```
flag[2] = {false, false};

turn = 0;
```

```
do{

flag [i] = true;

turn = j;

while(flag [j] && turn == [j]);

/* Critical Section */

flag [i] = false;

/* Remainder Section */

}while (true);
```

```
do{

flag [j] = true;

turn = i;

while(flag [i] && turn == [i]);

/* Critical Section */

flag [j] = false;

/* Remainder Section */

}while (true);
```

- Shared boolean variable lock, initialized to false

```
do{

while(test_and_set(&lock))

; /* do nothing */

/* Critical Section */

lock = false;

/* Remainder Section */

}while (true);
```

N = عدد العمليات التي سوف تنفذ

```
Int turn = 0;
```

```
while(true)

{

while(turn != i);

/* Critical Section */

turn = (turn + 1) % N;

/* Remainder Section */

}
```

- Shared integer lock initialized to 1;

```
while (true) {
while(compare_and_swap(&lock, 0, 1) != 0)
; /* do nothing */
/* Critical Section */
lock = 0;
/* Remainder Section */
}
```

- Shared integer lock initialized to 0;

```
while (true) {

while(compare_and_swap(&lock, 1, 1) != 0)

; /* do nothing */

/* Critical Section */

lock = 0;

/* Remainder Section */

}
```

Semaphore chopstick [5] = { };

Semaphore number_of_philisophers = ;

```
do{
think();


wait(chopstick[i]);
wait(chopstick[(i + 1) % 5]);
      /* eat */
signal(chopstick[i]);
signal(chopstick[(i + 1) % 5]);

}while (true);
```

Fill in the empty spaces in a way that, the given solution will have no deadlock and then explain your reasons below.

Using only one semaphore variable

- Produce C is executed first
- Produce B or Produce A may be the second

Semaphore ……………… ………………………..

Process A:

```
{

//Produce A;

}
```

Process B:

```
{

//Produce B;

}
```

Process C:

```
{

//Produce C;

}
```

Using semaphore variable

- Produce C is executed first
- Produce B or Produce A may be the second

Semaphore ……………… ……………………….

Process A:

```
{

//Produce A;

}
```

Process B:

```
{

//Produce B;

}
```

Process C:

```
{

//Produce C;

}
```