

## Group Part

### 1. Code Review

請參閱SE\_HW5\_CodeReview\_1~3。

### 2. 計算

- Bellman-Ford.c

$$\text{Estimated Total Defects} = (16 * 18) / 9 = 32$$

$$\text{Total Defects Found} = 16 + 18 - 9 = 25$$

$$\text{Yield} = 25 / 32 * 100\% = 78.125\%$$

$$\text{Defect Density} = 25 / 134 = 0.186 \text{ defects/page}$$

$$\text{Inspection Rate} = 134 / 3 \text{hrs} = 44.67$$

$$\text{Defect Finding Efficiency} = 25 / 3 \text{hrs} = 8.33$$

- FibonacciHeap.c

$$\text{Estimated Total Defects} = (27 * 29) / 27 = 29$$

$$\text{Total Defects Found} = 27 + 29 - 27 = 29$$

$$\text{Yield} = 29 / 29 * 100\% = 100\%$$

$$\text{Defect Density} = 29 / 392 = 0.074 \text{ defects/page}$$

$$\text{Inspection Rate} = 392 / 5.27 \text{hrs} = 44.67$$

$$\text{Defect Finding Efficiency} = 29 / 5.27 \text{hrs} = 5.50$$

- philutku2.c

$$\text{total defect found} = 19 + 19 - 15 = 23$$

$$\text{estimated total defects} = (19 * 19) / 15 = 24.067$$

$$\text{defect removal efficiency} = \text{total defect found} / \text{estimated total defects} = 95.6\%$$

$$\text{LOC(Lines Of Code)} = 240$$

$$\text{defect density} = \text{total defect found} / \text{LOC} = 23 / 240 = 0.096 \text{ defects / LOC}$$

$$\text{inspection rate} = \text{LOC} / \text{total inspection hours} = 240 / 3.083 = 77.846 \text{ LOC / hour}$$

$$\text{defect finding efficiency} = \text{total defect found} / \text{total inspection hours} = 23 / 3.083 = 7.46 \text{ defects / hour}$$

### 3. Code Review工具回報

- 邱煒甯、顏浩昀

## Code Review-- Gitlab

GitLab offers a built-in code review feature known as "Merge Requests." It allows you to request feedback on your code changes, and team members can provide comments, suggestions, and approve the changes.

Firstly, we can new a merge request and give it a title and describe the changes in this request and also the part the reviewers should be aware of.



**New merge request**

From [fixes](#) into [main](#) [Change branches](#)

**Title (required)**

bug fixed

☐ Mark as draft  
Drafts cannot be merged until resolved manually.

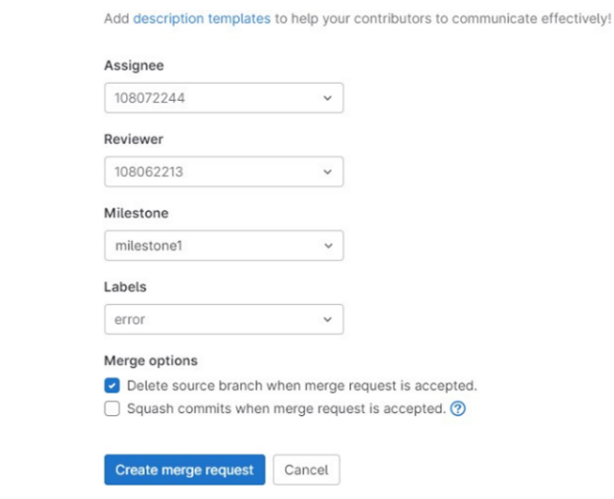
**Description**

Bug fixed, reviewer should focus on the function parameter

Supports Markdown. For quick actions, type [/](#).

Add descriptions liberally to help your contributors to communicate effectively!

Next, we can assign the assignee, reviewers, milestone, and also some labels to this request.



Add [description templates](#) to help your contributors to communicate effectively!

**Assignee**

108072244

**Reviewer**

108062213

**Milestone**

milestone1

**Labels**

error

**Merge options**

☒ Delete source branch when merge request is accepted.

☐ Squash commits when merge request is accepted. [?](#)

[Create merge request](#) [Cancel](#)

If a person is assigned to be a reviewer, then he would have a new task in his to-do list.

## To-Do List

To Do 1 Done 1

Group

Project

bug fixed · 108072244 / SE !4



**108072244** requested a review.

The members of this project can comment on the code to indicate some errors or note some important points.



After the members review all the contents, it can approve the request by pressing the button.

8✓ Approve Approval is optional ?

Merge blocked: merge conflicts must be resolved.

