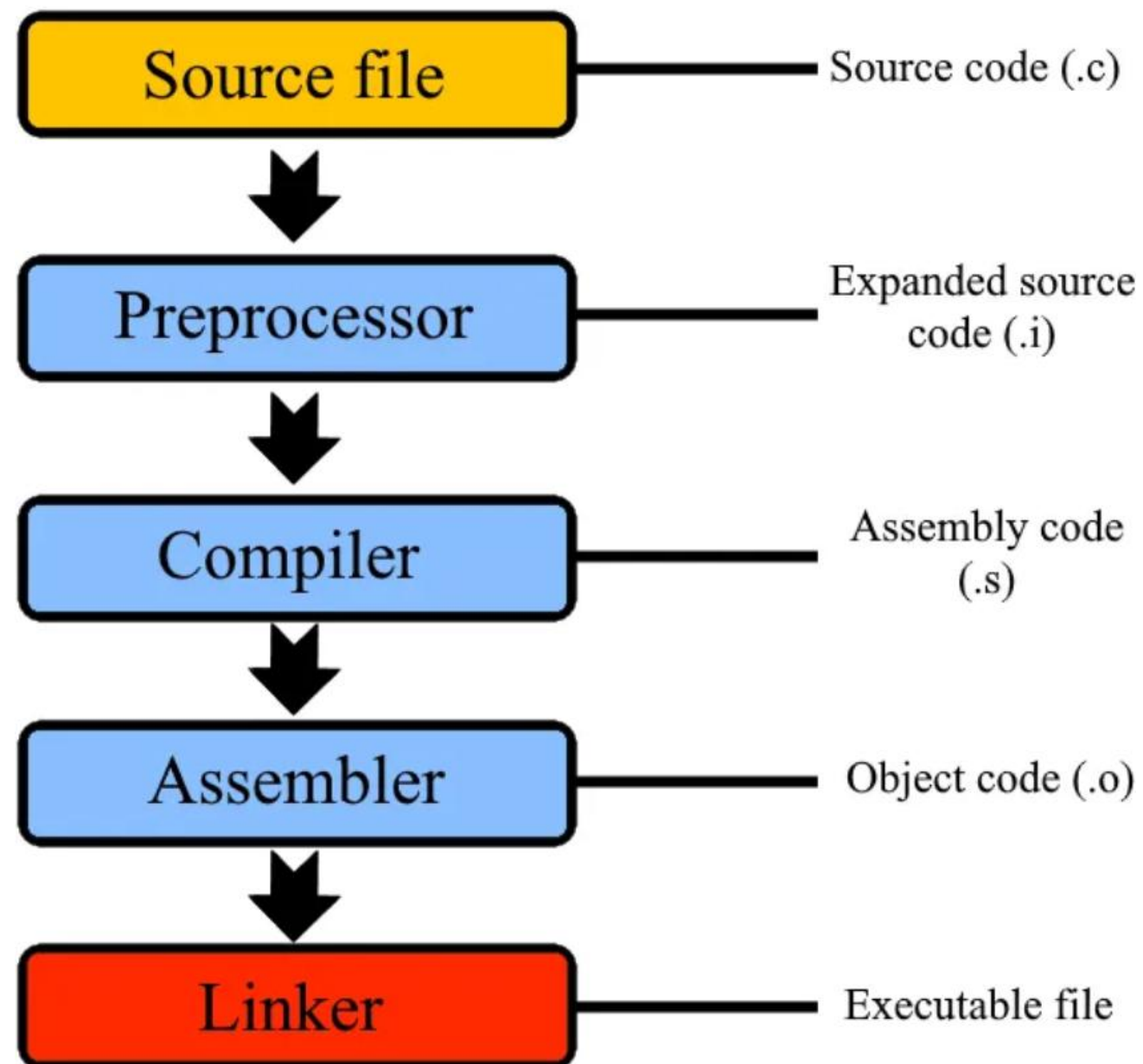
- Translating a high-level language created by humans into a low-level language that a computer can interpret.



Compile/Build Process

- The **Compile Process** is the sequence of steps that a compiler follows to convert **source code** (written in a high-level language like **C, C++, or Java**) into **machine code** (binary instructions that a computer can execute).

Source code : <https://github.com/lattera/glibc/tree/master/stdio-common>



Compile Process

1. main.c : Source code

> Preprocessor : Processes headers (#include) and macros (#define)

2. main.i : Intermediate file containing expanded source code

> Compiler : Edits preprocessed source code into assembly code for a specific processor

```
gcc -E main.c -o main.i
```

3. main.s : Assembly file

> Assembler : Converts assembly code into machine code

```
gcc -S main.i -o main.s
```

4. main.o : Object file

> Linker : Creates an executable file using the object file and library

```
gcc -c main.s -o main.o -> objdump -d main.o / nm main.o
```

5. main.exe : Executable file

```
gcc main.o -o main -> hexdump -C main.exe
```



gcc main.c -o main

Build the first my program

Use the -c flag with gcc to compile the source code into an object file without linking.

gcc -c main.c -o main.o => **main.o**

gcc main.o -o main.exe => **main.exe**

If you don't need an object file and just want an executable, omit the -c flag and use -o flag.

gcc main.c -o main.exe => **main.exe**

Development Environment & setup - Make the first my program

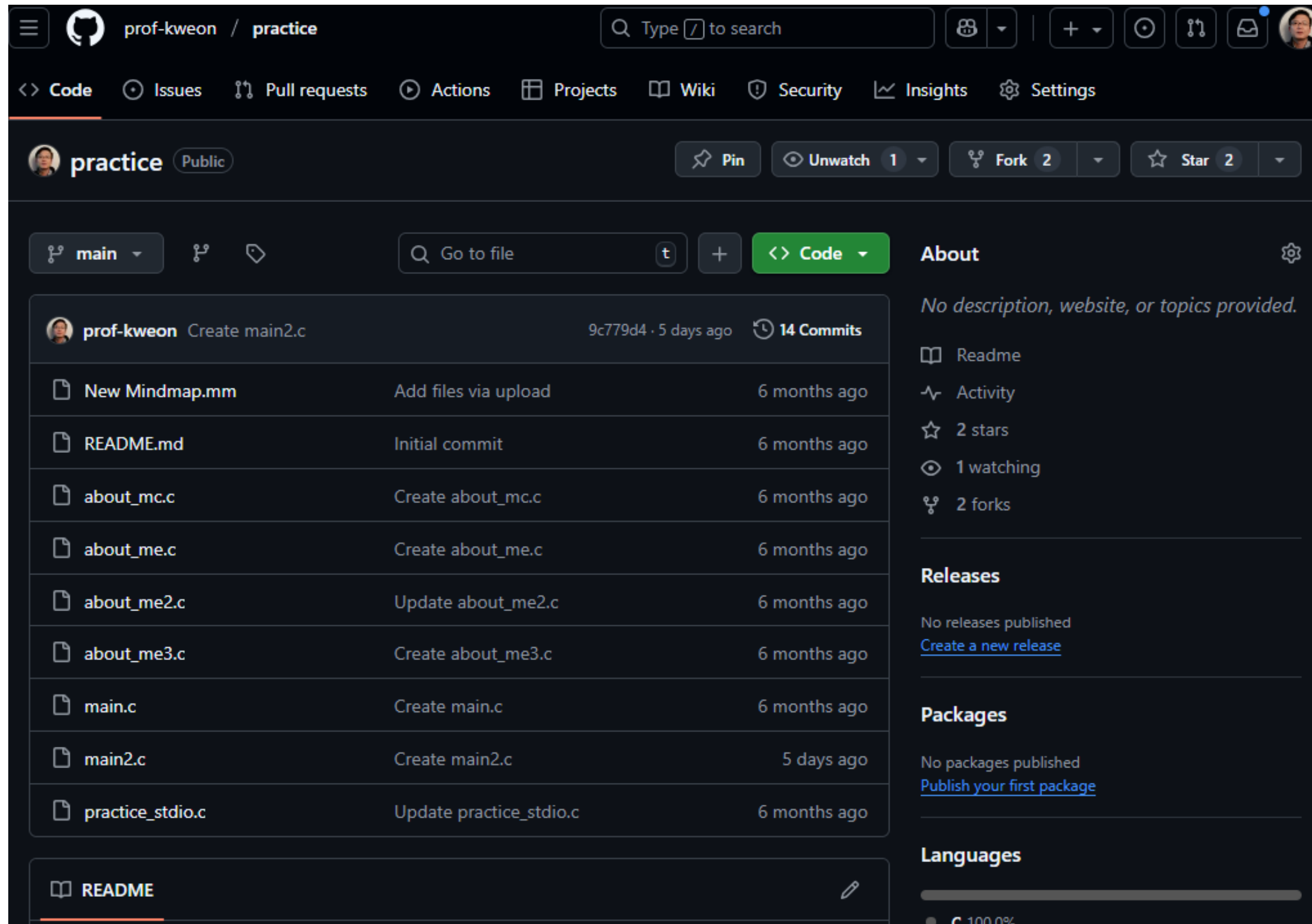
- 01** | @ VSCode (in terminal)
- 02** | @ Codespaces (For personal projects, Not use for homework)
- 03** | @Terminal & VSCode – git clone (For homework)

