開發環境

本程式在 Windows 64-bit 環境下開發，使用 C++ 語言編寫。

實作方法和流程

A. 印出提示，請使用者輸入檔案名稱。

B. 嘗試以檔案名稱開啟檔案。
   a. 若開檔失敗，則回到 1.。
   b. 若開檔成功，則呼叫 readFile() 將檔案資料讀入。

C. 根據檔案中選擇的排序方式來執行 FIFO()、LRU()、LFU_FIFO()、MFU_FIFO()、LFU_LRU() 中的其中一個或全部一起執行，在執行過程中即逐步輸出結果至檔案中。

D. 回到步驟 A，等待下一次 User 輸入。

- **FIFO()：**

用 for 迴圈模擬 reference string，處理當下的 page。在每個時間點上：

在 pageTable 中尋找該 page 的資訊：

若有找到，將其 counter++。

若未找到，則在 pageTable 中創立此 page 的資料，並將 counter 設為 1。

若 frame 中目前未含有此 page。

若 frame 未滿，將此 page 加入 frame 中。

若 frame 已滿，移除其中的第一個 page，再將此 page 加入其中。

- **LRU()：**

用 for 迴圈模擬 reference string，處理當下的 page。在每個時間點上：

在 pageTable 中尋找該 page 的資訊：

若有找到，將其 counter++。

若未找到，則在 pageTable 中創立此 page 的資料，並將 counter 設為 1。

若 frame 中目前未含有此 page。

若 frame 未滿，將此 page 加入 frame 中。

若 frame 已滿，移除其中的第一個 page，再將此 page 加入其中。

若 frame 中目前已含有此 page。

將其移至 frame 的最尾端。

- LFU_FIFO() :

用 for 迴圈模擬 reference string，處理當下的 page。在每個時間點上：

在 pageTable 中尋找該 page 的資訊：

若有找到，將其 counter++。

若未找到，則在 pageTable 中創立此 page 的資料，並將 counter 設為 1。

若 frame 中目前未含有此 page。

若 frame 未滿，將此 page 加入 frame 中。

若 frame 已滿，移除其中 counter 最少的 page，再將此 page 加入 frame 的最尾端。

- MFU_FIFO() :

用 for 迴圈模擬 reference string，處理當下的 page。在每個時間點上：

在 pageTable 中尋找該 page 的資訊：

若有找到，將其 counter++。

若未找到，則在 pageTable 中創立此 page 的資料，並將 counter 設為 1。

若 frame 中目前未含有此 page。

若 frame 未滿，將此 page 加入 frame 中。

若 frame 已滿，移除其中 counter 最多的 page，再將此 page 加入 frame 的最尾端。

- LFU_LRU() :

用 for 迴圈模擬 reference string，處理當下的 page。在每個時間點上：

在 pageTable 中尋找該 page 的資訊：

若有找到，將其 counter++。

若未找到，則在 pageTable 中創立此 page 的資料，並將 counter 設為 1。

若 frame 中目前未含有此 page。

若 frame 未滿，將此 page 加入 frame 中。

若 frame 已滿，移除其中 counter 最少的 page，再將此 page 加入 frame 的最尾端。

若 frame 中目前已含有此 page。

將其移至 frame 的最尾端。

分析不同方法之間的比較

FIFO（先進先出）：

Page Fault：FIFO 方法在每次 page fault 時都會將最早載入的 page 置換出去，不考慮該 page 是否是最常使用的。因此，如果最近使用的 page 很重要，FIFO 的表現可能較差。

Page Replace：FIFO 方法會根據時間標記來置換最早載入的 page，因此會將最久未使用的 page 替換掉。

LRU（最近最少使用）：

Page Fault：LRU 方法考慮到了最近使用的 page，如果一個 page 最近被使用過，那麼它被保留在 memory 中的機會較高。因此，LRU 可以減少 page fault 的數量。

Page Replace：當 frame 滿了時，LRU 方法將最久未使用的 page 置換出去，以便為新的 page 留出空間。

LFU_FIFO（最不常使用先進先出）：

Page Fault：LFU_FIFO 方法根據 counter 和時間標記來決定 page 的重要性，並且置換 counter 最小的 page。這使得較不常使用的 page 在一段時間後被置換的可能性較高。

Page Replace：LFU_FIFO 方法會根據 counter 和時間標記來置換 counter 最小的 page，若有多個 page 的 counter 值一樣，則置換其中時間標記最小的 page。

MFU_FIFO（最常使用先進先出）：

Page Fault：MFU_FIFO 方法根據 counter 和時間標記來決定 page 的重要性，並且置換 counter 最大的 page。這意味著最常使用的 page 可能會從 memory 中被刪除。

Page Replace：MFU_FIFO 方法會根據 counter 和時間標記來置換 counter 最大的 page，若有多個 page 的 counter 值一樣，則置換其中時間標記最小的 page。