[Resolve locally](#)

There is an activity area which record all the happed events. For example, we can see who has approved the request.

## Activity

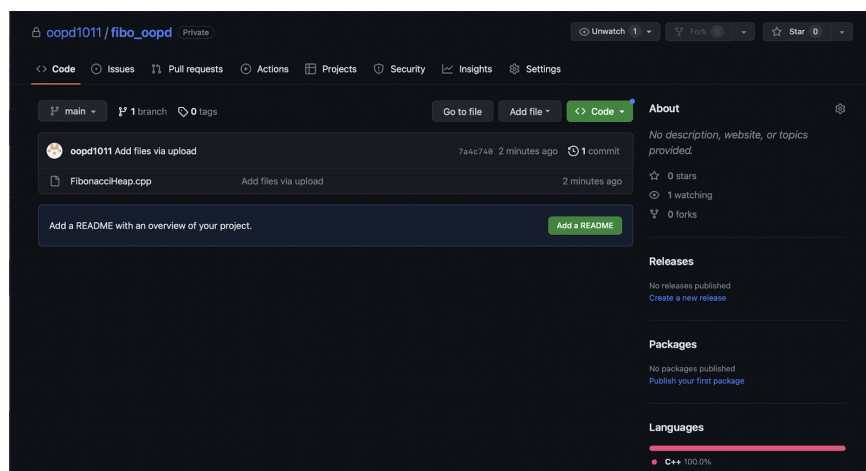
- 108072244 changed milestone to %milestone1 4 minutes ago
- 108072244 added **error** label 4 minutes ago
- 108072244 requested review from @108062213 4 minutes ago
- 108072244 assigned to @william08162010 4 minutes ago
- ✓ 108072244 approved this merge request just now
- ✓ 108062213 approved this merge request just now
- 108062213 unapproved this merge request just now
- ✓ 108062213 approved this merge request just now
- 108072244 unapproved this merge request just now
- 108062213 unapproved this merge request just now
- ✓ 108062213 approved this merge request just now

- 陳璽正、林冠宏

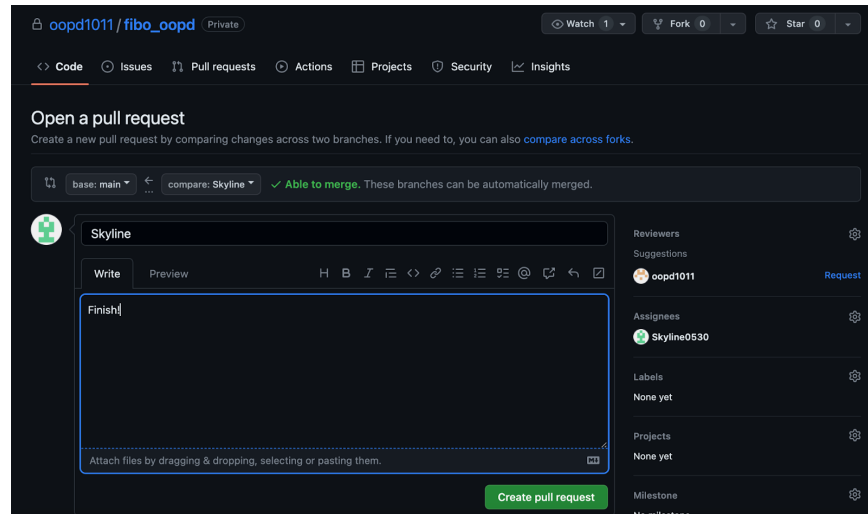
## Code Review- Github

On GitHub, code review tools are built into pull requests. We can request reviews, propose changes, keep track of versions, and protect branches to improve the quality of our code.

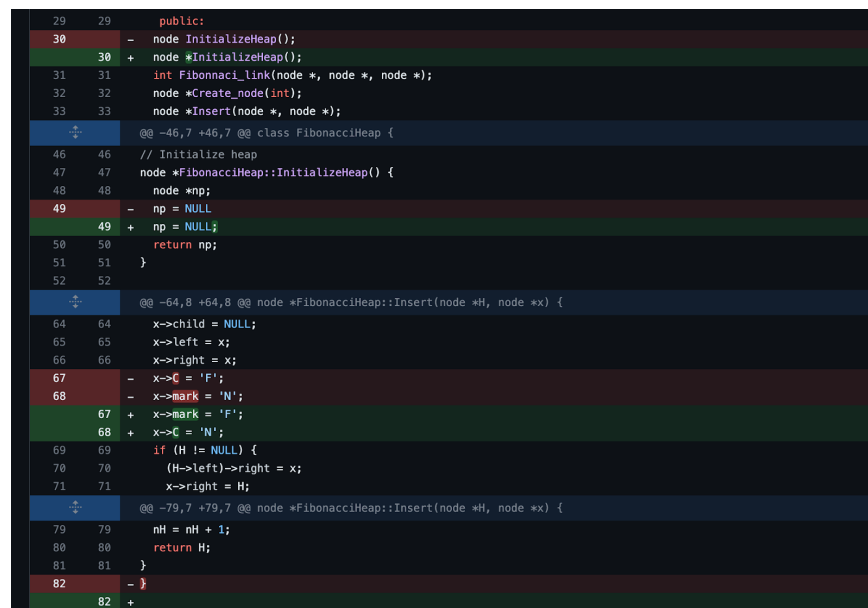
Firstly, upload the original code to the repository, and create a new branch for the reviewed code.



