RM、EDF 與、strict LST 排程器

說明：

撰寫 RM、EDF、與、strict LST 排程機制，從 task.txt 檔案中讀取週期性任務的資料，之後模擬時間的前進，在當有新工作抵達放入 linked list 的 ready queue。當新工作抵達或是工作結束時，從 ready queue 中挑選優先權最高的工作執行。

輸入檔格式(文字格式)task.txt

每一行即為一個週期性的 task，第一個數字為週期，第二個為執行時間，且都為正整數，且以逗號隔開，週期與相對截限時間不一定相等。任務個數不固定。

Ex:

0, 5, 2, 1　(phase time, period, relative deadline, execution time)

2, 3, 2, 1

輸出檔案格式

(1)如果是使用 EDF 排程方法，請先以 Schedulability test 判斷是否可排。 $\sum_{i=1}^{n} \frac{e_i}{\min(p_i, D_i)} \leq 1$ 。如果是 RM，請以 $\sum_{i=1}^{n} \frac{e_i}{\min(p_i, D_i)} \leq n(2^{\frac{1}{n}} - 1)$

(2)請依據排程演算法(EDF 或是 RM)計算出每個單位時間由哪一個 task 在使用，或是畫出甘特圖。任務編號請從 1 開始。

Ex:

以 RM 為例，

0 T1

1 T2

2 T1

3 T2

4 T1

5

以時間 3 為例，代表時間間隔 3-4 為任務 T2 的工作所使用

繳交時間：

11/23

繳交內容：

原始程式、2-3 頁報告

PS: 請以 **linked list** 實作存放系統中等待被執行工作的 ready queue。需要改變目前優先權最高工作之執行，只有在新工作抵達或是工作執行結束。

```
struct Task{
	int TID;
	int Phase;
	int Period;
	int WCET; //worst-case execution time
	int RDeadline; //relative deadline
	float Utilization;
};

struct Job{
	int release_time;
	int remain_execution_time;
	int absolute_deadline;
	int TID; //屬於哪個 Task 的工作
};
int Total_Job_Number=0;
int Miss_Deadline_Job_Number=0;
```

Pseudocode

1: 先將 task.txt 中的每個任務資料讀入，並儲存在適當資料結構中。N 為任務總個數

2: 求得所有任務之週期的最小公倍數 LCM，與所有任務之最大 Phase Time MaxPH

3: 將系統的就緒佇列 Q 初始化成空佇列

4: 初始化時間 Clock 為 0;

5: While(Clock<(LCM+MaxPH))

{

- 6: 判斷 Q 中的每一個工作是否能在它的絕對截限時間之前完成，((d-Clock-rem_exe)>=0)，若不能，則輸出該工作訊息，並紀錄 miss 工作加 1 和刪除
- 7: 針對每一個任務判斷目前時間 Clock 是否為它的工作之抵達時間((Clock-Phase)%pi)==0。若是，則在 Q 中加入工作，並紀錄進入系統的工作個數加 1
- 8: 當有新工作抵達或是結束時，從 Q 中找到優先權最高的工作，並將其執行時間減 1。如果執行時間已為 0，則刪除該工作。RM、EDF、strict LST 在尋找優先權最高的工作時，所參考的參數不同，RM 檢視週期；EDF 檢視絕對截限時間；strict LST 則檢視 slack。否則，將正在執行中的

工作執行時間減一。

```
    9: Clock++;
}
```