Development Environment & setup - Git

1. Download Git for Windows : <https://git-scm.com/download/win>
2. Run the Installer : For beginners, the default options are safe (just keep clicking “Next”).
3. Important Setup Options (Recommended)
 - Editor Selection: Choose VS Code
 - PATH Environment: Select “Git from the command line and also from 3rd-party software”.
 - HTTPS Transport: Use the default OpenSSH.
 - Line Ending Conversions: Choose “Checkout Windows-style, commit Unix-style” (recommended default).
4. After installation, you’ll have two main options: Git CMD / PowerShell:
 - Git also works from Windows Command Prompt
5. Verify Installation
`git --version`

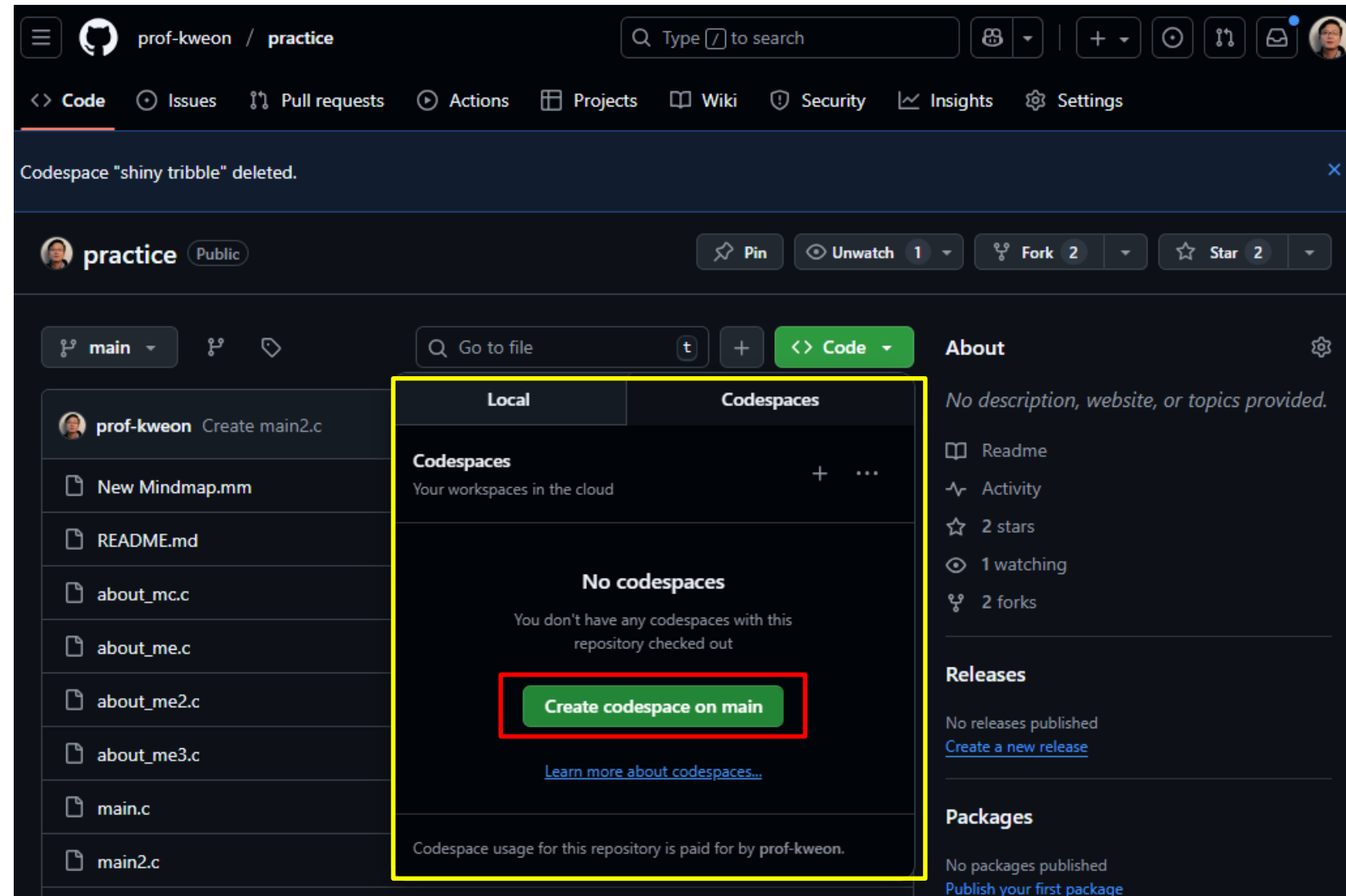
Development Environment & setup - Codespaces

