一、 開發環境 ：

1. Windows11
2. IDE : dev c++

二、 實作方法與流程

1. 資料結構:

```
struct cFrame {
  char ch ; // page reference string
  int count ; // counter
};

struct frame{
    char pageNum ; // page reference string
    bool pageFault ;
    vector<cFrame> frameContent ; // page frame
};

class pageReplacement{
    private:
        int gframeNum = 0 ; // frame 數量
        int PageReplaces = 0 ; // Page Replaces次數
        int PageFault = 0 ; //  Page Fault 次數
        vector<cFrame> currentFrame ; // 目前frame的內容
        vector<frame> vec ; // 所有data
```

2. Page Replacement 方法實作說明 ：

   I. **First In First Out (FIFO) :**

   資料讀入 vector 後，將讀入的 char(參考頁面)依序放入 currentFrame 中

   - 若 currentFrame 的 size 小於 frame size 則直接放入
   - 若資料已存在 currentFrame 中，則不改變 currentFrame 內容
   - 若 Page Frame 被放滿無法再放入其他 Page 時，則執行 Page Replacement，選擇 currentFrame 中的第一筆資料，即 loading time 最小的 page 作為 victim page，將它替換出去並載入新的 page。

   II. **Least Recently Used (LRU) :**

   資料讀入 vector 後，將讀入的 char(參考頁面)依序放入 currentFrame 中

   - 若 currentFrame 的 size 小於 frame size 則直接放入
   - 若資料已存在 currentFrame 中，則將資料取出，再重新放入 currentFrame
   - 若 Page Frame 被放滿無法再放入其他 Page 時，則執行 Page Replacement，選擇 currentFrame 的第一筆資料，即最近不常使用的 page 作為 victim page，將它替換出去並載入新的 page。

III. **Least Frequently Used (LFU) + LRU :**

資料讀入 vector 後，將讀入的 char(參考頁面)依序放入 currentFrame 中，且在每個 frame 以 counter 紀錄 Page Frame 被參考或修改的次數，初值設為 1，當 Page Frame 被參考或修改時，counter 值會加 1。

- 若 currentFrame 的 size 小於 frame size 則將參考頁面直接放入 currentFrame
- 若資料已存在 currentFrame 中，則將資料取出，再重新放入 currentFrame
- 若 Page Frame 被放滿無法再放入其他 Page 時，則執行 Page Replacement，選擇 counter 值最小的 Page Frame 作為 victim page，將它替換出去並載入新的 page。若有 frame 的 counter 值相同時，則選擇過去不常使用的 page 也就是 vector index 較小的資料作為 victim page。

IV. **Most Frequently Used (MFU) + FIFO :**

資料讀入 vector 後，將讀入的 char(參考頁面)依序放入 currentFrame 中，且在每個 frame 中以 counter 紀錄 Page Frame 被參考或修改的次數，初值設為 1，當 Page Frame 被參考或修改時，counter 值會加 1。

- 若 currentFrame 的 size 小於 frame size 則將參考頁面直接放入 currentFrame
- 若資料已存在 currentFrame 中，則將資料的 counter 值加 1
- 若 Page Frame 被放滿無法再放入其他 Page 時，則執行 Page Replacement，選擇 counter 值最大的 Page Frame 作為 victim page，將它替換出去並載入新的 page。若有 frame 的 counter 值相同時，則選擇 loading time 最小的 page 也就是 vector index 較小的資料作為 victim page。

V. **MFU+LRU :**

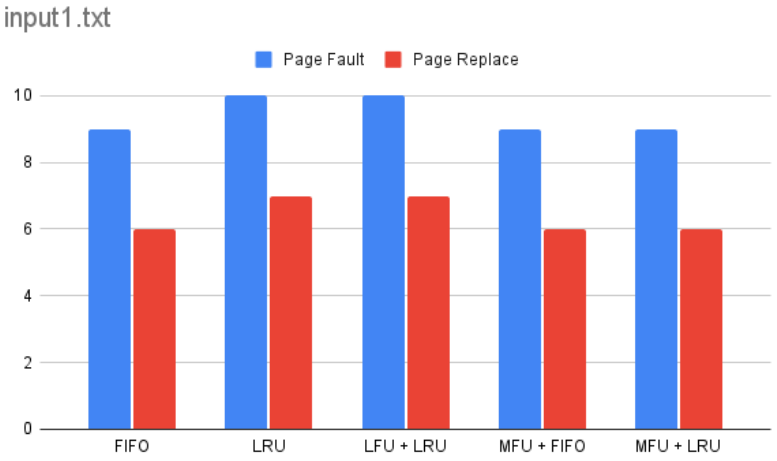
資料讀入 vector 後，將讀入的 char(參考頁面)依序放入 currentFrame 中，且在每個 frame 中以 counter 紀錄 Page Frame 被參考或修改的次數，初值設為 1，當 Page Frame 被參考或修改時，counter 值會加 1。

