Homework 9

Software Engineering Principle

Software Engineering Program,

Department of Computer Engineering,

School of Engineering, KMITL

67011352 Theepakorn Phayonrat

Features

Gantt Chart

Used for planning project plan and tasks duration or deadline.

Class Diagram

Used for designing classes in the projects.

Interaction Diagram

Used for designing how classes interact each others in the projects.

Markdown Renderer for the task assignment page

How it works:

- As mentioned earlier, we can use markdown to express the task, $\therefore$ we need a markdown renderer.

Implementation Approach:

- Use QEngineWebView Module in PyQt.

VS-Code Extension (OPTIONAL)

`TODO` extension in VS-Code with better description for the task and with team member(s) assigned to that task.
How it works:

- If you have comment with `TODO` in the front, you can add description of the task in a different entry and also in a markdown file.
- If you want to add a person in charge for that task (OPTIONAL), you can use `@TEMP`, where `TEMP` can be either role or team member names.
- After saved, you can access the `TODO` description as you hover and click to inspect task in the comment.

Implementation Approach:

- Scan through the file looking for comment with `TODO` in the front then keep the entry into the DB.
- We can edit the `TODO` description inside a external markdown file.

Page included in this homework

- **Task Assignment Page**: Page to edit `TODO` for task assignments with markdown supported for better view. User can choose whether to edit in the manual mode or external text editor and save file because it can also fetch from real `.md` files in the real program.

Code:

HW9_67011352_Theepakorn.html

```html
<!DOCTYPE html>
<html lang="en">
    <head>
        <meta charset="UTF-8">
        <meta name="viewport" content="width=device-width, initial-scale=1">
        <title></title>
        <link rel="stylesheet" href="https://pyscript.net/releases/2026.2.1/core.css" />
        <script type="module" src="https://pyscript.net/releases/2026.2.1/core.js"></script>
    </head>
    <body style="padding: 0; margin: 0">
    <div id="output"></div>
    <py-config>
        packages = ["markdown"]
    </py-config>
    <script type="py" src="HW9_67011352_Theepakorn.py"></script>
    </body>
</html>
```

