一、開發環境 ：

Window10，DevC++，語言使用 C++

二、程式設計 ：

功能 ： 讀入 input，並依照檔案內所選擇的 method 來進行排程。

Method1： 先到先服務排程法，依照到達的時間先後排序，若到達時間相同則依照 process ID 小至大來排序，依序佔有 CPU 執行，直到執行完後才會換下一個 process 執行。

Method2 : Round Robin 排程法，依到達的順序執行，每次執行一個 time slice，執行完後就到 ready queue 內取下一個 process 執行，並排到 ready queue 的最後面，等待執行。

Method3： 最短剩餘時間優先排程法，每當有 process 到達時，計算每個 process 的剩餘 CPU Burst Time 為多少，由最少的先佔有 CPU 執行。

Method4： 優先等級排程法搭配 Round Robin 排程法，依照 process priority 來排序，priority 越小就越先執行，若遇到 priority 相同的情況，則使用 Round Robin 排程法來排程，每當有 process 到達時，都要檢查是否有 priority 較高的 process 到達，若有較高優先等級的 process 到達，則要先換較高優先等級的 process 執行。

Method5： 最高反應時間比率優先排程法，計算Response Ratio，Response Ratio較高的先做，每次做完後都會重新計算Response Ratio，若Response Ratio相同則依照到達時間排序，若連到達時間都相同則依照process ID來排序，越小越先做。

※ Response Ratio = (Waiting Time + CPU Burst Time) / CPU Burst Time

Method6： 執行Method1~5的所有排程法。

最後輸出甘特圖、Waiting Time以及Turnaround Time。

使用的資料結構 ：

使用 struct Data，裡面包含兩種 type 的 process ID(integer 跟 string，比較大小時使用 integer→pid，輸出時使用 string→id)、Arrival Time(arr_time)、CPU Burst Time(cputime)、process priority(priority)、waiting time(wait_time)以及 turnaround time(done_time)，皆為 integer。

用 class Scheduling 來做排程的工作，有 6 個 type 為 Data 的一維 vector，一個是用來讀入 input 檔案的資料(data)，其他分別為儲存 5 種排程方法所產生的結果(fcfs, rr, srtf, pprr, hrrn)，有儲存 Method 方法的 integer(method)，儲存 timeSlice 的 integer(time_slice)，以及儲存甘特圖結果的 string(print)。另外還有一個 struct Priority_Queue，是在 Method4(PPRR)用來讓 process 排隊用的資料結構，裡面有儲存 priority 的 integer，以及儲存排隊的 process 的 index 用的 type 為 integer 的一維 queue。

流程 ：

先宣告一個 Scheduling s，請使用者輸入 input 檔名，若檔名錯誤就會告訴使用者找不到檔案並問使用者是否要繼續執行，若要繼續執行就輸入 0 以外的數字，若輸入 0 則結束程式。若檔名正確，則會讀入 input 檔，並且將內容存到 data、method 跟 time_slice。讀完檔案後，就判斷要執行哪種 Method。同時我會傳一個 Boolean 參數(doAll)，告訴 function 我是否要直接做完就輸出 output 檔，若傳入參數為 false，則代表做完這個 function 即可輸出檔案，若為 true 則要等 5 種排程法都做好才輸出檔案。以下我會詳細說明這 5 個 Method 的執行流程。

5 種排程法的相同流程 ：

一個 type 為 Data 的 vector(schedule)來複製一份 data，讓原始讀入資料可以留著，主要會以 schedule 來做排程，宣告 1 個 integer timer 初始化為 0 來計算時間，以及宣告 1 個 string 初始化為 NULL 來儲存甘特圖結果(out)。程式會將 schedule 依照到達時間排序(BubbleSort)，若到達時間相同則依照 process ID 大小排序，較小的排在前面。然後會從 schedule 的第一筆資料開始判斷直到所有 process 皆完成，所有 process 執行完畢後，將 schedule 的結果複製到對應的 vector，並將 vector 依 process ID 做排序，判斷是否要輸出檔案，若要就輸出檔案。

FCFS(doAll) :

