Q1

```
// P_i
do {
    flag[i] = true;
    while (flag[j]) {
        if (turn == j) {
            flag[i] = false;
            while (turn == j)
            ; /* do nothing */
            flag[i] = true;
        }
    }
    /* critical section */
    turn = j;
    flag[i] = false;
    /* remainder section */
} while (true);
```

```
// P_j
do {
    flag[j] = true;
    while (flag[i]) {
        if (turn == i) {
            flag[j] = false;
            while (turn == i)
            ; /* do nothing */
            flag[j] = true;
        }
    }
    /* critical section */
    turn = i;
    flag[j] = false;
    /* remainder section */
} while (true);
```

首先說明共享變數flag及turn代表的意義

共享變數 : flag[2] of Boolean。(初值皆為 False)

共享變數 : turn: int。(初值皆為0)

flag[i] = True P_i 有意願進入C.S; flag[i] = False P_i 無意願進入C.S。
Turn = i or j 代表哪個process目前擁有可以進入C.S的權限。

接著證明符合critical-section problem 三大特性:

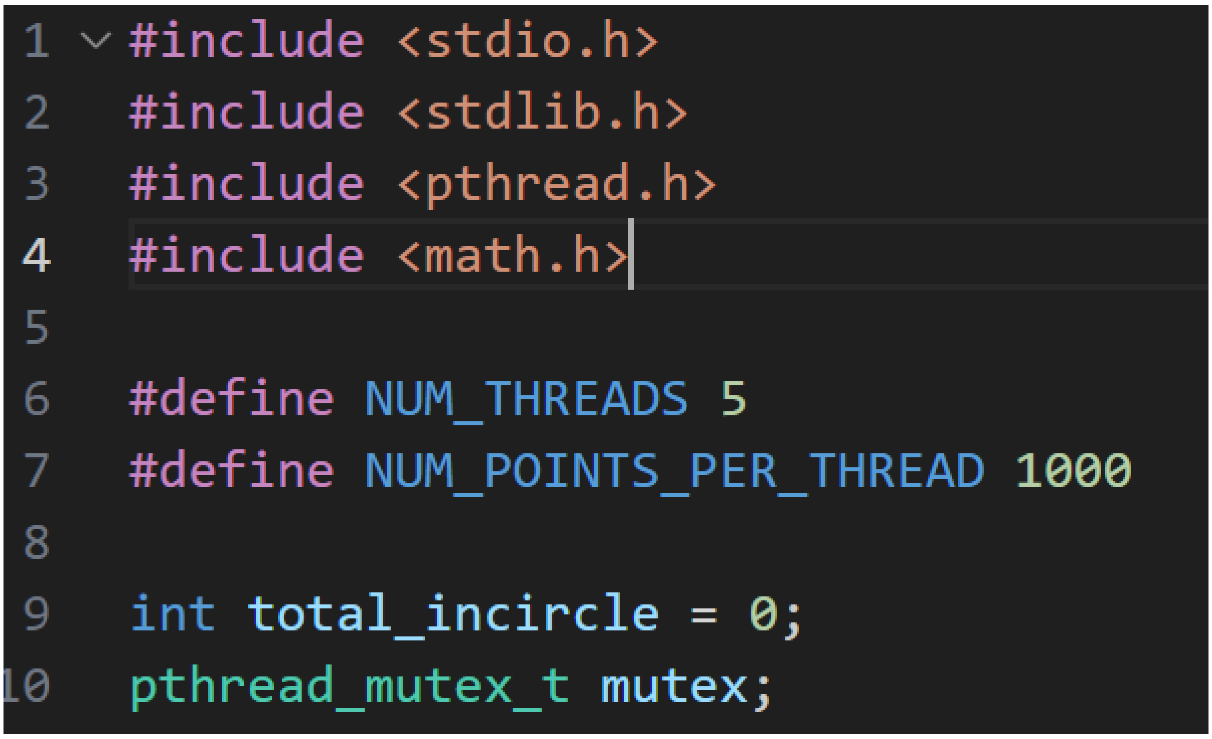
1. Mutual Exclusion
   - 假設P_i&P_j皆有意願進入C.S且i ≠ j，由於turn只會是i or j不可能同時是i & j，因此只有與turn對應的processs能夠進入C.S，而與turn不對應的process只能等待直到另一個process結束C.S將turn權限轉換才能進入C.S，符合Mutual Exclusion假設確保同一時間只有一個process能夠進入C.S。