HW9_67011352_Theepakorn.py

```
import js
import markdown
from abc import ABC, abstractmethod
from pyodide.ffi import create_proxy
from pyscript import document

colors = {
    "white": "#FFFFFF",
    "light_gray": "#AAAAAA",
    "yellow": "#FFD77F",
    "blue": "#32A5DA",
    "dark_blue": "#158BC1"
}

assets_path = {
    "logo_img": "./images/logo.png",
    "notification_snd": "./sounds/notification.wav"
}


class AbstractWidget(ABC):
    def __init__(self, element_id, root) -> None:
        self.element_id = element_id
        self._element = None
        self.root = root
        self.mode = "Manual"
        self.css = """
        <style>
            #preview_output table {
                border-collapse: collapse;
                width: 100%;
            }

            #preview_output th,
            #preview_output td {
                border: 1px solid #444;
                padding: 6px 10px;
            }
            #preview_output th {
                background: #f2f2f2;
            }
        </style>
        """

    @property
    def element(self):
        """Return the DOM Element"""
        if not self._element:
            self._element =
               ↪  document.querySelector(f"#{self.element_id}")
        return self._element

    @abstractmethod
    def drawWidget(self) -> None:
        pass


    def get_color(self, choice: str) -> str | None:
        global colors
        return colors.get(choice)

    def get_asset_path(self, path: str) -> str | None:
        global assets_path
        return assets_path.get(path)


    def get_mode(self):
        return self.mode

    @abstractmethod
    def toggle_mode(self, event):
        pass

class IndexHTML(AbstractWidget):
    def __init__(self, element_id, root):
        AbstractWidget.__init__(self, element_id, root)
        self.mode = "Manual"

    def toggle_mode(self, event):
        _ = event
        if self.get_mode() == "Manual":
            self.mode = "Sync"
        else:
            self.mode = "Manual"
        self.status_value.innerText = self.get_mode()

    def drawWidget(self) -> None:

        # NavBar
        self.navbar = document.createElement("div")
        self.navbar.style.height = "7.5vh"
        self.navbar.style.width = "100vw"
        self.navbar.style.display = "flex"
        self.navbar.style.flexDirection = "row"
        self.navbar.style.alignItems = "center"
        self.navbar.style.justifyContent = "space-between"
        self.navbar.style.backgroundColor = self.get_color("blue")

        ## NavBar Left
        self.navbar_left = document.createElement("div")
        self.navbar_left.style.width = "22.5rem"
        self.navbar_left.style.display = "flex"
        self.navbar_left.style.flexDirection = "row"
        self.navbar_left.style.justifyContent = "space-between"
        self.logo_img = document.createElement("img")
        self.logo_img.src = self.get_asset_path("logo_img")
        self.logo_img.style.height = "50px"
        self.status_div = document.createElement("div")
        self.status_div.style.width = "10rem"
        self.status_div.style.display = "flex"
        self.status_div.style.flexDirection = "row"
        self.status_div.style.justifyContent = "space-between"
        self.status_label = document.createElement("h2")
        self.status_label.innerText = "Mode:"
        self.status_value = document.createElement("h2")
        self.status_value.innerText = self.get_mode()

        ## NavBar Right
        self.navbar_right = document.createElement("div")
        self.mode_toggle_btn = document.createElement("button")
```

```
self.mode_toggle_btn.innerText = "Toggle Mode"
self.mode_toggle_btn.style.padding = "5px 5px"
self.mode_toggle_btn.style.margin = "10px"
self.mode_toggle_btn.onclick = self.toggle_mode

# Main Div
self.main_div = document.createElement("div")
self.main_div.style.height = "92.5vh"
self.main_div.style.display = "flex"
self.main_div.style.flexDirection = "row"

## Code Area
self.code_div = document.createElement("div")
self.code_div.id = "code_div"
self.code_div_widget = CodeArea("code_div", root=self)
self.code_div_widget._element = self.code_div
self.code_div_widget.drawWidget()


## Preview Area
self.preview_div = document.createElement("div")
self.preview_div.style.width = "54.5vw"
self.preview_div.style.display = "flex"
self.preview_div.style.flexDirection = "column"
self.preview_div.style.padding = "10px"
self.preview_label = document.createElement("h1")
self.preview_label.innerText = "Preview:"
self.preview_area = document.createElement("div")