1. 先檢查 timer 是否小於 process 的到達時間，代表在 process 到達之前，上一個 process 就已經做好了，故 CPU 在 timer~process 到達的這段時間內都在閒置，所以我會用迴圈看 CPU 閒置多久，將'-'加到 string 中，並更新 timer，計算 process 做完需要多久，並將其 process ID 加到 string 中，且計算 Turnaround time 以及等待時間，由於在 process 到達之前 CPU 就閒置了，故等待時間為 0，Turnaround time 為其 CPU Burst Time。
2. 若 timer 大於或等於 process 到達的時間，則代表 CPU 並沒有閒置，先計算 process 等待了多久，再計算 Turnaround time，Turnaround time 為等待時間+CPU Burst Time(因為在這裡沒有做 I/O，故不需考慮做 I/O 的時間)，計算 process 做完需要多久，並將其 process ID 加到 string 中，更新 timer。
3. 最後將 process 存回 schedule 內，換下一個 process。

RR(doAll) :

宣告 1 個 type 為 integer 的 queue(q)，儲存排隊的 process index。

1. 先判斷是否 queue 為空，若為空則代表沒有 process 在排隊，就看是否所有 process 都處理完了，若所有 process 都處理完了，則代表全部做完了。若還有 process 尚未被處理到，則到下面的(3.)繼續判斷。

2. 若 queue 不為空，則取排隊隊伍的第一個 process 做事，並將 queue 的第 1 個 process pop 掉。
   (1) 判斷正在執行的 process 是否可在此次的 time_slice 中完成工作，如果能完成，計算做完需要多久，並將其 process ID 加到 string 中，更新 timer，計算 Turnaround time 為完成的時間減掉到達的時間，再計算等待了多久→Turnaround Time – process 工作時長。
   (2) 若無法完成，則將正在執行的 process 的 CPU Burst Time 減掉 time_slice，並用 d.wait_time 來暫存到目前為止，d 佔有 CPU 多久時間，以便之後計算等待時間(等待時間為 Turnaround time – CPU Burst Time)，更新 timer，計算做完需要多久，並將其 process ID 加到 string 中。
   (3) 最後將結果存回 schedule，並繼續執行下面的第 3 點。
3. 判斷是否還有下一個 process，若有就 timer 是否大於等於下一個 process 到達時間，若 timer 較大或等於到達時間，則代表在 process 執行期間下一個 process 已到達或 process 做完時剛好有新 process 到達，下一個 process 會先排進 queue 中，並看後面是否有相同到達時間的 process，都先加到 queue 中。若 timer 較小，則是剛才執行的 process 先排到 queue 中。若無下一個 process，則重複執行第 1 點，直到所有 process 皆執行完畢。

SRTF(doAll) :

宣告 1 個 type 為 Data 的 vector(q)，讓 process 依照剩餘時間大小排隊(小至大)，若大小相同則依到達時間先後排序，若到達時間相同則依 process ID 排序(小至大)。

1. 若有下一個 process，判斷 timer 是否小於下一個 process 的到達時間，若 timer 較小，則看隊伍裡是否有 process 在排隊，如果沒有代表 CPU 有閒置，就輸出'-'到 string 中並將下一個 process 加到 q 中，更新 timer。
2. 若 timer 大於等於下一個 process 的到達時間，則直接將下一個 process 加到 q 中，並判斷後面是否有 process 的到達時間小於 timer 的，都加到 q 中。
3. 判斷是否有 process 在排隊，將隊伍依照 CPU 剩餘時間排序(小至大)，取出隊伍第一個 process，讓它做到下一個 process 到達為止，計算做完需要多久，並將其 process ID 加到 string 中，若在此期間 process 完成工作就計算其等待時間以及 Turnaround time，且更新 timer 並將此 process 從 q 中移除。若無下一個 process，則直接讓 process 做完，並計算其等待時間以及 Turnaround time，且更新 timer 並將此 process 從 q 中移除。

PPRR(doAll) :

宣告 1 個 type 為 Priority_Queue 的 vector(q)，讓 process 依 priority 排隊。

1. 若有下一個 process，判斷 timer 是否小於下一個 process 的到達時間，若 timer 較小，則看隊伍裡是否有 process 在排隊，如果沒有代表 CPU 有閒置，就輸出'-'到 string 中並將下一個 process 加到 q 中，更新 timer。
2. 若 timer 等於下一個 process 的到達時間，則直接將下一個 process 加到 q 中，並判斷後面是否有 process 的到達時間小於 timer 的，都加到 q 中。
3. 判斷是否有 process 在排隊，將隊伍依照 priority 排序(小至大)，取出 q 的第一個 queue，取出 queue 的第一個 process，若 queue 的長度為 1，就讓它做到下一個 process 到達為止，計算做完需要多久，並將其 process ID 加到 string 中，若在此期間 process 完成工作就計算其等待時間以及 Turnaround time，且更新 timer 並將此 process 從 q 中移除。若無下一個 process，則直接讓 process 做完，並計算其等待時間以及 Turnaround time，且更新 timer 並將此 process 從 q 中移除。
4. 若 queue 長度大於 1，代表要做 RR，看 process 在此 timeSlice 能否完成工作，若能就計算等待時間與 Turnaround time 更新 timer，並將結果存回 schedule，且將 process 從 queue 中移除。若 timeSlice 用完就從 queue 中取下一個 process 執行，直到下一個 process 到達為止。
5. 若所有 process 都進入 queue 中，就看是否要做 RR，不用則直接讓 process 做完，若需要，則讓相同 priority 的 process 做 RR，直到相同 priority 的 process 都完成了，在換下一個 priority 的 process，並將結果存回 schedule。