2. Progress
   - 情況一: 假設只有P_i有意願進入且P_j無意願進入C.S不管turn = i or j皆會跳出第三行`while (flag[j])` 直接讓P_i進入C.S，符合Progress假設。
   - 情況二:假設P_i&P_j皆有意願進入C.S，只有與turn對應的process才擁有進入C.S的權限，在演算法中第四行`if (turn == j)`會檢查turn權限是否在對方，如果是在對方會透過第五行`flag[i] = false`將自身狀態改成無意願進入同時進入第六行`while (turn == j) do nothing`直到timer結束切換回擁有turn權限的process，此時因為已經將沒有turn權限的process狀態改成無意願進入就會像情況一可以跳出第三行`while (flag[j])` 直接進入C.S，符合Progress假設確保擁有turn權限且有意願進入的process一定能夠優先進入C.S。

3. Bounded-waiting
   - Dekker's演算法透過交替轉換turn值確保Bounded-waiting，假設P_i&P_j皆有意願進入C.S且turn = i，藉由Progress特性擁有turn權限且有意願進入的process一定能夠優先進入C.S，在結束的同時會轉換turn值`turn = j;`給正在等待的process，這邊舉一個比較極端的例子在turn轉換且進入下一個迴圈`flag[i] = true`此時也會因為turn已經轉換透過`if (turn == j)` 檢查是否有權限並透過`flag[i] = false;`將自身狀態改成無意願進入同時進入第六行`while (turn == j) do nothing`直到timer結束，這樣的設計機制使得 P_i 無法再度早於 P_j 進入 C.S，一定是 P_j 先進入 C.S.，所以 P_j 至多等待一次後即可進入 C.S.，防止一個process能夠無限期地阻塞另一個process。

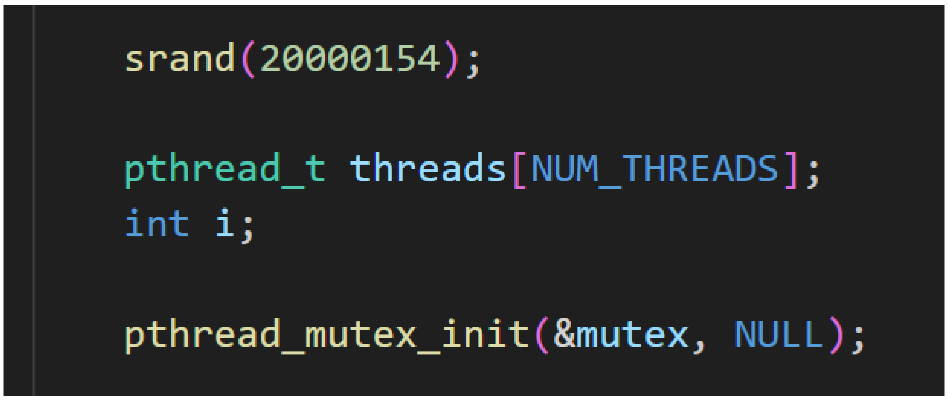
Q2

1. include相關library、定義worker_thread數量&每個word_thread所要產生的隨機點數量、Pthreads Mutex Locks和統計總共落在圓內點數量兩個global variable。



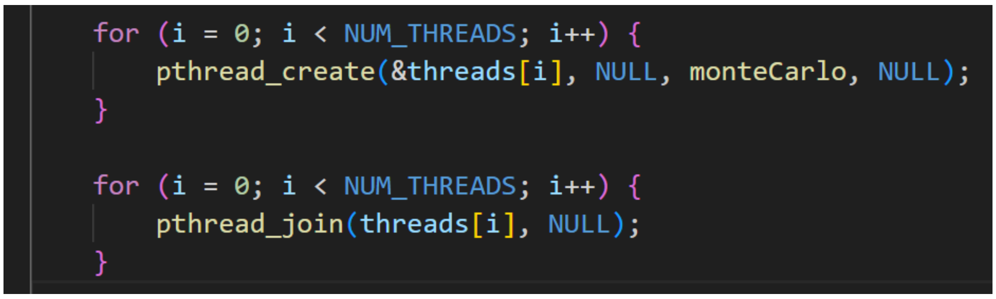
```c
#include <stdio.h>
#include <stdlib.h>
#include <pthread.h>
#include <math.h>

#define NUM_THREADS 5
#define NUM_POINTS_PER_THREAD 1000

int total_incircle = 0;
pthread_mutex_t mutex;
```