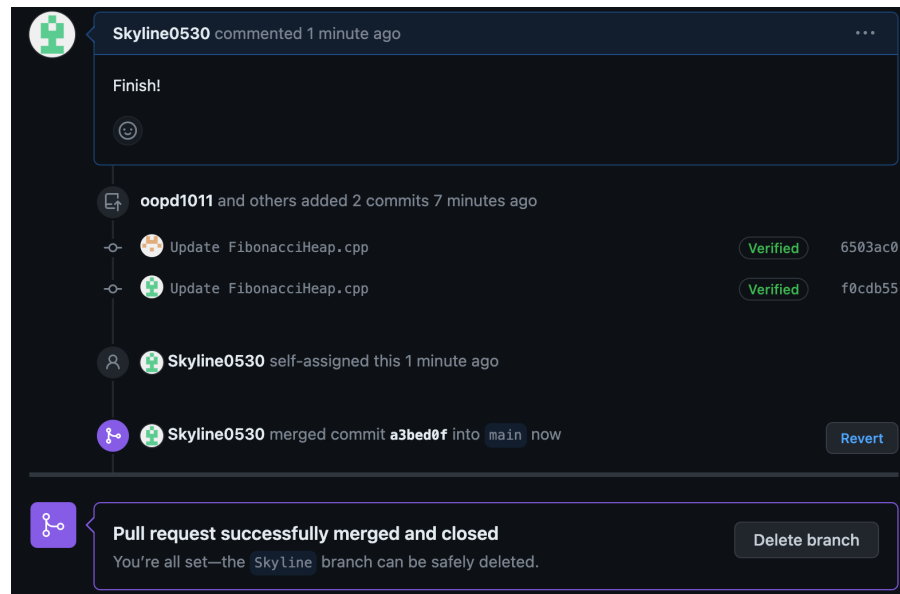
After uploading the reviewed code to the new branch, we can create a pull request. Select reviewers and assignees and leave some messages about the review.



This tool shows the inserted and deleted lines of the code and also the characters that are modified.



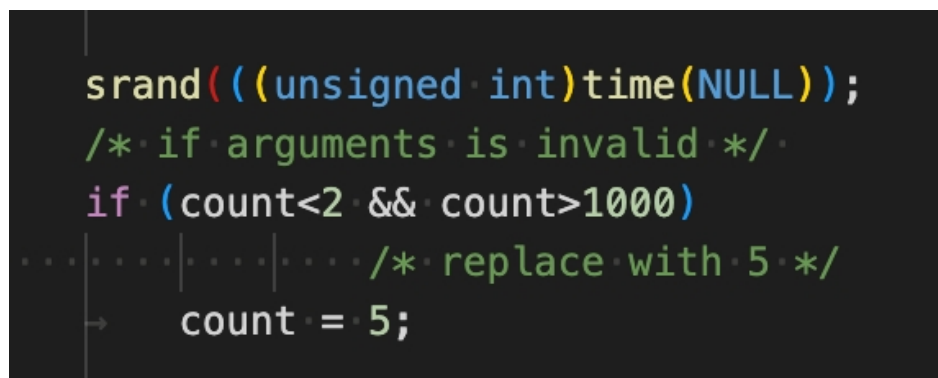
Reviewers merged the pull request with the main branch, the code in main branch is code-reviewed now. We can also see the activities with a timeline in the repository.



