資料結構與進階程式設計（107-2）
作業六

作業設計：楊其恆
國立臺灣大學資訊管理學系

繳交作業時，請將第一題的答案以中文或英文作答後，以 PDF 檔上傳到 NTU COOL；***不接受紙本繳交***；第二至四題請至 PDOGS（`http://pdogs.ntu.im/judge/`）上傳一份 C++ 原始碼（以複製貼上原始碼的方式上傳）。這份作業的截止時間是 ***2019 年 5 月 20 日星期一凌晨 1:00***。在你開始前，請閱讀課本的第 12、13 章[1] 不接受遲交。

第一題

（40 分，每題 10 分）請回答以下數題。在以下各題之中，請假設所有的資料都是`int` 型態。

(a) 考慮一個以 link-based 實作的 sorted list，只有一個成員變數`listPtr`，指向一個儲存了 list 資料的 linked list。請使用 C++ 為這個 sorted list 實作以下的 operator overloading。

```
LinkedSortedList LinkedSortedList::opertor+(const LinkedSortedList&
    anotherList)
```

你可以使用所有有提過的 linked list 及 node 之函數。

(b) 給定一個 queue，請使用 C++ 將 queue 之中的內容從頭至尾依序印出，並且以逗號分隔。在實作時，請使用 queue 的方法來對其中的元素進行操作。

```
void display(Queue aQueue);
```

(c) 考慮一個類似於 queue 的 ADT deque，與 queue 的不同在於 queue 只能從最前端取出資料，但 deque 可以允許從頭尾取出資料或放入資料。舉例來說，如果一個 deque 當中有以下內容：
(front)2, 3, 1, 4, 5(back)
並對之進行以下操作

1. `add_front(7)`
2. `remove_back()`
3. `add_back(3)`

則其內容就會變為 (front)7, 2, 3, 1, 4, 3(back)。考慮一個定義如下的 deque：

```
class deque
{
private:
  Node* backPtr;
  Node* frontPtr;
```

[1]課本是 Carrano and Henry 著的 *Data Abstraction and Problem Solving with C++: Walls and Mirrors* 第六版。

```
public:
  deque();
  deque(const deque& aQueue);
  ~deque();
  bool isEmpty() const;
  bool add_front(const int& newEntry);
  bool add_back(const int& newEntry);
  bool remove_front();
  bool remove_back();
};
```

我們可以仿照 queue 的 linked-based implementation，實作它的`add_back()` 方法如下：

```
template< class ItemType>
bool dequeue::add_back(const int& newEntry)
{
  Node* newNodePtr = new Node(newEntry);

  if (isEmpty())
    frontPtr = newNodePtr; // The queue was empty
  else
    backPtr->setNext(newNodePtr); // The queue was not empty
  backPtr = newNodePtr; // New node is at back
  return true;
}
```

請使用 C++ 實作 deque 的`add_front` 方法，你可以使用 array-based 或 linked-based 方法進行實作。

(d) 承上題，請使用 C++ 實作 deque 的`remove_back` 方法，你可以使用 array-based 或 linked-based 方法進行實作。

第二題

（20 分）考慮一個服務台以及台前的一個隊列，當有顧客抵達時，如果櫃檯沒有人在使用，就會直接前往櫃檯開始接受服務。反之則會在隊列中排隊等待，直到櫃檯的人服務完畢。在這樣的情境下，如果顧客到達的時間間格為指數分配（以 λ 為參數），且每位顧客在櫃台接受服務的時間亦為指數分配（以 μ 為參數），亦即這些時間長度的大小，是由機率決定的。[2] 則根據機率模型，可以計算出每位顧客在隊列中的平均等待時間 (W)，以及該隊列平均每單位時間會有多少人在裡面等待 (N_Q)。其中前者可以透過以下公式計算得出

$$W = \frac{\lambda}{\mu(\mu - \lambda)} \tag{1}$$

[2]例如可能有 0.1 的機率是 20 秒，0.2 的機率是 30 秒，以此類推

在本題中，我們將以測資的型式，提供多筆顧客抵達時間以及服務時間資料，這個資料是根據給定的 2 個指數分配參數隨機生成的。請大家使用一個 queue 來模擬每位顧客的排隊與等待情形，並將這個結果與機率模型的計算結果比較，輸出誤差百分比。