2. 選定一個幸運數字做為隨機種子及初始化5個worker_thread和Mutex。



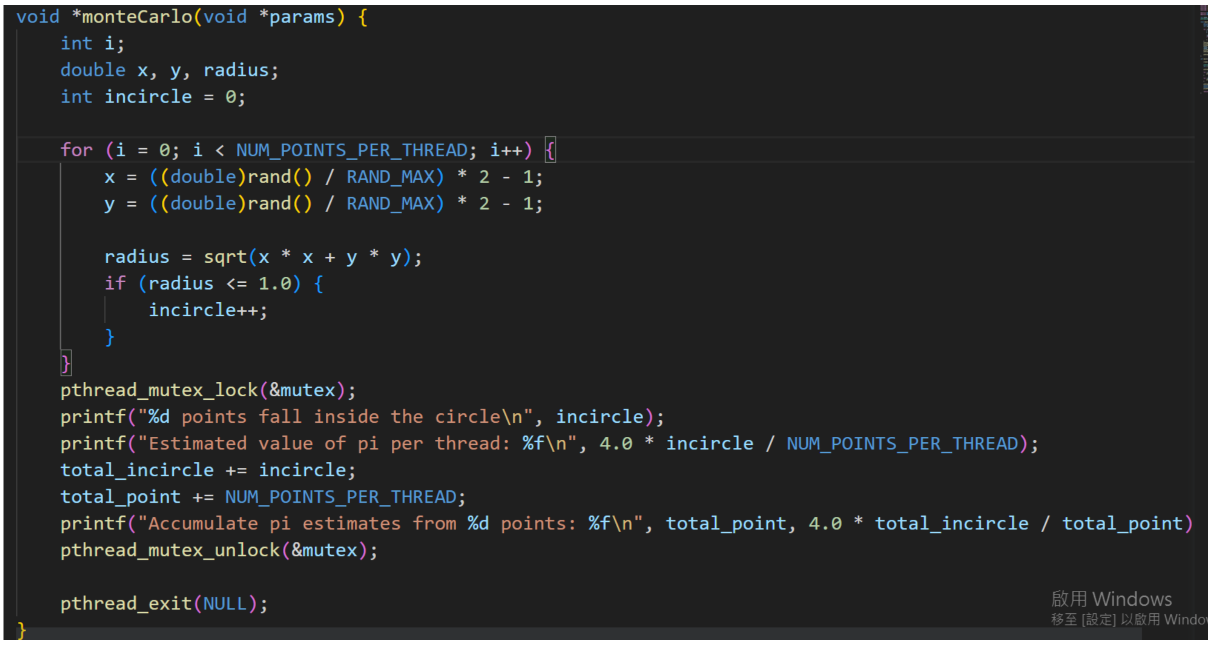
```c
    srand(20000154);

    pthread_t threads[NUM_THREADS];
    int i;

    pthread_mutex_init(&mutex, NULL);
```

3. 創建5個worker_thread分配任務計算蒙地卡羅近似圓周率，同時使用pthread_join() api 讓main thread等待worker_thread結束再繼續執行並回收其資源，而為了能夠達到multithread平行運行將pthread_create及pthread_join分成不同loop進行。



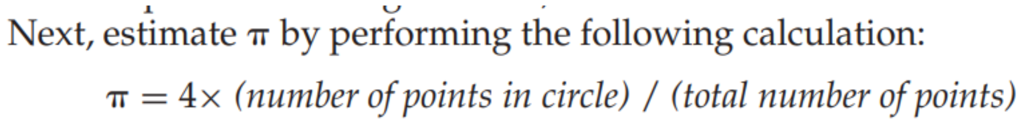
```c
    for (i = 0; i < NUM_THREADS; i++) {
        pthread_create(&threads[i], NULL, monteCarlo, NULL);
    }

    for (i = 0; i < NUM_THREADS; i++) {
        pthread_join(threads[i], NULL);
    }
```