HRRN(doAll) :

宣告 1 個 type 為 Data 的 vector(schedule_ratio)，讓 process 依 Response Ratio 大小排隊(大至小)，若大小相同則依到達時間先後排序，若到達時間相同則依 process ID 排序(小至大)。

1. 若有下一個 process，判斷 timer 是否小於下一個 process 的到達時間，若 timer 較小，則看隊伍裡是否有 process 在排隊，如果沒有代表 CPU 有閒置，就輸出'-'到 string 中並將下一個 process 加到 schedule_ratio 中，更新 timer。
2. 判斷是否有 process 在排隊，若有就計算 schedule_ratio 內的所有 process 的 Response Ratio，並將其排序(大至小)，取出 schedule_ratio 的第一個 process 並將其移除，直接讓它做完，計算等待時間與 Turnaround time 並更新 timer，將結果存回 schedule。

Method 6 : 執行上述 5 種排程法後再輸出檔案。

三、未完成的功能 :

無。

四、不同排程法的比較(等待時間) :

Input1： 平均等待時間:SRTF < HRRN < FCFS < PPRR < RR

ID	FCFS	RR	SRTF	PPRR	HRRN
0	19	18	0	0	19
1	13	8	0	0	5
2	22	19	2	14	16
3	18	25	6	0	14
4	13	19	0	11	13
5	20	27	19	21	23
6	0	15	6	11	0
7	15	2	0	55	3
8	21	14	0	9	11
9	5	13	1	0	6
10	8	37	49	45	18
13	18	3	0	0	4
20	13	17	0	40	13
27	16	28	19	10	9
29	14	31	19	4	20
平均等待時間	14.33333	18.4	8.066667	14.66667	11.6

Input2： 平均等待時間:SRTF < RR <HRRN < FCFS < PPRR

ID	FCFS	RR	SRTF	PPRR	HRRN
1	0	13	13	0	0
2	10	2	0	21	10
3	10	2	0	8	12
4	11	6	1	9	8
5	11	9	1	9	11
平均等待時間	8.4	6.4	3	9.4	8.2

Input3： 平均等待時間:SRTF = FCFS = HRRN < RR < PPRR

ID	FCFS	RR	SRTF	PPRR	HRRN
1	0	0	0	0	0
2	0	20	0	30	0
3	20	30	20	35	20
4	15	15	15	0	15
5	0	0	0	10	0
6	5	5	5	0	5
平均等待時間	6.666667	11.66667	6.666667	12.5	6.666667

由上面表格可看出，平均等待時間最短的是 SRTF。

FCFS： 先到的 process 會先做，但是有可能先到的 process CPUtime 較長，導致後面的 process 到了卻要等很久，所以平均等待時間時長時短，由於是依照到達時間執行，故不會有餓死的問題。

RR： 由於使用分時排程需要輪流執行，故完成的時間較長，所以等待時間也會變長，但是 RR 能夠讓每個 process 都能執行一些，故不會有餓死的情況發生。

SRTF： 會先執行剩餘時間較短的 process，因為每個 process 只需等待前面執行時間比自己短的 process 即可，故平均等待時間最短，但是可能一直有執行時間較短的 process 進入，有可能會出現餓死的情況。

PPRR： 依照優先程度，較優先的先做，而當優先程度相同時會使用 RR，因為優先抵即較高的的 process CPUtime 可能較長，故平均等待時間通常會較長，但是可以優先處理較為緊急的 process，不過有可能一直有優先等級較高的 process 進入，故可能會有餓死的情況。

HRRN： 計算 Response Ratio，Response Ratio 越大的先做，有時間升級的機制，故等待時間變長時會提高優先等級，若 CPUtime 較長的話則會降低優先等級，故平均等待時間不會太長，也因為有時間升級機制，所以不會有餓死的情況。