1. Goto your personal repository of github

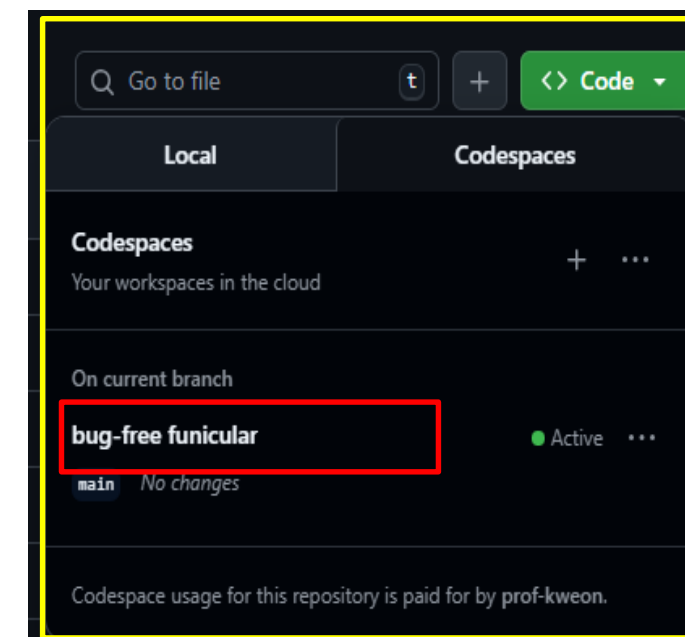


Development Environment & setup - Codespaces

2. Click “code”, select “Codespace” tab, then “Create codespace on main”

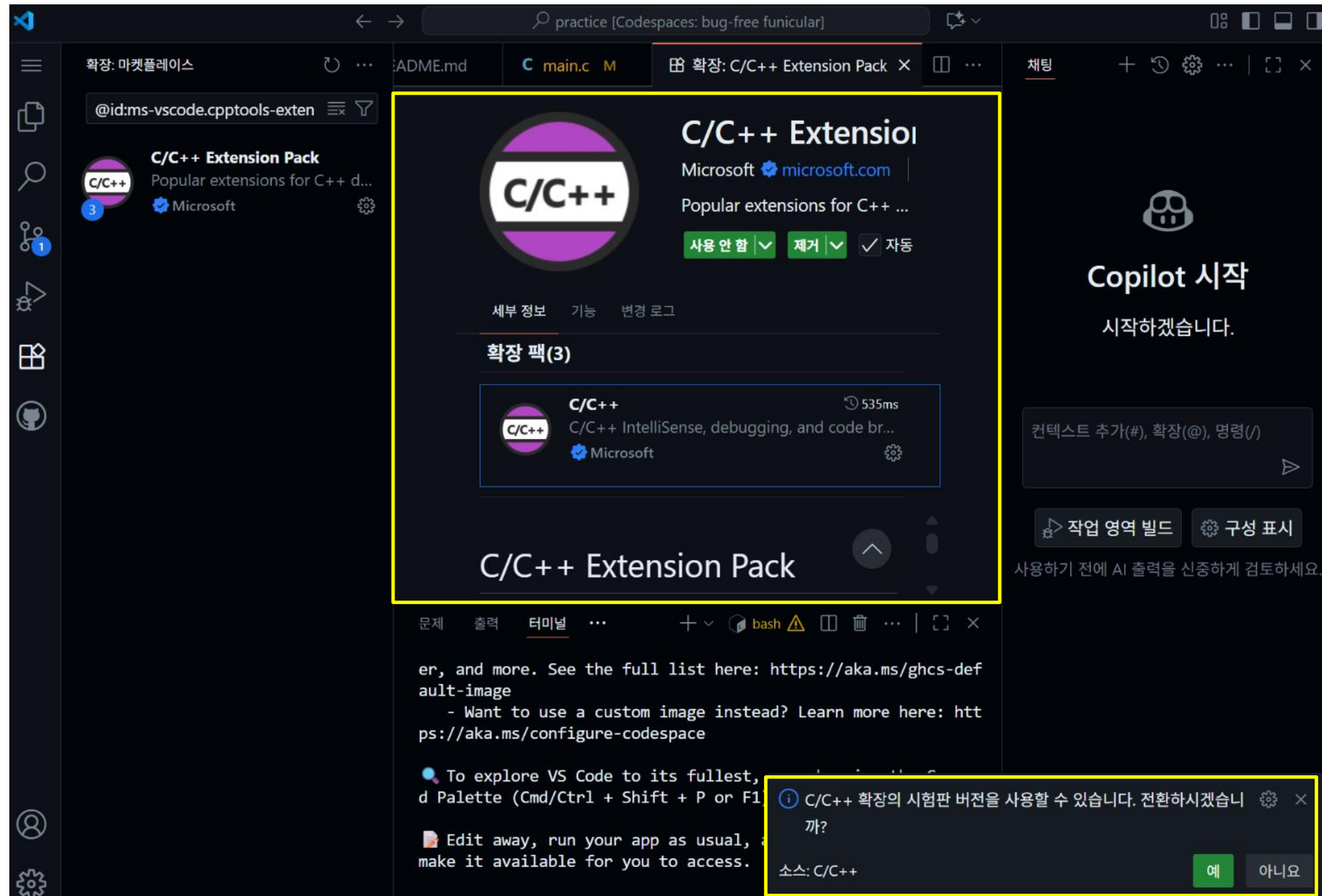


2-1. If you already created the codespace with certain repository, select “On current branch”



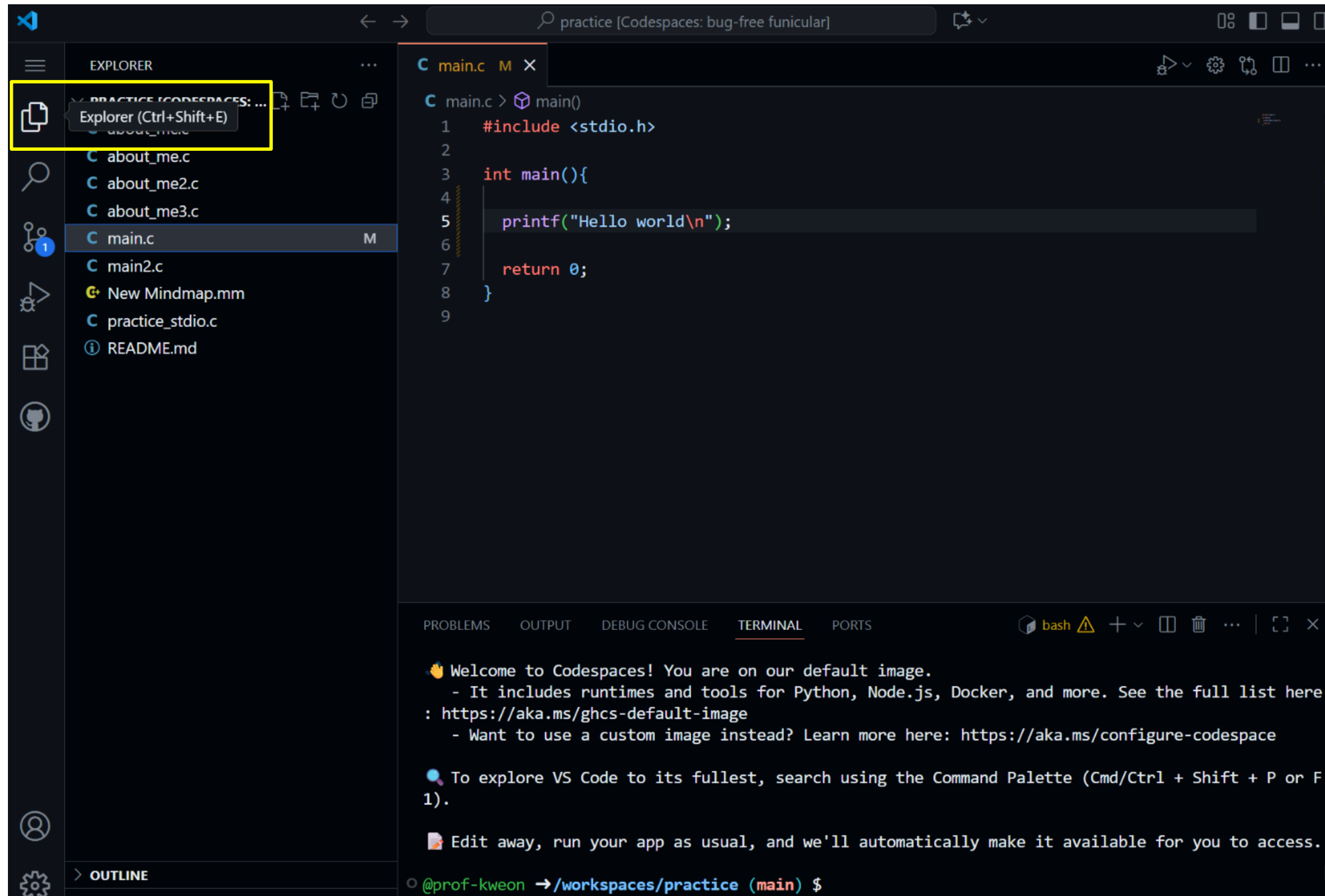
Development Environment & setup - Codespaces

3. When you got the popup messages from codespaces : install or switch C/C++ extension..... > install / accept them



Development Environment & setup - Codespaces

4. You can view the files in the current folder by clicking the **Explorer** button in the upper-left corner.



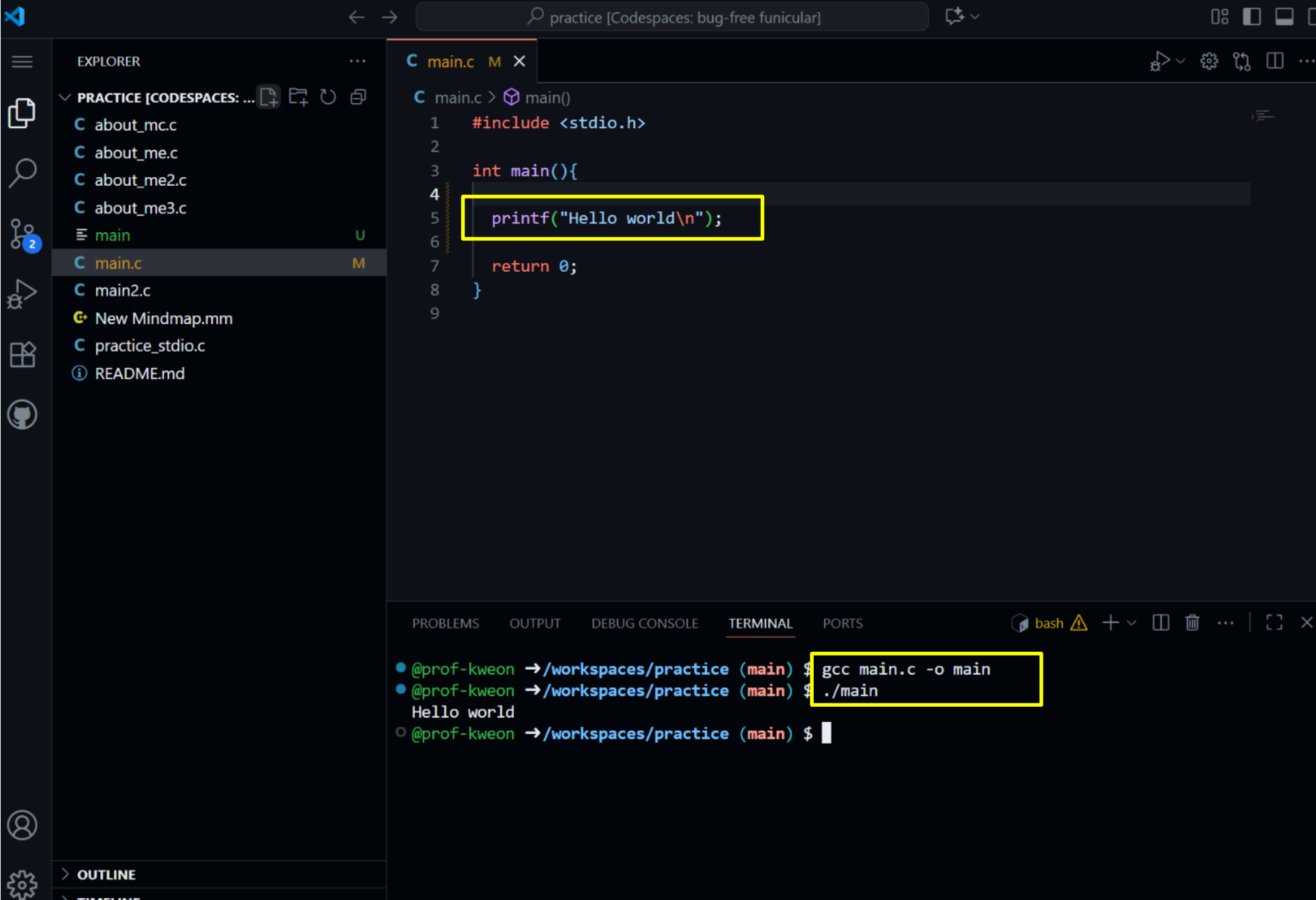
Development Environment & setup – Codespaces (Modify & Verify)