輸入輸出格式

系統會提供一共 10 組測試資料，每組測試資料裝在一個檔案裡。每個檔案會有 $n+1$ 行，第一行會有一個整數 n 與二個浮點數 λ, μ，分別代表在顧客的數量、用於生成這筆測資的顧客到達時間的指數分配參數、以及服務時間的指數分配參數。第二行起的第 i 行之中，每一行會包含 2 個浮點數，分別代表第 i 個顧客的抵達時間，以及該顧客的服務時間，即從到櫃台開始接受服務，直到服務完畢這個區間的時間。各行中的所有數字皆以一個空白字元分隔，且大於 0。此外也有 $\mu > \lambda$，這是確保理論可以做出正確計算的條件之一。

讀入上述資料後，請根據給定的顧客資料，模擬排隊等候的情形，計算實際平均等待時間 w_p，以及理論上的平均等待時間 w_t，並將誤差百分比 $\frac{w_p - w_t}{w_t}$ 印出，印出時請以% 為單位，並且四捨五入至整數位。

舉例來說，如果輸入為

```
10 0.035 0.071
34.0 6.0
38.0 10.0
135.0 15.0
167.0 21.0
187.0 17.0
207.0 2.0
286.0 43.0
292.0 4.0
295.0 3.0
296.0 4.0
```

則輸出應該為

```
-14
```

因為根據輸入資料，我們可以做以下的模擬

顧客編號	抵達時間	開始接受服務的時間	服務結束時間	等待時間
1	34	34	40	0
2	38	40	50	2
3	135	135	150	0
4	167	167	188	0
5	187	188	205	1
6	207	207	209	0
7	286	286	329	0
8	292	329	333	37
9	295	333	336	38
10	296	336	340	40

總等待時間為 118，因此實際的平均每人等待時間為 11.8，而根據理論值計算出的理論平均等待時間為 $\frac{0.035}{0.071\times(0.071-0.031)} = 13.6933$，誤差百分比為 $\frac{11.8-13.6933}{13.6933} = -0.1383 = -13.83\%$

你上傳的原始碼裡應該包含什麼

你的.cpp 原始碼檔案裡面應該包含讀取測試資料、做運算，以及輸出答案的 C++ 程式碼。當然，你應該寫適當的註解。在進行運算時，建議對於所有浮點數皆使用`double` 型態，以增加精準度。

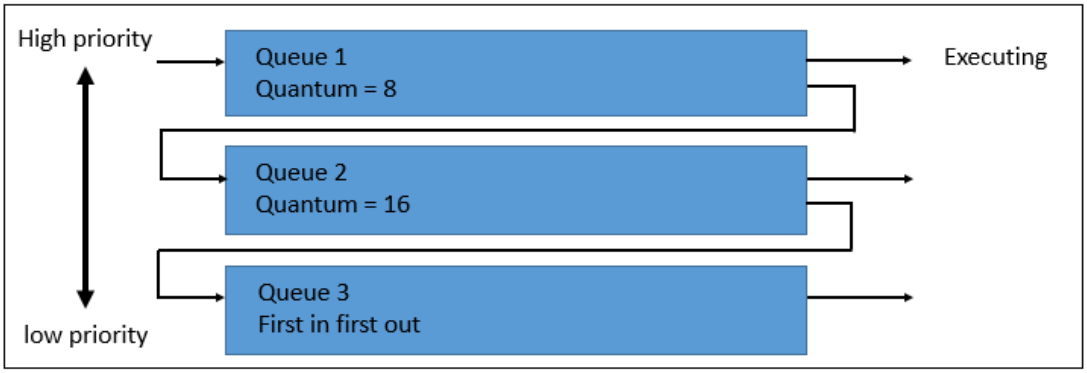
評分原則

這一題的分數都根據程式運算的正確性給分。PDOGS 會編譯並執行你的程式、輸入測試資料，並檢查輸出的答案的正確性。一筆測試資料佔 2 分。在本題中，你可以直接使用 C++ 內建的 queue library。