- 若 currentFrame 的 size 小於 frame size 則將參考頁面直接放入 currentFrame
- 若資料已存在 currentFrame 中，則將資料取出，再重新放入 currentFrame
- 若 Page Frame 被放滿無法再放入其他 Page 時，則執行 Page Replacement，選擇 counter 值最大的 Page Frame 作為 victim page，將它替換出去並載入新的 page。若有 frame 的 counter 值相同時，則選擇過去不常使用的 page 也就是 vector index 較小的資料作為 victim page。

三、 不同方法之間的比較

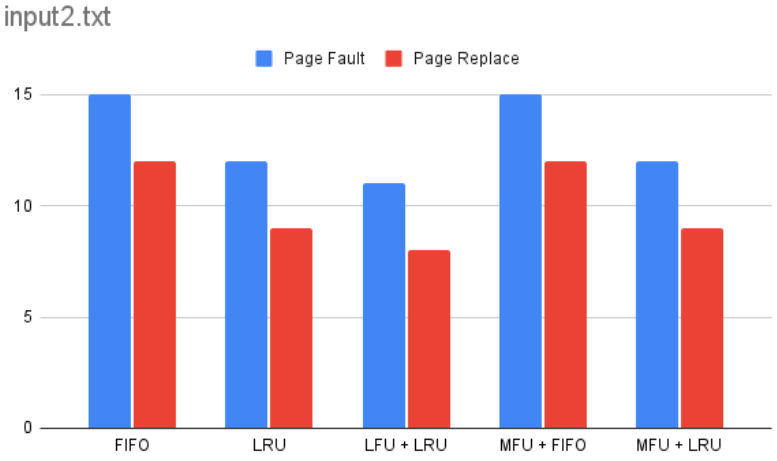
1. Input1.txt : Page Fault 與 Page Replace 次數

	Page Fault	Page Replace
FIFO	9	6
LRU	10	7
LFU + LRU	10	7
MFU + FIFO	9	6
MFU + LRU	9	6



2. Input2.txt : Page Fault 與 Page Replace 次數

	Page Fault	Page Replace
FIFO	15	12
LRU	12	9
LFU + LRU	11	8
MFU + FIFO	15	12
MFU + LRU	12	9



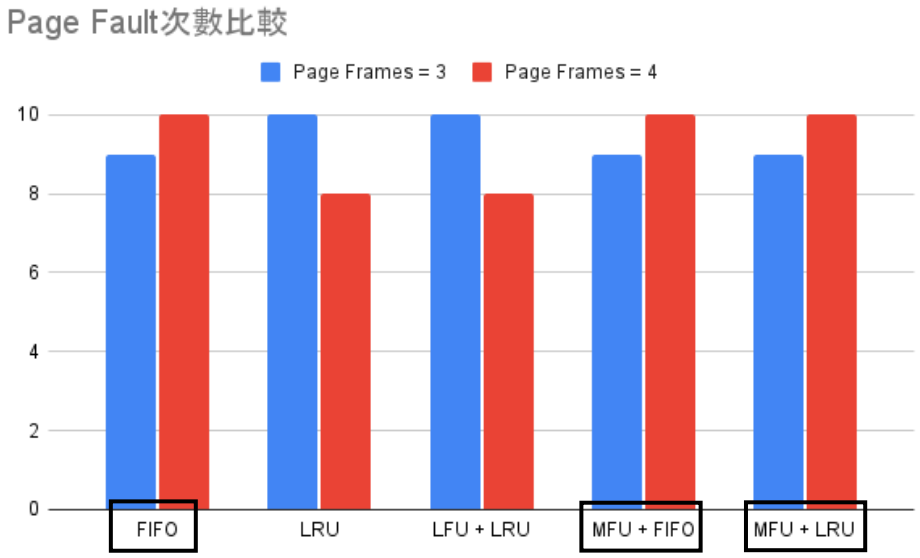
四、結果與討論:

1. 畢雷笛反例(Belady Anomaly)

(1) Belady Anomaly 是指分配給 process 的頁框數增加，照理來說 page fault ratio 應該下降，但是其 page fault ratio 不降反升的異常現象。

(2) 以 input1.txt 為例，當 Page Frames = 3、4 的 Page Fault 次數比較:

	Page Frames = 3	Page Frames = 4
FIFO	9	10
LRU	10	8
LFU + LRU	10	8
MFU + FIFO	9	10
MFU + LRU	9	10



(3) 由圖表可發現當 page frames 從 3 增加到 4 時，FIFO 法則、MFU + FIFO 法則、MFU + LRU 法則的 Page Fault 次數皆從 9 上升到 10，即發生了 Belady Anomaly 的異常現象。

(4) 沒有發生 Belady Anomaly 的法則之共同特徵:

- Stack property : 3 個 page frames 所包含的 pages set 是 4 個 page frames 所包含的 pages set 之子集合
- 以 LRU 為例:

-------- 3 個 page frames ---------	
1	1
2	21
3	321
4	432
1	143
2	214
5	521
1	152
2	215
3	321
4	432
5	543

-------- 4 個 page frames ---------	
1	1
2	21
3	321
4	4321
1	1432
2	2143
5	5214
1	1524
2	2154
3	3215
4	4321
5	5432

2. 其他討論:

由 input1、input2 的數據可以看出在所有 replacement 法則中，沒有法則是一定最差的，結果大部分取決於輸入的資料。