5. Now modify main.c (It might be saved when you compile the file. To make sure, press ctrl+s)

6. Compile & run main.c at terminal which is bash shell

```
gcc main.c -o main
./main
```

7. Verify your source codes

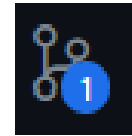


The screenshot shows a VS Code interface for a Codespace named 'practice [Codespaces: bug-free funicular]'. The Explorer sidebar on the left lists files: about_me.c, about_me2.c, about_me3.c, main, main.c, main2.c, New Mindmap.mm, practice_stdio.c, and README.md. The main editor displays the content of main.c, which includes `#include <stdio.h>`, `int main(){`, `printf("Hello world\n");`, and `return 0;`. The line containing the printf statement is highlighted with a yellow box. At the bottom, the Terminal panel shows a bash shell with the following commands and output: `gcc main.c -o main`, `./main`, and the output `Hello world`. The command `./main` is also highlighted with a yellow box.

Development Environment & setup – Codespaces (Commit - GUI)

8. Commit your source codes to the personal rep.

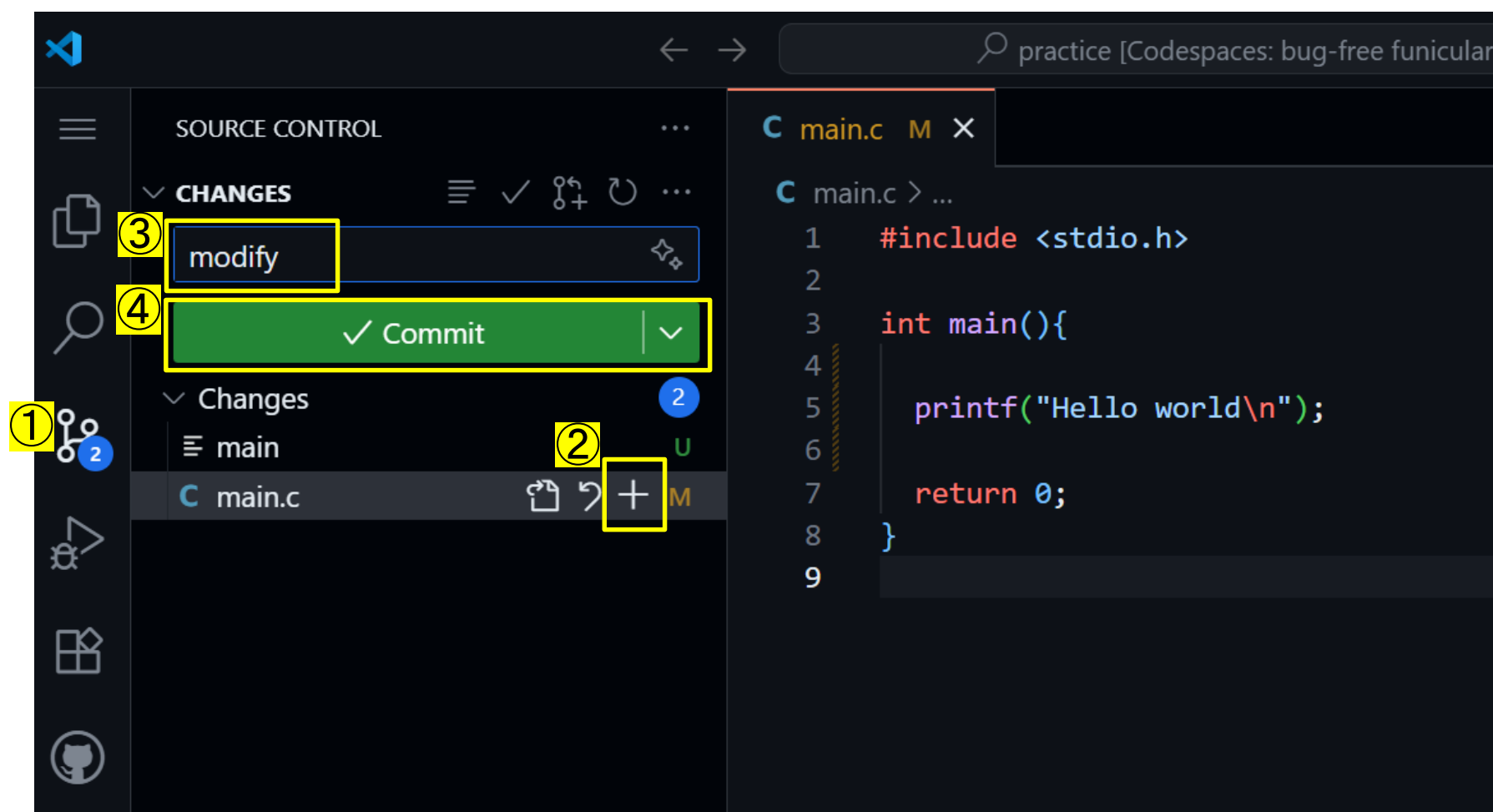
8-1. Press “Source Control” button



8-2. Press “+” button to commit

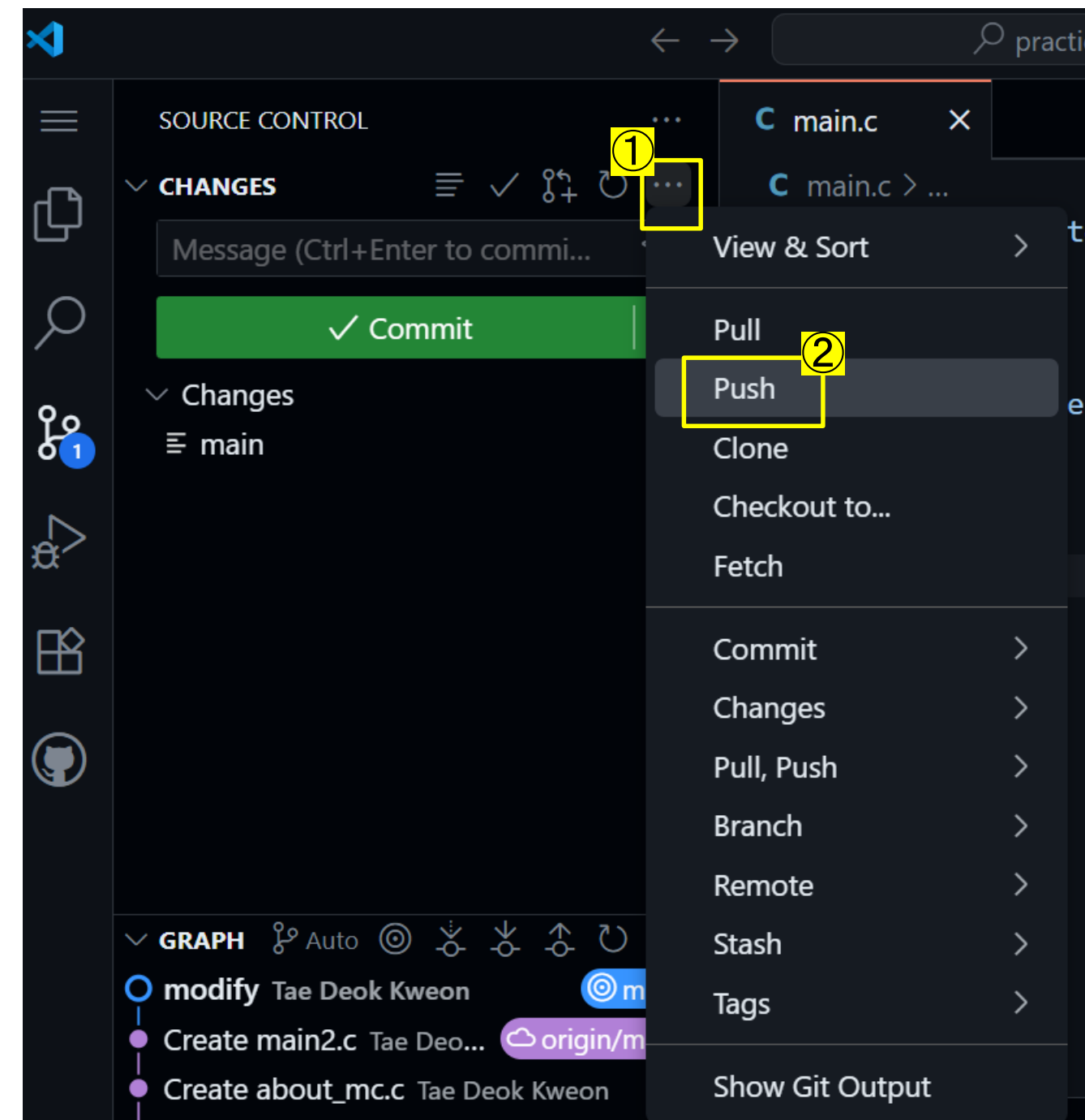
8-3. write any message for committing

8-4. Press “Commit” button



9. Press “...”, then “Push”

10. Close browser tab & reopen your github repo.



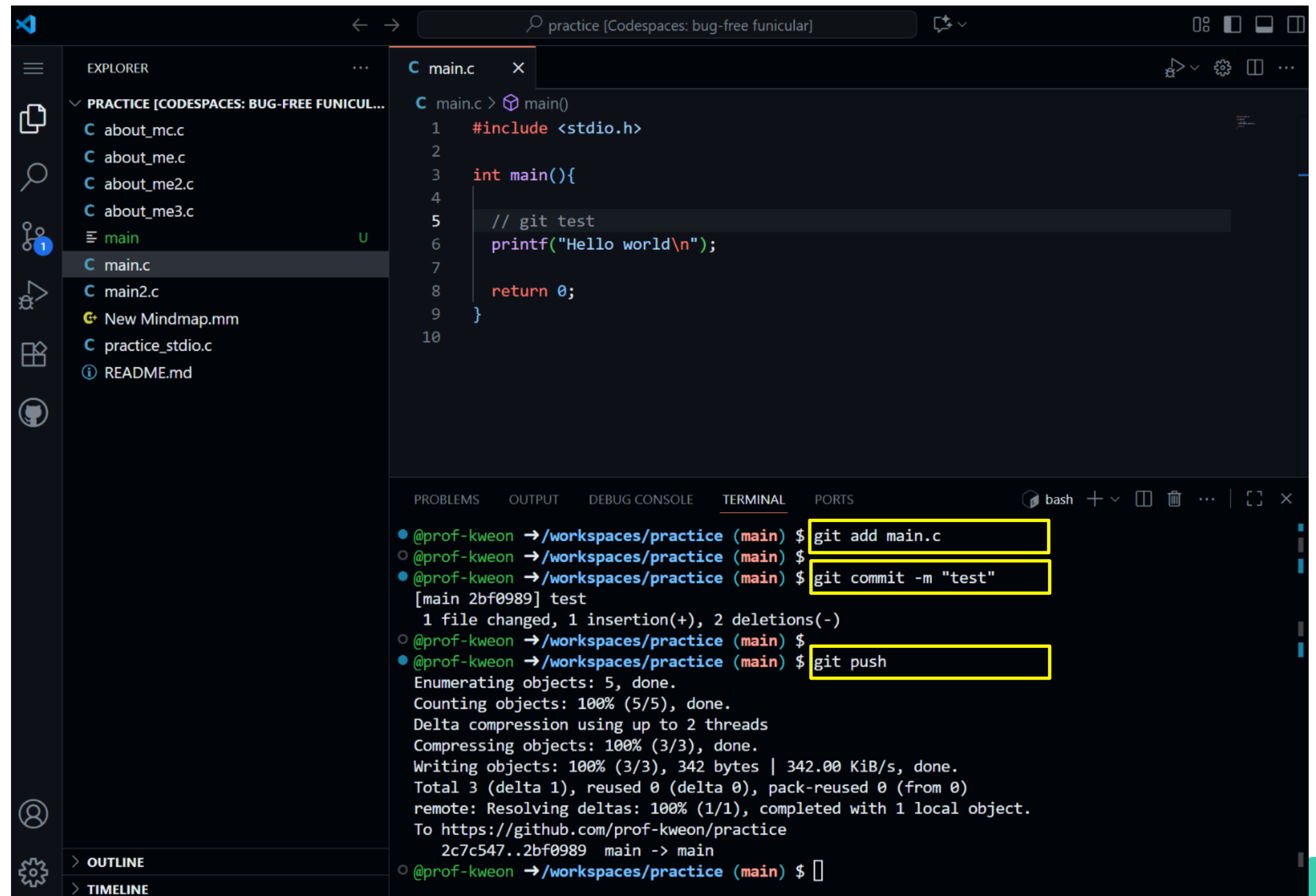
Development Environment & setup – Codespaces (Commit - CLI)

8. After modification & verification, type the following.

```
git add . (git add main.c)
```

```
git commit -m "msg"
```

```
git push
```



The screenshot shows a VS Code Codespace environment for a project named 'practice'. The Explorer panel on the left lists files: about_mc.c, about_me.c, about_me2.c, about_me3.c, main, main.c, main2.c, New Mindmap.mm, practice_stdio.c, and README.md. The main.c file is open in the editor, showing a C program that prints 'Hello world\n'. The terminal at the bottom shows the following commands and output:

```
@prof-kweon →/workspaces/practice (main) $ git add main.c
@prof-kweon →/workspaces/practice (main) $ git commit -m "test"
[main 2bf0989] test
1 file changed, 1 insertion(+), 2 deletions(-)
@prof-kweon →/workspaces/practice (main) $ git push
Enumerating objects: 5, done.
Counting objects: 100% (5/5), done.
Delta compression using up to 2 threads
Compressing objects: 100% (3/3), done.
Writing objects: 100% (3/3), 342 bytes | 342.00 KiB/s, done.
Total 3 (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/prof-kweon/practice
2c7c547..2bf0989 main -> main
@prof-kweon →/workspaces/practice (main) $
```

Development Environment & setup – git clone (Best way)

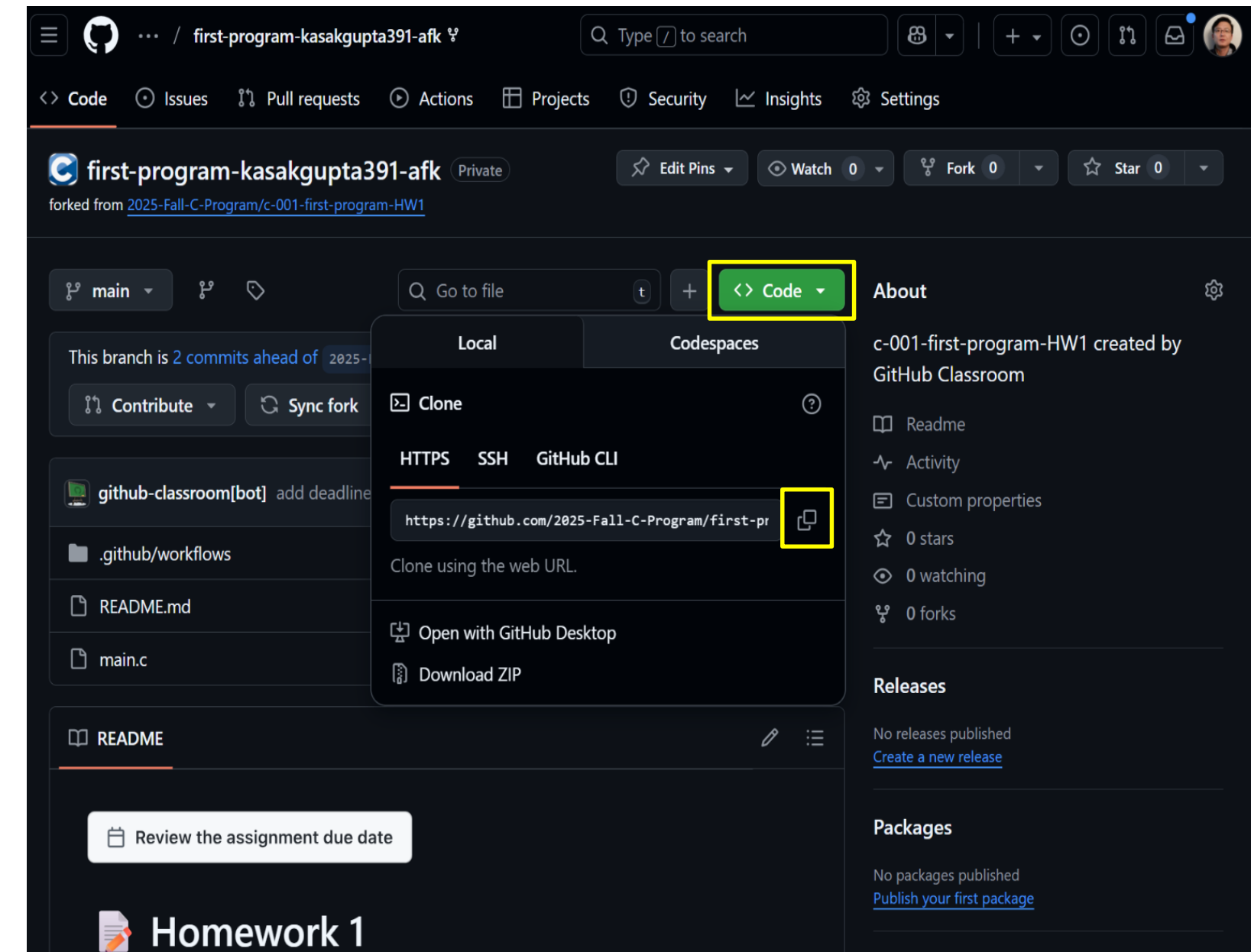
1. Go to your new assignment repository on GitHub.
2. Copy the Repository URL
 - Press “Code”, then copy url
ex) <https://github.com/2025-Fall-C-Program/first-program-kasakgupta391-afk.git>
3. Open **Terminal** (Linux/Mac) or **Git Bash** (Windows).
 - Goto proper folder: `cd c:\c-class`
 - Run: `git clone https://github.com/2025-Fall-C-Program/first-program-kasakgupta391-afk.git`
-> This creates a local folder with the same name as your repo.

```
D:\Projects\wsu>cd c
D:\Projects\wsu\c>dir
Volume in drive D has no label.
Volume Serial Number is E883-9CA4

Directory of D:\Projects\wsu\c

2025-03-05(수) 오후 01:34 <DIR>      .
2025-09-08(월) 오후 01:38 <DIR>      ..
2024-12-23(월) 오후 02:01 <DIR>      test_c
2025-03-05(수) 오후 01:34 <DIR>      the-c-programming-language-second-edition-solutions-master
0 File(s)              0 bytes
4 Dir(s)  1,909,492,338,688 bytes free

D:\Projects\wsu\c>git clone https://github.com/2025-Fall-C-Program/first-program-kasakgupta391-afk.git
Cloning into 'first-program-kasakgupta391-afk'...
remote: Enumerating objects: 14, done.
remote: Counting objects: 100% (2/2), done.
remote: Compressing objects: 100% (2/2), done.
remote: Total 14 (delta 0), reused 0 (delta 0), pack-reused 12 (from 2)
Receiving objects: 100% (14/14), 5.65 KiB | 263.00 KiB/s, done.
D:\Projects\wsu\c>
```