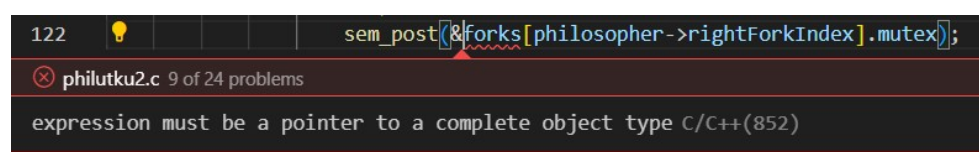
- 劉祥暉、許瀚杰

### Code Review-vscode C/C++ extension

這次我們使用的code review 工具為基本的VS Code配上C/C++ Extention pack，其使用方法相對簡單，只需將安裝並啟用C/C++ Extention pack後就能夠在寫程式頁面看見Extention pack在初步檢查程式碼後發現的錯誤並且將錯誤的部分用紅色標示出來，如下圖



除了直接用顏色來標示之外也會有波浪底線的方式來標註可能的語法或編譯錯誤且能透過點擊標示來獲得更多得錯誤訊息提示，如下圖所示



最後則是能夠透過呼叫console來看整份程式碼的錯誤總攬以及其位置，如下圖

```
C philutku.c 24
❌ too many arguments in function call C/C++(140) [Ln 73, Col 42]
❌ too many arguments in function call C/C++(140) [Ln 79, Col 50]
❌ expression must be a modifiable lvalue C/C++(137) [Ln 85, Col 7]
❌ expression must be a pointer to a complete object type C/C++(852) [Ln 85, Col 20]
❌ expected a ';' C/C++(65) [Ln 89, Col 36]
❌ expression must be a pointer to a complete object type C/C++(852) [Ln 98, Col 22]
❌ too many arguments in function call C/C++(140) [Ln 101, Col 43]
```

## 4. Chatgpt Code Review



cr-gpt [bot]

reviewed 7 minutes ago

[View reviewed changes](#)

```
bellman-ford.c
... @@ -11,7 +11,7 @@ struct Edge
11 11 // This structure is equal to an edge. Edge contains two end points. These edges are directed edges
12 12 //contain source and destination and some weight. These 3 are elements in this structure
13 13 int source, destination, weight;
14 - }
14 + };
15 15
16 16 // a structure to represent a connected, directed and weighted graph
17 17 struct Graph
```

 cr-gpt [bot] 7 minutes ago

Overall, the code patch looks like a simple declaration and definition of structs `Edge` and `Graph`.

However, it's difficult to provide a comprehensive code review without additional context. Here are a few observations:

- The `Edge` struct seems to be missing a semicolon after the closing brace.
- It would be helpful to include comments or documentation explaining the purpose and usage of these structures and how they relate to the larger program or project.
- It's unclear whether there are any functions or methods associated with these structs that manipulate or interact with their data.
- It's generally good practice to include header guards in header files to prevent multiple includes during compilation and potential naming clashes.
- It could also be useful to consider implementing related algorithms or data structures using these structs, especially if this is part of a larger project.

As for improvement suggestions, it depends on the goals and scope of the larger project. If you could provide more information, I could offer more specific ideas for how to improve or expand upon this code.

 cr-gpt[bot] <notifications@github.com> 取消訂閱  
寄給 CHIUWEINING/codereview · 我 · Author

@cr-gpt[bot] commented on this pull request.

In [Bellman-Ford.c](#):

> @@ -55,7 +55,7 @@ void BellmanFord(struct Graph\* graph, int source)

int E = graph->V;

- int StoreDistance[V];

+ int StoreDistance[V];

int i,j;

Based on the code patch:

- The indentation seems to be fixed in the 'FinalSolution' function, which is good.
- In 'BellmanFord', there's a fix where 'StoreDistance' is now correctly initialized as an array of V elements.
- There seem to be no significant issues or bugs. However, without seeing the entire original codebase and its context, it's not possible to identify all potential issues.

As for improvements, some suggestions are:

- It's best practice to name variables in lowercase letters instead of camel case (e.g., 'store\_distance' instead of 'StoreDistance').
- Adding comments to the code can make it more readable and easier to understand the logic and purpose of each function.
- Consider renaming 'FinalSolution' to a more descriptive function name, as it isn't clear from the current name what the function does.
- Add error handling to handle unexpected inputs and corner cases that could cause the program to crash or produce incorrect results.