self.preview_area.id = "preview_output"
self.preview_area.style.width = "100%"
self.preview_area.style.height = "82.5vh"
self.preview_area.style.padding = "2rem"
self.preview_area.style.overflowY = "auto"
self.preview_area.style.backgroundColor =
   ↪  self.get_color("white")

# appendChild

self.status_div.appendChild(self.status_label)
self.status_div.appendChild(self.status_value)
```

```
        self.navbar_left.appendChild(self.logo_img)
        self.navbar_left.appendChild(self.status_div)

        self.navbar_right.appendChild(self.mode_toggle_btn)

        self.navbar.appendChild(self.navbar_left)
        self.navbar.appendChild(self.navbar_right)

        self.preview_div.appendChild(self.preview_label)
        self.preview_div.appendChild(self.preview_area)

        self.main_div.appendChild(self.code_div)
        self.main_div.appendChild(self.preview_div)

        self.element.appendChild(self.navbar)
        self.element.appendChild(self.main_div)

class CodeArea(AbstractWidget):
    def __init__(self, element_id, root) -> None:
        AbstractWidget.__init__(self, element_id, root)

    def toggle_mode(self, event):
        _ = event
        if self.get_mode() == "Manual":
            self.mode = "Sync"
        else:
            self.mode = "Manual"

    def file_handle(self, event):
        _ = event  # unused
        file = self.file_input.files.item(0)
        if not file:
            return

        reader = js.FileReader.new()

        def onload(_):
            self.code_area.value = reader.result

        reader.onload = onload
        reader.readAsText(file)
```

```
def render_convert(self, event):
    _ = event
    md_text = self.code_area.value

    html = markdown.markdown(
        md_text,
        extensions=["fenced_code", "tables", "toc"]
    )

    # send to IndexHTML
    self.root.preview_area.innerHTML = self.css + html

def save_file(self, event):
    _ = event
    js.alert("File Saved")
    _ = js.Audio.new(self.get_asset_path("notification_snd")).play()


def drawWidget(self):
    ## Code Area
    self.code_div_inner = document.createElement("div")
    self.code_div_inner.style.width = "40vw"
    self.code_div_inner.style.display = "flex"
    self.code_div_inner.style.flexDirection = "column"
    self.code_div_inner.style.padding = "10px"
    self.code_label = document.createElement("h1")
    self.code_label.innerText = "Code:"
    self.code_area = document.createElement("textarea")
    self.code_area.id = "code_input"
    self.code_area.style.width = "100%"
    self.code_area.style.height = "80vh"
    self.code_area_btn_div = document.createElement("div")
    self.code_area_btn_div.style.width = "100%"
    self.code_area_btn_div.style.display = "flex"
    self.code_area_btn_div.style.flexDirection = "row"
    self.code_area_btn_div.style.justifyContent = "space-around"
    self.file_input = document.createElement("input")
    self.file_input.id = "md_file_input"
```

```
        self.file_input.type = "file"
        self.file_input.style.flex = "1"
        self._file_change_proxy = create_proxy(self.file_handle)
        self.file_input.addEventListener("change",
           ↪  self._file_change_proxy)
        self.convert_btn = document.createElement("button")
        self.convert_btn.innerText = "Convert"
        self.convert_btn.style.flex = "1"
        self.convert_btn.id = "convert_btn"
        self._convert_proxy = create_proxy(self.render_convert)
        self.convert_btn.addEventListener("click",
           ↪  self._convert_proxy)
        self.save_btn = document.createElement("button")
        self.save_btn.innerText = "Save"
        self.save_btn.style.flex = "1"
        self.save_btn.onclick = self.save_file


        # appendChild

        self.code_area_btn_div.appendChild(self.file_input)
        self.code_area_btn_div.appendChild(self.convert_btn)
        self.code_area_btn_div.appendChild(self.save_btn)

        self.code_div_inner.appendChild(self.code_label)
        self.code_div_inner.appendChild(self.code_area)
        self.code_div_inner.appendChild(self.code_area_btn_div)

        self.element.appendChild(self.code_div_inner)


if __name__ == "__main__":
    body = document.querySelector("body")
    body.style.display = "flex"
    body.style.flexDirection = "column"
    body.style.fontFamily = "monospace"
    body.style.backgroundColor =  colors.get("yellow")
    output = document.querySelector("#output")
    output_widget = IndexHTML("output", None)
    output_widget.drawWidget()
```