Development Environment & setup – git clone (Best way)

[Problem]

```
git clone https://github.com/2025-Fall-C-Program/basic-operations-\*\*\*\*\*.git
```

```
Cloning into 'basic-operations-*****' ...
```

```
remote: Repository not found.
```

```
fatal: repository 'https://github.com/2025-Fall-C-Program/basic-operations-\*\*\*\*\*.git/' not found
```

[Solution]

- Authentication

you must be **logged in to GitHub with the account that accepted the Classroom assignment.**

If you use HTTPS cloning, Git will ask for authentication the first time. If you are not logged in properly, you'll see "Repository not found".

- **Assignment not accepted**

In GitHub Classroom, each student must click the assignment link to generate their personal repo.

If you never accepted the assignment, your repo may not exist yet.

Development Environment & setup – git clone (Best way)

4. Check new folder after cloning repo.

- dir / ls :
- cd <new_folder> : go to new folder
- cd / pwd :check whole path of new folder)

```
D:\Projects\wsu\c>dir
Volume in drive D has no label.
Volume Serial Number is E883-9CA4

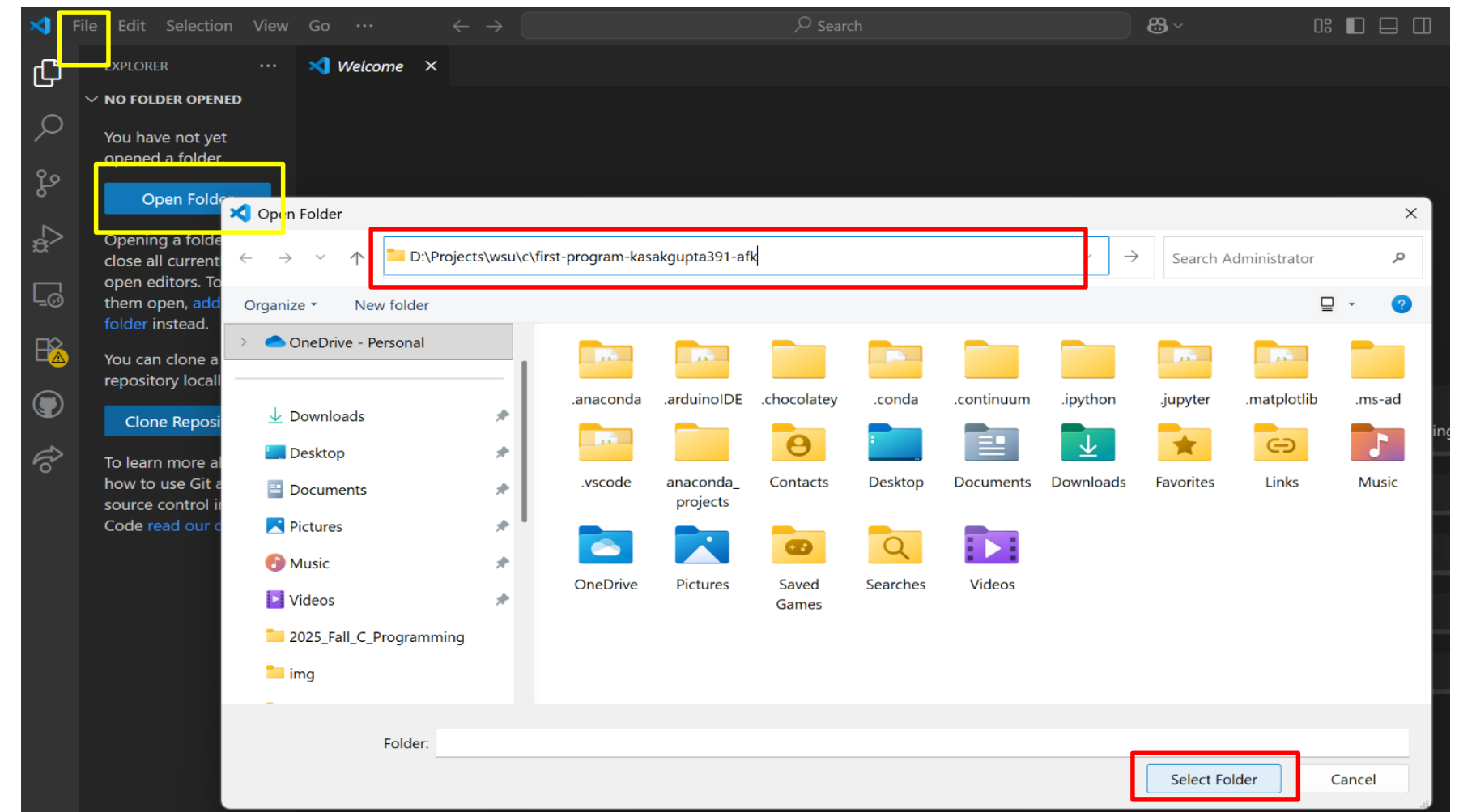
Directory of D:\Projects\wsu\c

2025-09-15(월)   오후 06:00   <DIR>          .
2025-09-08(월)   오후 01:38   <DIR>          .
2025-09-15(월)   오후 06:00   <DIR>          first-program-kasakgupta391-afk
2024-12-23(월)   오후 02:01   <DIR>          test_c
2025-03-05(수)   오후 01:34   <DIR>          the-c-programming-language-second-edition-solutions-master
               0 File(s)                0 bytes
               5 Dir(s)  1,909,491,998,720 bytes free

D:\Projects\wsu\c>
D:\Projects\wsu\c>cd first-program-kasakgupta391-afk
D:\Projects\wsu\c\first-program-kasakgupta391-afk>
D:\Projects\wsu\c\first-program-kasakgupta391-afk>cd
D:\Projects\wsu\c\first-program-kasakgupta391-afk>
D:\Projects\wsu\c\first-program-kasakgupta391-afk>
```

5. Open VSC & open cloned folder (new_folder).

- File > Open folder or Click “Open Folder”
- Find & write the cloned folder > “Select folder”