從上述比較中可以看出，相較於手動的程式碼審查（Manual code review），GPT Bot 可以提供更詳盡的審查內容。首先，GPT Bot 能夠對良好的修改進行稱讚，同時也能從程式碼中指出需要加強的部分。換句話說，它能提供有關程式碼風格的建議，例如在圖片中提到的，建議將變數命名從駝峰式（camel case）改為蛇式命名（snake case）。此外，GPT Bot 也建議我們加入除錯機制，以減少系統崩潰的可能性。

然而，相較於手動的程式碼審查，我們也認為 GPT Bot 提供的審查內容稍嫌冗長，其中包含了許多不重要的資訊。這可能在審查評論數量眾多時讓開發人員難以抓住重點。因此，在這一點上，手動的程式碼審查具有優勢。

總結而言，GPT Bot 的優勢在於提供詳盡的審查內容，並給予讚揚和改進建議，例如程式碼風格和除錯機制。然而，在處理大量審查評論時，手動的程式碼雖說需要花費更多時間及人力資源，但此作法能夠更清晰地將重點傳達給開發人員。最佳的方法可能是結合兩者的優點，以達到更有效的程式碼審查流程。

以下為chatgpt產出針對philutku2.c檔所偵測出的錯誤

語法錯誤：

1. 在 `#include` 的每一行末尾缺少分號。
2. 在 `struct Fork` 的定義中，`fork` 應該是大寫開頭，與結構名稱保持一致。
3. 在 `is_finished` 函數中，`counter = NotEatenCount();` 的 `()` 應該移除，將其更正為 `counter = NotEatenCount;`。
4. 在 `philosopher_thread` 函數中，`int again = 0;` 應該更改為 `int again = 1;`，否則 `while` 循環將永遠不執行。
5. 在 `philosopher_thread` 函數中，`printf("Philosopher %d is Eating\n", philosopher-number)` 的最後缺少分號。



6. 在 `philosopher_thread` 函數中，`if (sem_trywait(&forks[philosopher-leftForkIndex].mutex) = 0)` 的 `=` 應該更正為 `==`，進行相等比較。

邏輯錯誤：

1. 在 `is_finished` 函數中，將 `counter` 與 `0` 進行比較並返回結果是正確的。但是該函數應該返回布爾值而不是整數。你可以將返回語句更正為 `return counter == 0;`。
2. 在 `philosopher_thread` 函數中，`while(again)` 的條件應該更正為 `while (again == 1)`，否則循環永遠不會執行。
3. 在 `philosopher_thread` 函數中，`int waiting_times = 10 + rand() mod 50;` 的 `mod` 應更正為 `%`，以計算取模。
4. 在 `philosopher_thread` 函數中，`waiting_times -= waiting_times;` 應更正為 `waiting_times = -waiting_times;`，以將等待時間變為負數。
5. 在 `philosopher_thread` 函數中，`else if {` 應更正為 `else {`。
6. 在 `philosopher_thread` 函數中，`printf("Philosopher %s cannot eat at this moment...\n", philosopher->number);` 中的 `%s` 應更正為 `%d`，因為 `number` 是整數。
7. 在 `main` 函數中，`srand(((unsigned int)time(NULL)));` 缺少一個右括號 `)`。

比較、討論：

在上述回答中可以發現撇除為被檢查出的內容不計，Chatgpt在分類邏輯錯誤以及語法錯誤的分類就有所不精，譬如說邏輯錯誤中的第7點應該歸類語法錯誤但卻放在邏輯錯誤中，而在語法錯誤中的第4點則應該放在邏輯錯誤中。由上述兩個例子能發現雖然chatgpt有辦法找出錯誤卻無法將其準確歸類。除此之外，在Chatgpt檢查出得錯誤中也能發現可能有錯誤回報的情形發生。如語法錯誤的第1點就是多餘且無需要的修正，再來是邏輯錯誤的第4點，不僅有前面所述的分類錯誤的問題，也有非必要修正且誤判原本程式含意的情況出現。

在討論完chatgpt產出的錯誤報告本身後接下來是與人工檢查的交叉比對結果，由於人工檢查出的項目有許多因此這邊不逐一列出，改由歸納與整理的方式來討論。首先是經由人工修改過的版本才能夠真正的正確運行程式，chatgpt由於是透過程式碼本身來分析與除錯，因此難以向人工審查的方式來用極端test case去檢測程式的正確性，也無法藉由運行程式來判斷是否有超出預期的錯誤。另外就是在語法上的出錯雖然chatgpt也能提供一部分的錯誤回報並提出解決方案，但其能檢查出的錯誤仍然並沒有達到百分之百，且除了正確回報外有會參雜了不正確或是多餘的錯誤回報。

綜合以上，我們認為chatgpt在語法除錯上是有一定的處理能力的，但在邏輯除錯上則是幾乎沒有辦法提供使用者難以快速發現的錯誤，依然需要以人工審查的方式下去進行code review才能避免絕大多數的錯誤。因此我們覺得chatgpt的幫助可以在code review的初期快速找出可能的問題供工程師參考，最後再由人工的方式才能完成完整的code review。