Image:

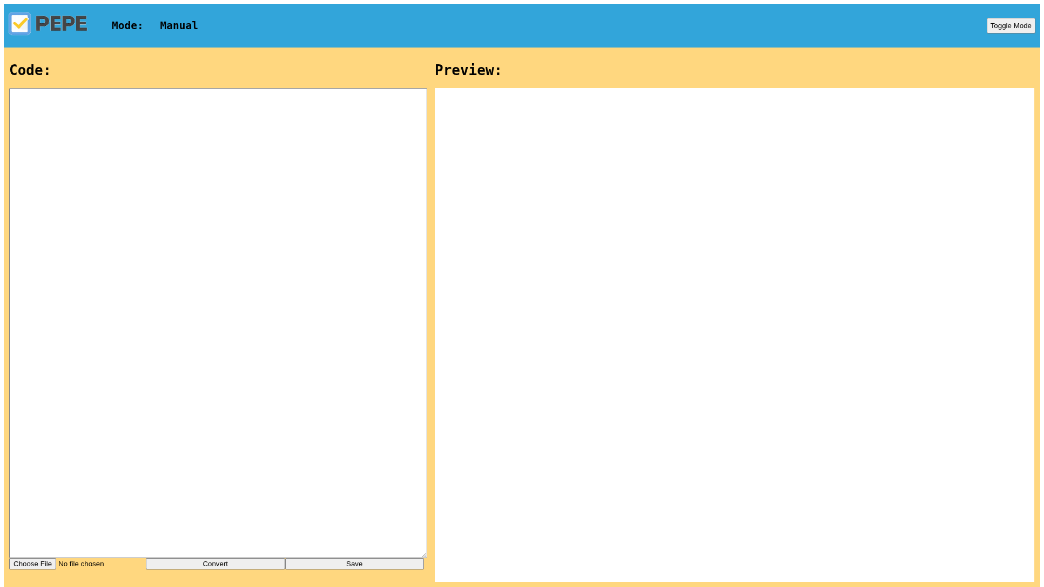
`logo.png`

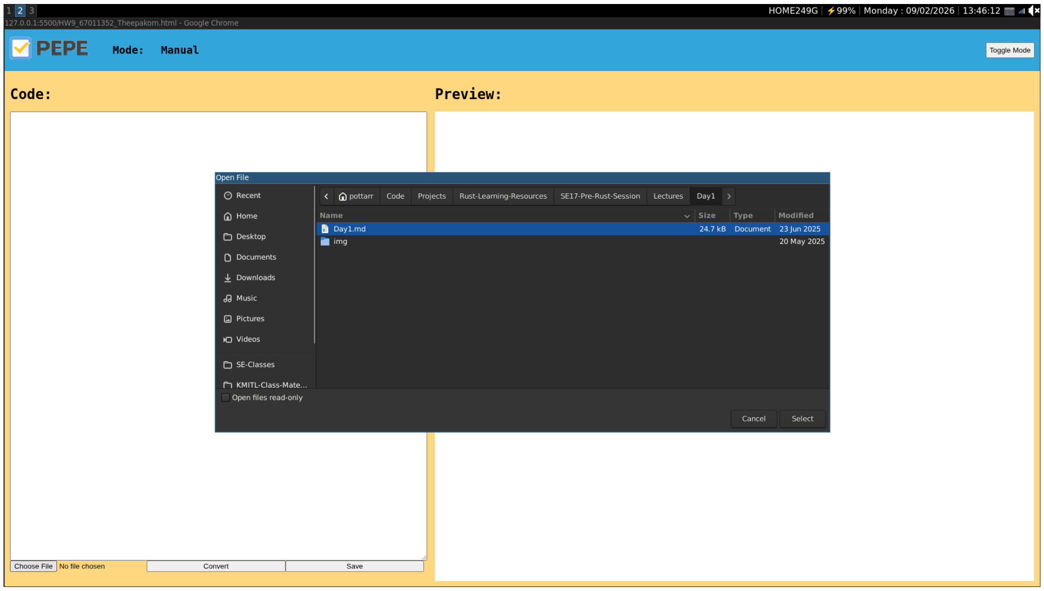
Sound:

- `notification.wav`

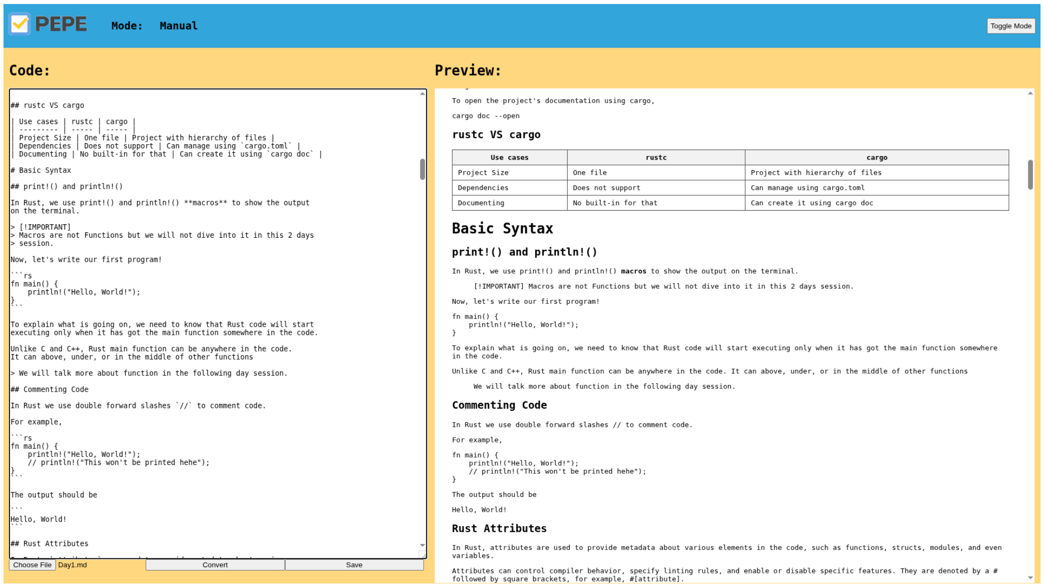
Output:

Task Assignment Page





Name	Size	Type	Modified
Day1.md	24.7 kB	Document	23 Jun 2025
img			20 May 2025



Mode: Manual

~~~
## rustc VS cargo

| Use cases | rustc | cargo |
| --------- | ----- | ----- |
| Project Size | One file | Project with hierarchy of files |
| Dependencies | Does not support | Can manage using `cargo.toml` |
| Documenting | No built-in for that | Can create it using `cargo doc` |

# Basic Syntax

## print!() and println!()

In Rust, we use print!() and println!() **macros** to show the output
on the terminal.

> [!IMPORTANT]
> Macros are not Functions but we will not dive into it in this 2 days
> session.

Now, let's write our first program!

```rs
fn main() {
    println!("Hello, World!");
}
```

To explain what is going on, we need to know that Rust code will start
executing only when it has got the main function somewhere in the code.

Unlike C and C++, Rust main function can be anywhere in the code.
It can above, under, or in the middle of other functions

> We will talk more about function in the following day session.

## Commenting Code

In Rust we use double forward slashes `//` to comment code.

For example,

```rs
fn main() {
    println!("Hello, World!");
    // println!("This won't be printed hehe");
}
```

The output should be

```
Hello, World!
```

## Rust Attributes
~~~

To open the project's documentation using cargo,

cargo doc --open

rustc VS cargo

Use cases	rustc	cargo
Project Size	One file	Project with hierarchy of files
Dependencies	Does not support	Can manage using cargo.toml
Documenting	No built-in for that	Can create it using cargo doc

Basic Syntax

print!() and println!()

In Rust, we use print!() and println!() **macros** to show the output on the terminal.

> [!IMPORTANT] Macros are not Functions but we will not dive into it in this 2 days session.

Now, let's write our first program!

```
fn main() {
    println!("Hello, World!");
}
```

To explain what is going on, we need to know that Rust code will start executing only when it has got the main function somewhere in the code.

Unlike C and C++, Rust main function can be anywhere in the code. It can above, under, or in the middle of other functions

> We will talk more about function in the following day session.

Commenting Code

In Rust we use double forward slashes // to comment code.

For example,

```
fn main() {
    println!("Hello, World!");
    // println!("This won't be printed hehe");
}
```