第三題

(40 分) 在電腦的運行當中，需要處理的一大問題是程序（process）的排程。當我們在電腦上執行一支程式時，他便會以 process 的形式在電腦中運行。然而功能簡單的處理器一次只能執行一個 process，因此如何決定執行的順序，好讓使用者能有最好的使用體驗，是排程的一大問題。在進行排程時，最常見的方法是使用一個稱為 multilevel feedback queue 的方法。其規則相當複雜，但在本題中我們考慮一個簡化版的情形。

如下圖所示，有多個存放 process 的 queue，process 在被執行前會在 queue 中等待，被放在越上層的 queue 可以獲得越高的優先執行權。之所以稱為優先，是因為當上層的 queue 有 process 在排隊時，就完全不會讓下層的有機會可以執行。除非上層的 queue 已空，才會輪到下層的執行。且執行過程中，一旦上層的 queue 有新的 process 進入，便會立刻被暫停，讓 CPU 去執行上層 queue 的 process。但是，為了避免一個單一 process 不斷的執行，而佔據了所有的 CPU 資源，這個排程方法會在每一個 queue 中定義每一輪的可執行時間（稱為 quantum）。



當輪到一個 process 執行時，他會被執行這個時間長度，如果沒有執行完畢，他的執行就會被暫停，並且被丟回 queue 的最後面等待下一輪的執行。而通常最下層的 queue 會被規定為沒有執行時間的限制，一旦輪到某個 process 執行，它便可以被執行直到結束。此外，每次被暫停時，process 被丟回的 queue 會是他原本執行的那層的下面一層，換句話說，一旦這個 process 沒有在第一輪被執行完畢，他的優先權就會被迫降低。第二輪若還是沒有執行完，就會被再往下丟一層。以此類推。使用這個排程方法的好處是，可以藉由多層的 queue 界定出哪些 process 是會耗費運算資源的，就將他的優先權降低，讓快速可以處理完的優先處理，如此一來平均的等待時間也會縮短。

在本題中，將會給定多個 process 的名稱、開始時間、以及所需的執行時間。請大家使用多個 queue，模擬上述的排程方法，並印出指定時間的各 process 執行情況，以及總共的等待時間。

輸入輸出格式

系統會提供 20 組的測試資料，每組測試資料裝在一個檔案裡。在每個檔案之中，會有若干行。第一行會有 n 個整數，分別以空白字元隔開。第一個整數為 n，代表總共有幾個 queue。第 2 至第 n 個整數分別代表第 1 到第 $n-1$ 個 queue 所允許的執行時間。第 n 個 queue 由於是最下層的，通常在進行排程時不會限制其執行時間，僅按照先進先出方式進行。因此在本題中我們不指定它允許的執行時間，任何一個 process 只要在這個 queue 中被輪到執行，除非較上層的 queue 中有新的 process 進入，皆可執行到結束。第二行起的每一行代表一個指令。指令分為以下二種：

1. 執行某個 process：這個指令會包含三個部分，分別以空白字元隔開，第一部分包含一個整數，表示現在的時間。第二部分包含一個字串，表示這個被執行的 process 的名稱；第三部分包含一個整數，表示這個 process 的執行時間。例如 23 xxx.exe 2 表示在時間為 23 時，有一個名為 xxx.exe 的 process 準備好要被執行了，如果它不用排隊就可以被執行，它會在時間為 25 時被執行完畢。
2. 列出某個 process 的狀態：這個指令包含二個部分，以空白字元隔開，第一部分包含一個整數，顯示現在的時間。第二部分為一個字串 ps。

在本題中，$n \leq 100$，且 process 的總數不會超過 500 個。讀入以上資料後，若遇到以上第二種指令，印出在給定時間時所有已出現過的 process 的剩餘執行時間，每一行印出一個 process 的資料，包含 process 的名稱，一個空白字元，以及它剩餘的執行時間。各 process 印出的順序請依照它出現的時間順序（非字典順序）。