4. 接著頗析每個worker_thread執行計算蒙地卡羅近似圓周率的函式，進行助教要求1000次生成隨機點，這裡確保生成的隨機點能落在(-1,1)之間同時透過計算半徑判斷是否落在圓內，若有落在圓內則將數量加到前面提到統計落在圓內總數的global variable，值得注意的是為了確保global variable在各個worker_thread不會inconsistency使用mutex_lock確保這個加的過程是atomic的。



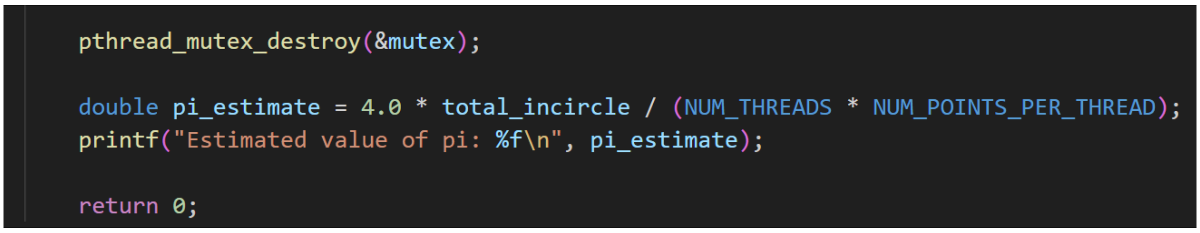
```c
void *monteCarlo(void *params) {
    int i;
    double x, y, radius;
    int incircle = 0;

    for (i = 0; i < NUM_POINTS_PER_THREAD; i++) {
        x = ((double)rand() / RAND_MAX) * 2 - 1;
        y = ((double)rand() / RAND_MAX) * 2 - 1;

        radius = sqrt(x * x + y * y);
        if (radius <= 1.0) {
            incircle++;
        }
    }
    pthread_mutex_lock(&mutex);
    printf("%d points fall inside the circle\n", incircle);
    printf("Estimated value of pi per thread: %f\n", 4.0 * incircle / NUM_POINTS_PER_THREAD);
    total_incircle += incircle;
    total_point += NUM_POINTS_PER_THREAD;
    printf("Accumulate pi estimates from %d points: %f\n", total_point, 4.0 * total_incircle / total_point)
    pthread_mutex_unlock(&mutex);

    pthread_exit(NULL);
}
```

5. 等待所有worker_thread完成統計後釋放mutex佔用的資源使其無效最後再由main_thread完成圓周率的估計。



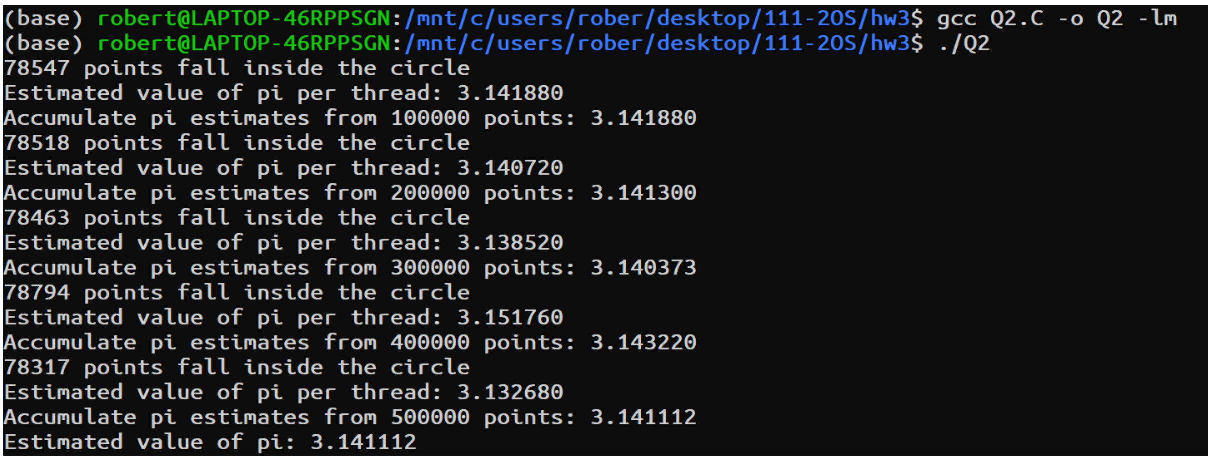
Next, estimate π by performing the following calculation:

$$\pi = 4\times \textit{(number of points in circle) / (total number of points)}$$