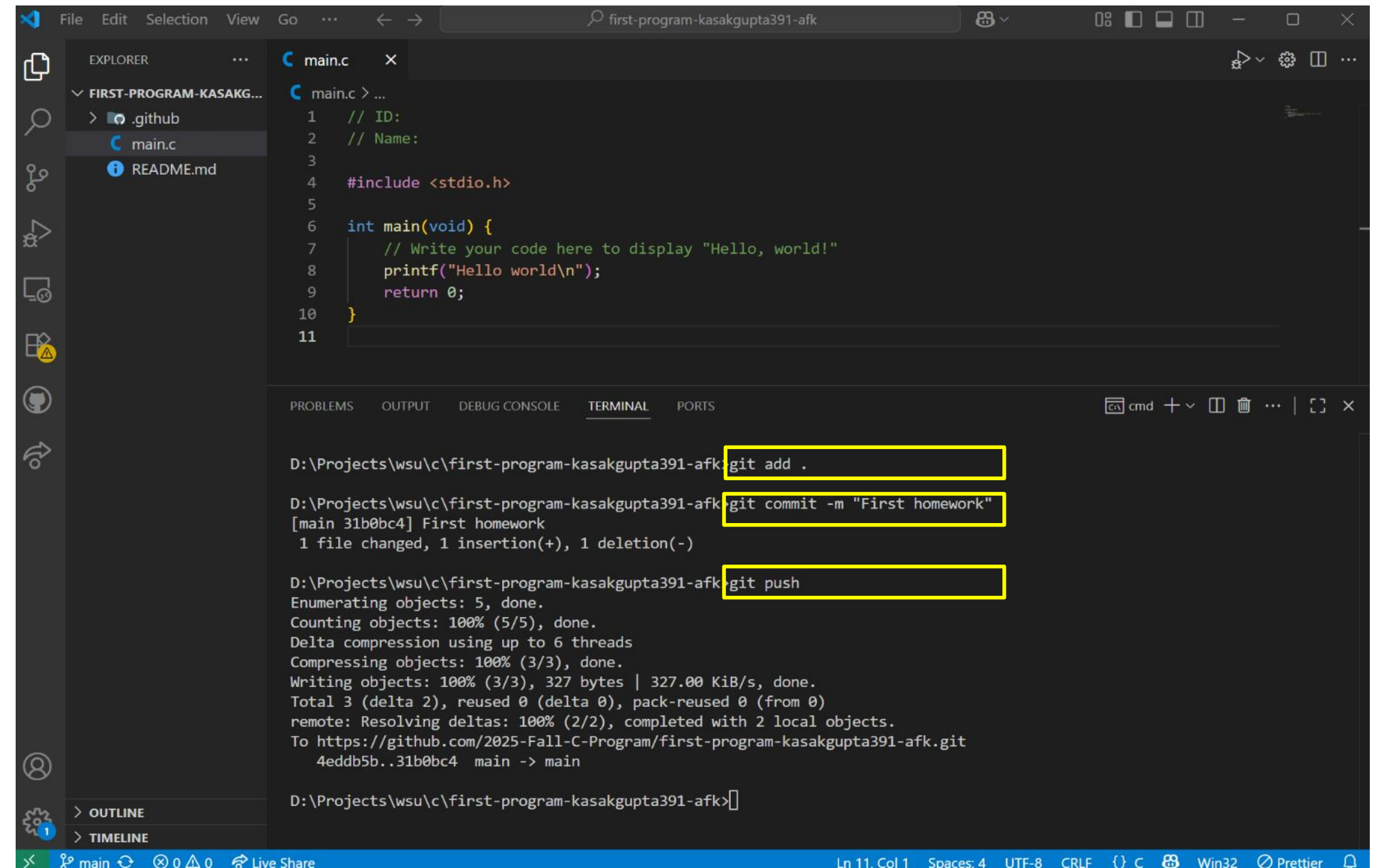
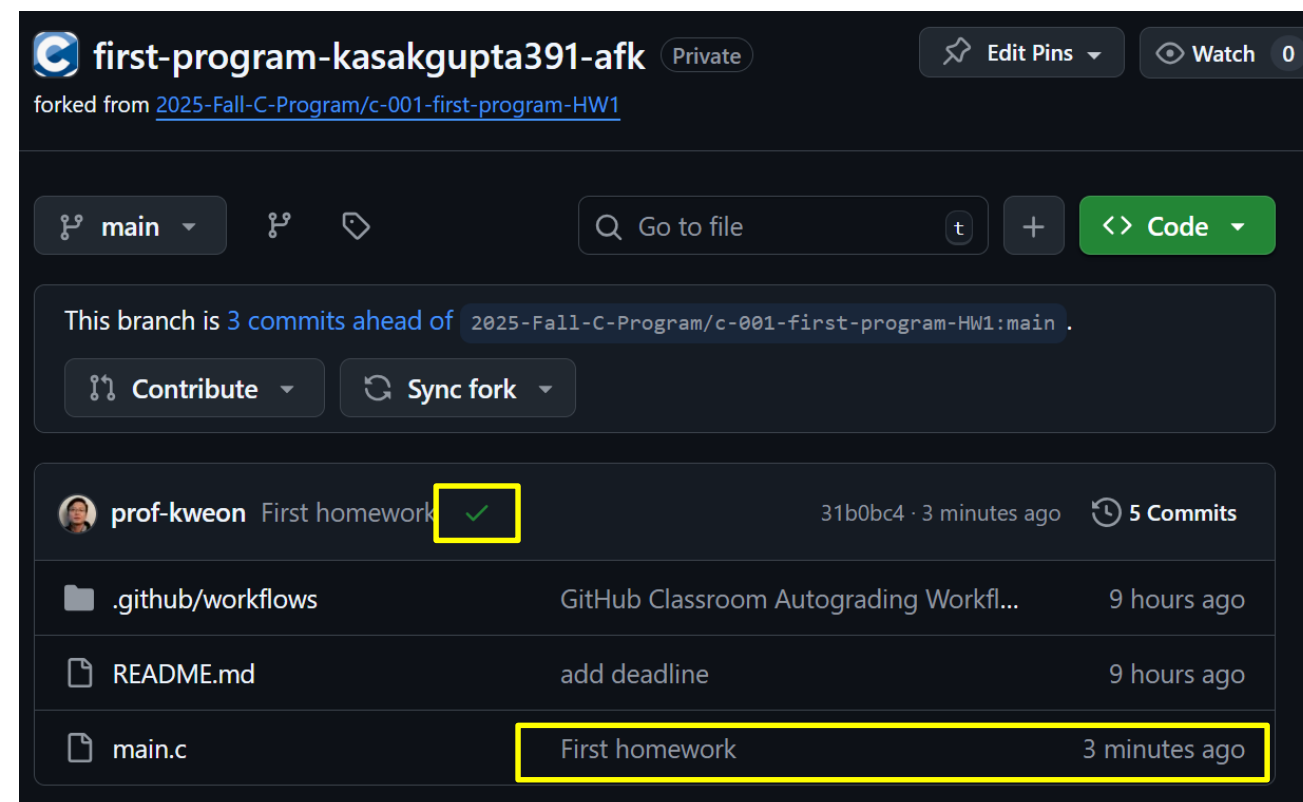


Development Environment & setup – git clone (Best way)

6. After modification & verification, type the following.

```
git add . (git add main.c)
git commit -m "msg"
git push
```

7. Verify if your code submitted with message & final result



- If a student modifies their cloned assignment repo, commits, and pushes back to their **private repo**, GitHub Classroom will automatically run the auto-grader and record the score.

Development Environment & setup – git clone (Optional, Not use for HW)

1. User Identity Setup (Only once per environment)

```
git config --global user.name "Your Name"
git config --global user.email "your_email@example.com"
```

2. Repository Initialization (if starting from scratch. Not using clone) : **Not needed** if you already cloned from GitHub (Classroom assignments)

```
git init
```

3. Check Repository Status

```
git status
```

4. Commit first message

```
git commit -m "Write a clear commit message"
```

5. Connect to Remote Repo (if not already connected)

```
git remote add origin https://github.com/username/repo.git
```

6. Upload the source codes into repo (local commit > Github repo)

```
git push -u origin main
```

Scenario	Commands Needed
First-time setup (new machine or empty Codespace)	git config --global user.name "Name" git config --global user.email "email@example.com" git init (if no repo yet)
Regular workflow (every assignment submission)	git status → git add . → git commit -m "message" → git push

Development Environment & setup – git clone (Best way)

Summary workflow

1. Accept assignment link
2. Copy repository URL
3. `git clone <URL>`
4. Edit → Save → Compile & Run locally
5. Update all source codes into assignment repo

`git add .`

`git commit -m "message"`

`git push`

6. Check the result

Compiler vs. Interpreter

Compiler is a program that translates the entire source code of a programming language into machine code (binary) **before execution**. The resulting executable file can be run independently without requiring the original source code.

Examples of compiled languages:

C, C++, Rust, Go

Characteristics:

- Translates the entire program at once. (Modify-> Compile all again)
- Generates a separate executable file.
- Faster execution since the program is already compiled.
- Errors are detected before execution.