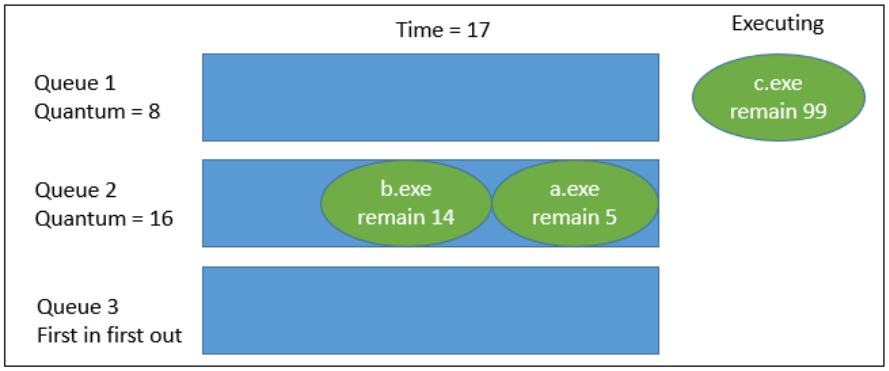
舉例來說，如果輸入是

```
3 8 16
0 a.exe 13
1 b.exe 22
5 c.exe 100
17 ps
26 d.exe 7
34 ps
100 ps
```

則輸出應該是

```
a.exe 5
b.exe 14
c.exe 99
a.exe 3
b.exe 13
c.exe 92
d.exe 0
a.exe 0
b.exe 0
c.exe 0
d.exe 62
```

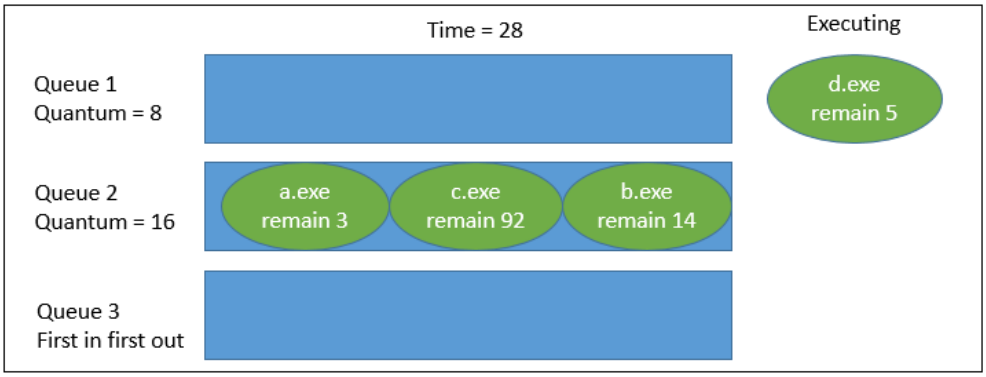
在本題的測試資料中，我們可以做以下的模擬。由於 a.exe 在時間為 0 時抵達，沒有其他 process 在排隊，因此進入第一個 queue 後可以立即被執行 8 個單位時間（從時間為 0 執行到時間為 8）。而 b.exe 及 c.exe 抵達時，因為 a.exe 正在被執行，進入第一個 queue 之中排隊。然而讓 a.exe 執行了 8 個單位時間後它並沒有被執行完（仍需 5 個單位時間），因此它會被中斷，並且進入第二層的 queue 中排隊，除非第一層的 queue 之中沒有其他 process，才會輪到它被執行。接著又過了 8 個單位時間執行 b.exe，同樣的它也沒有執行完畢，也被放到第二層的 queue，排在 a.exe 之後。因此當時間為 17 時，queue 之中的狀態應該如下圖所示：