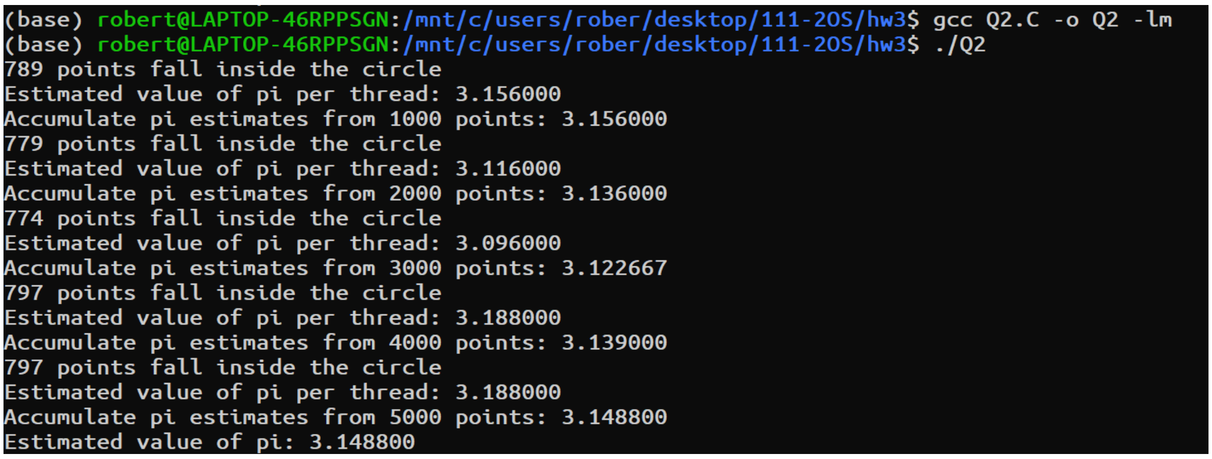


```c
    pthread_mutex_destroy(&mutex);

    double pi_estimate = 4.0 * total_incircle / (NUM_THREADS * NUM_POINTS_PER_THREAD);
    printf("Estimated value of pi: %f\n", pi_estimate);

    return 0;
```

6. 螢幕截圖結果，上面是將每個worker_thread生成隨機點增加至100000，下面是助教規定1000個點，可以發現生成隨機點越多圓周率估計越精準。



```
(base) robert@LAPTOP-46RPPSGN:/mnt/c/users/rober/desktop/111-2OS/hw3$ gcc Q2.C -o Q2 -lm
(base) robert@LAPTOP-46RPPSGN:/mnt/c/users/rober/desktop/111-2OS/hw3$ ./Q2
78547 points fall inside the circle
Estimated value of pi per thread: 3.141880
Accumulate pi estimates from 100000 points: 3.141880
78518 points fall inside the circle
Estimated value of pi per thread: 3.140720
Accumulate pi estimates from 200000 points: 3.141300
78463 points fall inside the circle
Estimated value of pi per thread: 3.138520
Accumulate pi estimates from 300000 points: 3.140373
78794 points fall inside the circle
Estimated value of pi per thread: 3.151760
Accumulate pi estimates from 400000 points: 3.143220
78317 points fall inside the circle
Estimated value of pi per thread: 3.132680
Accumulate pi estimates from 500000 points: 3.141112
Estimated value of pi: 3.141112
```



```
(base) robert@LAPTOP-46RPPSGN:/mnt/c/users/rober/desktop/111-20S/hw3$ gcc Q2.C -o Q2 -lm
(base) robert@LAPTOP-46RPPSGN:/mnt/c/users/rober/desktop/111-20S/hw3$ ./Q2
789 points fall inside the circle
Estimated value of pi per thread: 3.156000
Accumulate pi estimates from 1000 points: 3.156000
779 points fall inside the circle
Estimated value of pi per thread: 3.116000
Accumulate pi estimates from 2000 points: 3.136000
774 points fall inside the circle
Estimated value of pi per thread: 3.096000
Accumulate pi estimates from 3000 points: 3.122667
797 points fall inside the circle
Estimated value of pi per thread: 3.188000
Accumulate pi estimates from 4000 points: 3.139000
797 points fall inside the circle
Estimated value of pi per thread: 3.188000
Accumulate pi estimates from 5000 points: 3.148800
Estimated value of pi: 3.148800
```