LFU_LRU（最不常使用最近最少使用）：

Page Fault：LFU_LRU 方法同時考慮了 counter 和時間標記，以確定 page 的重要性。它將 counter 最小的 page 置換出去，如果有多個 page 的 counter 值一樣，則置換其中時間標記最小的 page。這使得不常使用且最近未使用的 page 在一段時間後被置換的可能性較高。

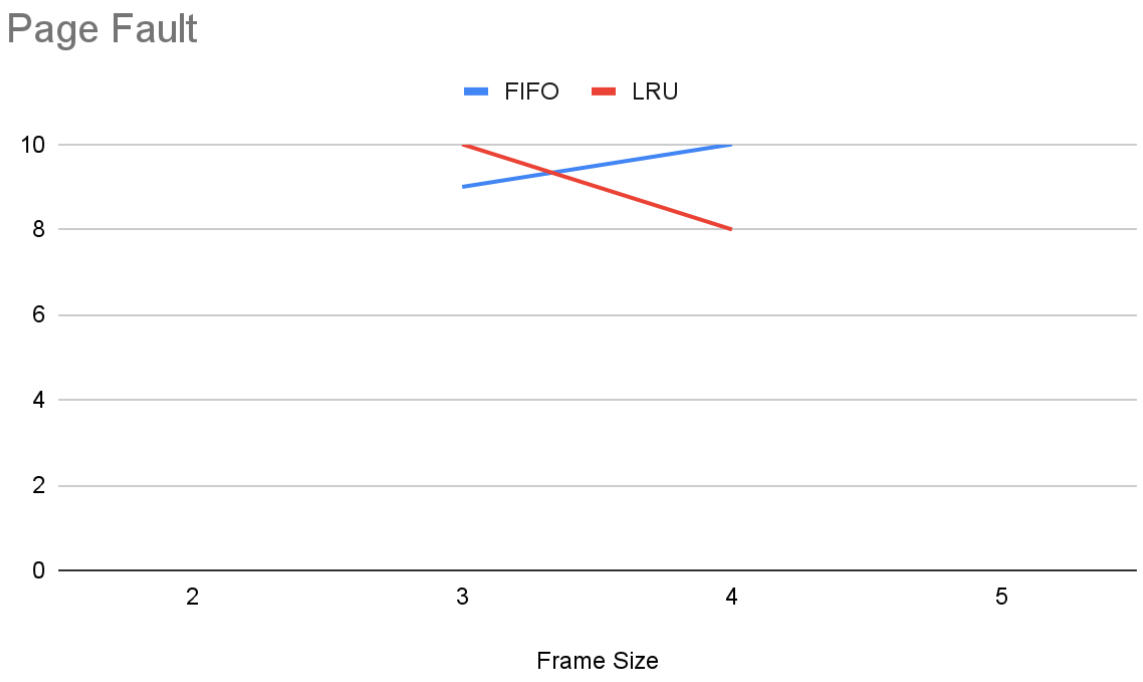
Page Replace：LFU_LRU 方法同時考慮了 counter 和時間標記，以確定要置換的 page。它將 counter 最小的 page 置換出去，如果有多個 page 的 counter 值一樣，則置換其中時間標記最小的 page。

總結而言，每種方法的表現都有所不同，取決於應用的特定情況和需求。FIFO 是最簡單的方法，但可能無法有效利用記憶體。LRU 考慮到最近使用的 page，可能能夠減少 page fault 的數量。LFU_FIFO 和 MFU_FIFO 基於 counter 和時間標記來決定 page 的重要性，適用於根據使用頻率進行調節。LFU_LRU 同時考慮了 counter 和時間標記，可以兼顧最不常使用和最近未使用的 page。選擇適合的方法需要考慮到特定應用的需求和資源限制。

結果與討論

當 frame 的大小增加時，一般來說，我們預期 page fault 的次數應該會下降。這是因為較大的 frame 可以容納更多的 page，提供更多的機會讓被引用的 page 保持在記憶體中，減少 page fault 的發生。

但當 reference string 為 "123412512345" 的情況下，我們將 frame size 從 3 增加到 4 時，FIFO 和 LFU_FIFO 的 page fault 次數上升，這正是 Belady's Anomaly 的一個例子。這意味著增加 frame size 反而導致了更多的 page fault。



Belady's Anomaly 是在一些特定的 reference string 情況下觀察到的，這些 reference string 是對某個演算法（如 FIFO 與 LFU_FIFO）不利的情況。當 frame size 增加時，這些演算法的 page fault 次數反而會增加，這是相當罕見的現象。

Belady's Anomaly 的存在表明，並不是所有的演算法都遵循 "增加 frame size 可以減少 page fault" 的規則。這也強調了在選擇和設計頁置換演算法時，需要考慮各種因素，包括參考串的特性以及演算法本身的特點。