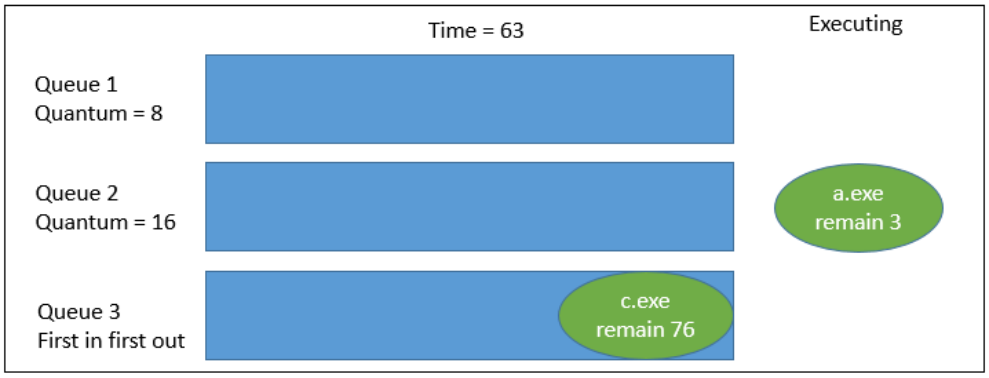
當時間來到 24 時，c.exe 也經過了 8 單位的時間執行，沒有執行完必，同樣被放到第二層 queue 中排隊。此時第一層的 queue 中已經沒有其他 process 要被執行了，所以開始執行排在第二層的 queue 之中

的 process。首先執行排在第一個的 a.exe，理論上應該要讓它執行 16 單位時間，然而在時間為 26 時（a.exe 執行了 2 單位時間後），d.exe 進入了第一層 queue，由於第一層的優先權高於第二層，因此直接將 a.exe 的執行中斷，並將它放回同一層 queue 的最末端重新排隊。

此時改為執行位在第一層的 d.exe，因此在時間為 28 時，queue 之中的狀態如下圖所示：



由於 c.exe 只需要 7 單位時間即可執行完畢，因此到了時間為 26+7=33 時，因為第一層 queue 已經沒有其他 process，所以回到第二層繼續執行排在最前面的 b.exe。b.exe 被執行了 13 單位時間後（時間來到 47），已執行完畢。因此此時讓 c.exe 開始執行 16 單位時間，時間來到 63 時，仍然沒有執行完，因此被放到最下層的 queue。queue 之中狀態如下圖所示：



此時剩下第二層的 queue 之中還有 a.exe 要執行 3 單位時間，讓它執行完畢後（時間為 66）。因為第一、二層的 queue 之中都沒有其他 process 排隊，所以可以執行第三層的 c.exe，此次執行除非第一、二層有新的 process 中途進入，否則 c.exe 可以直接被執行完畢。如上例中，自時間 66 至時間 100，讓 c.exe 執行了 34 單位時間，因此其剩餘執行時間為 42（92-16-34）。

你上傳的原始碼裡應該包含什麼

你的.cpp 原始碼檔案裡面應該包含讀取測試資料、做運算，以及輸出答案的 C++ 程式碼。當然，你應該寫適當的註解。

評分原則

這一題的分數都根據程式運算的正確性給分。PDOGS 會編譯並執行你的程式、輸入測試資料，並檢查輸出的答案的正確性。一筆測試資料佔 2 分。在本題中，你可以直接使用 C++ 內建的 queue library。

第四題

(20 分，加分題) 承上題，為了避免某些的 process 長時間執行的 process 在最下層 queue 之中佔據資源。現在我們將最下層的 queue 改為一個 priority queue，排隊的依據為進入時所剩餘的執行時間，剩餘時間越短者排在越前面，可以優先被執行。

輸入輸出格式

系統會提供 10 組的測試資料，每組測試資料裝在一個檔案裡。輸入輸出格式與上題相同。

舉例來說，如果輸入是

```
3 8 16
0 a.exe 13
1 b.exe 22
5 c.exe 100
17 ps
26 d.exe 7
34 ps
39 e.exe 37
100 ps
```

則輸出應該是

```
a.exe 5
b.exe 14
c.exe 99
a.exe 3
b.exe 14
c.exe 99
d.exe 0
a.exe 0
b.exe 0
c.exe 0
d.exe 76
e.exe 3
```

因為當 e.exe 被放入最下層的 queue 之中時（當時間為 90 時），剩下 13 單位時間，但 c.exe 雖然已經排在最下層 queue 的前面了，但它還剩下 76 單位時間要執行，因此 e.exe 會被放到 c.exe 前面被優先執行。

評分原則

這一題的分數都根據程式運算的正確性給分。PDOGS 會編譯並執行你的程式、輸入測試資料，並檢查輸出的答案的正確性。一筆測試資料佔 2 分。在本題中你可以直接使用 C++ 內建的 priority queue library。