Compiler vs. Interpreter

Interpreter is a program that translates and executes code **line by line** at runtime. It does not generate a separate executable file; instead, it processes the source code dynamically.

Examples of interpreted languages:

Python, JavaScript, Ruby, PHP

Characteristics:

- Translates and runs the code **line by line**.
- No separate executable file; the interpreter is needed each time.
- Slower execution compared to compiled programs.
- Errors are detected **during execution** (runtime).

Compiler vs. Interpreter

Key Differences

Feature	Compiler	Interpreter
Execution Speed	Fast (precompiled)	Slow (line-by-line)
Error Detection	Before execution	During execution
Output	Executable file	No separate executable
Usage	C, C++, Rust	Python, JavaScript

Some languages, like **Java**, use both:

- Java source code is compiled into bytecode (.class file).
- The JVM (Java Virtual Machine) interprets the bytecode at runtime.

Compiled from "Hello.java"

```
public class Hello {  
  public Hello();  
    Code:  
      0: aload_0  
      1: invokespecial #1    // Method java/lang/Object."<init>":()V  
      4: return  
  
  public static void main(java.lang.String[]);  
    Code:  
      0: getstatic   #2      // Field  
java/lang/System.out:Ljava/io/PrintStream;  
      3: ldc         #3      // String Hello, world!  
      5: invokevirtual #4     // Method  
java/io/PrintStream.println:(Ljava/lang/String;)V  
      8: return  
}
```

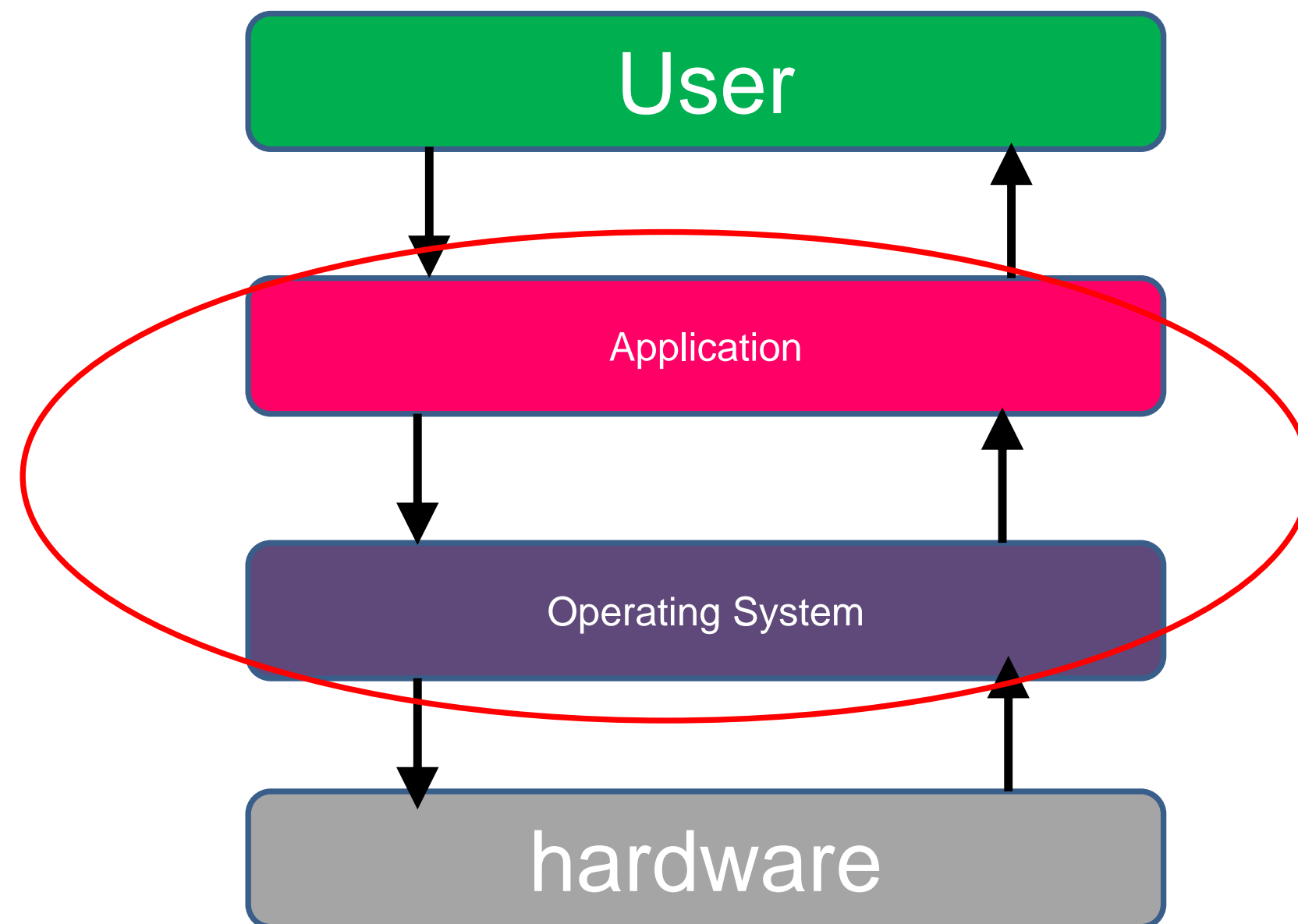
Compiler vs. Interpreter

	Compiler	Interpreter
HTML/CSS	X	X (Browser's direct interpretation)
JavaScript	X	O (JS Engine in browser interpretation)
C/C++	O	X

What if there is no program on the computer ?

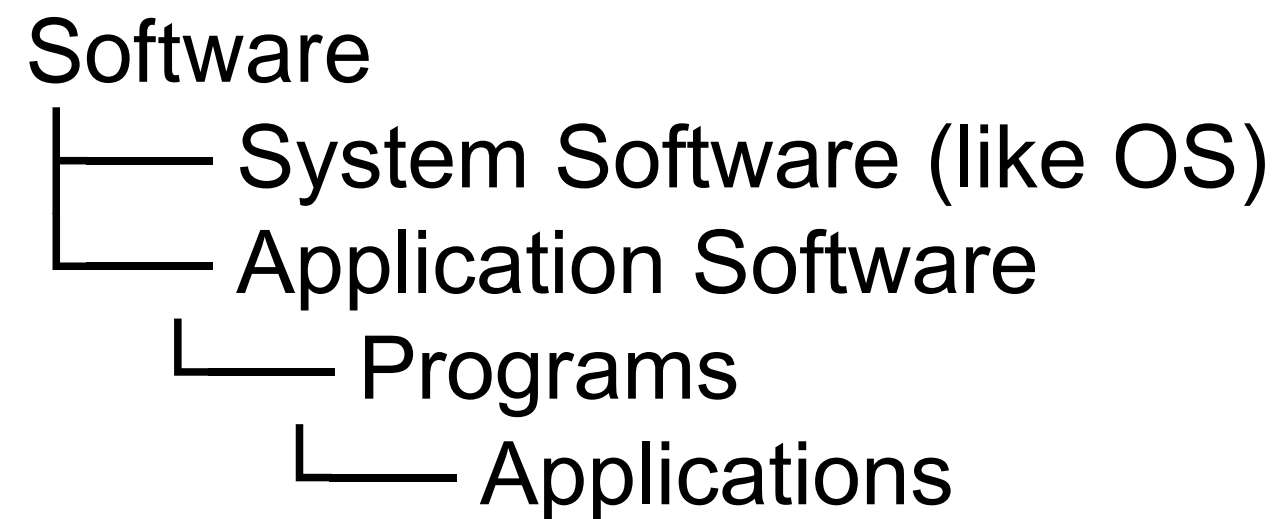
Without computer hardware and no programs, a computer is just a useless machine that generates some heat and noise .

“ Windows” and additionally installing various applications .



Software vs. Program vs. Application

- Software: the umbrella term for all code/data that runs on a computer.
- Program: a single executable unit—a set of instructions the computer can run.
- Application (App): a program built for end-user tasks.



Software vs. Program vs. Application

Category	Software	Program	Application
Definition	A broad term that includes all programs and data running on a computer	A set of instructions written to perform a specific task	A type of software designed for end users to perform specific tasks
Scope	Includes operating systems, drivers, utilities, and applications	A subset of software that consists of executable code	A subset of software that provides user-oriented functionality
Purpose	Manages hardware, provides system functionality, and supports applications	Performs a specific operation or task within a system	Enables users to complete tasks like document editing, communication, or browsing
User Interaction	Can run in the background (e.g., OS, drivers) or be user-facing	May or may not be user-facing (e.g., a script or background process)	Always designed for direct user interaction
Examples	Windows, macOS, Linux, firmware, database software	A simple Python script, a sorting algorithm, a file copy script	Microsoft Word, Google Chrome, Instagram

See you next week!

DO NOT miss the classes