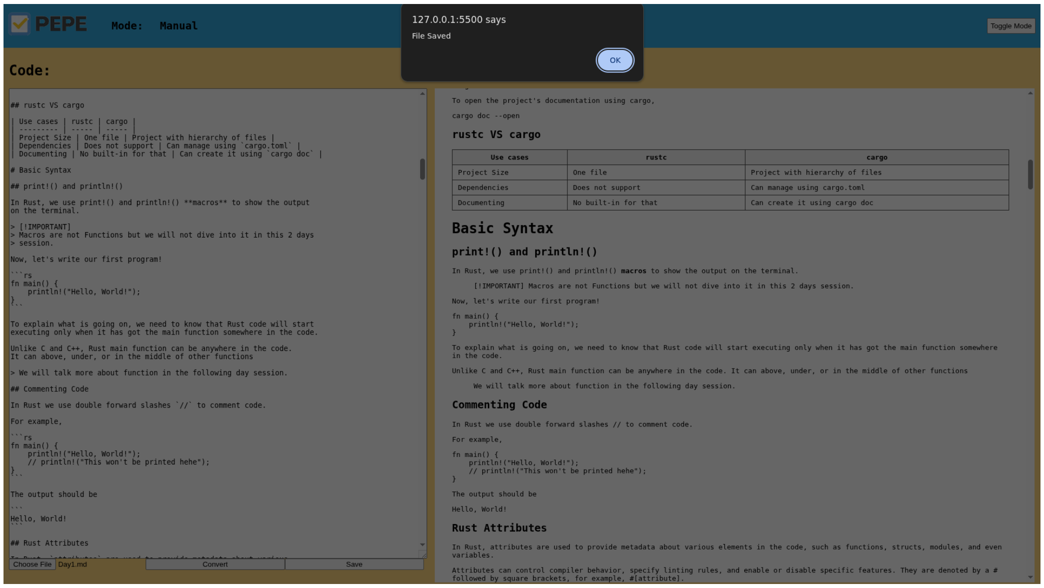
The output should be

```
Hello, World!
```

Rust Attributes

In Rust, attributes are used to provide metadata about various elements in the code, such as functions, structs, modules, and even variables.

Attributes can control compiler behavior, specify linting rules, and enable or disable specific features. They are denoted by a # followed by square brackets, for example, #[attribute].



127.0.0.1:5500 says

File Saved

Mode: Manual

~~~
## rustc VS cargo

| Use cases | rustc | cargo |
| --------- | ----- | ----- |
| Project Size | One file | Project with hierarchy of files |
| Dependencies | Does not support | Can manage using `cargo.toml` |
| Documenting | No built-in for that | Can create it using `cargo doc` |

# Basic Syntax

## print!() and println!()

In Rust, we use print!() and println!() **macros** to show the output
on the terminal.

> [!IMPORTANT]
> Macros are not Functions but we will not dive into it in this 2 days
> session.

Now, let's write our first program!

```rs
fn main() {
    println!("Hello, World!");
}
```

To explain what is going on, we need to know that Rust code will start
executing only when it has got the main function somewhere in the code.

Unlike C and C++, Rust main function can be anywhere in the code.
It can above, under, or in the middle of other functions

> We will talk more about function in the following day session.

## Commenting Code

In Rust we use double forward slashes `//` to comment code.

For example,

```rs
fn main() {
    println!("Hello, World!");
    // println!("This won't be printed hehe");
}
```

The output should be

```
Hello, World!
```

## Rust Attributes
~~~

To open the project's documentation using cargo,

cargo doc --open

rustc VS cargo

Use cases	rustc	cargo
Project Size	One file	Project with hierarchy of files
Dependencies	Does not support	Can manage using cargo.toml
Documenting	No built-in for that	Can create it using cargo doc

Basic Syntax

print!() and println!()

In Rust, we use print!() and println!() **macros** to show the output on the terminal.

[!IMPORTANT] Macros are not Functions but we will not dive into it in this 2 days session.

Now, let's write our first program!

```
fn main() {
    println!("Hello, World!");
}
```

To explain what is going on, we need to know that Rust code will start executing only when it has got the main function somewhere in the code.

Unlike C and C++, Rust main function can be anywhere in the code. It can above, under, or in the middle of other functions

We will talk more about function in the following day session.

Commenting Code

In Rust we use double forward slashes // to comment code.

For example,

```
fn main() {
    println!("Hello, World!");
    // println!("This won't be printed hehe");
}
```

The output should be

```
Hello, World!
```

Rust Attributes

In Rust, attributes are used to provide metadata about various elements in the code, such as functions, structs, modules, and even variables.

Attributes can control compiler behavior, specify linting rules, and enable or disable specific features. They are denoted by a # followed by square brackets, for example, #[attribute].

External Plugin

- `markdown`