HW1

2.1. The following solutions are intended to solve the two-thread critical section problem. Thread0 and Thread1 execute the same code, which is shown below. When Thread0 executes the code, i is 0 and j is 1. When Thread1 executes the code, i is 1 and j is 0. Variable *turn* can be initialized to 0 or 1. The execution of the `break` statement inside the first `while`-loop transfers control to the statement immediately following the loop. For each solution, determine whether it guarantees mutual exclusion. If mutual exclusion is guaranteed, determine whether it guarantees progress. If mutual exclusion and progress are guaranteed, determine whether it guarantees bounded waiting. Justify your answers.

(a)

```
while (true) {
        flag[i] = true;                 (1)
        while (flag[j]) {               (2)
            if (turn != i) {            (3)
              flag[i] = false;          (4)
              while (turn == j) {;}     (5)
              flag[i] = true;           (6)
              break;                    (7)
            }
        }
        critical section
        turn = j;                       (8)
        flag[i] = false;                (9)
        noncritical section
}
```

(b)

```
while (true) {
        flag[i] = true;                 (1)
        while (flag[j]) {               (2)
            flag[i] = false;            (3)
            while (turn == j) {;}       (4)
            flag[i] = true;             (5)
        }                               (6)
        critical section
        turn = j;                       (7)
        flag[i] = false;                (8)
        noncritical section
}
```

(c)

```
while (true) {
    flag[i] = true;                    (1)
    turn = j;                          (2)
    while (flag[j] && turn== i) {;}    (3)
    critical section
    flag[i] = false;                   (4)
    noncritical section
}
```

2.5. Suppose that we switch the order of the first two statements in Peterson's algorithm:

```
boolean intendToEnter0 = false, intendToEnter1 = false;
int turn; // no initial value for turn is needed.
```

T0		T1	
while (true) {		while (true) {	
turn = 1;	(1)	turn = 0;	(1)
intendToEnter0 = true;	(2)	intendToEnter1 = true;	(2)
while (intendToEnter1 && turn == 1) {;}	(3)	while (intendToEnter0 && turn == 0) {;}	(3)
critical section	(4)	critical section	(4)
intendToEnter0 = false;	(5)	intendToEnter1 = false;	(5)
noncritical section	(6)	noncritical section	(6)
}		}	

Does the modified solution guarantee mutual exclusion? Explain. If yes, does the modified solution guarantee progress and bounded waiting? Explain.

2.7. For the bakery algorithm in Section 2.2.2, the value of *number[i]* is not bounded.

(a) Explain how the value of *number[i]* grows without bound.

(b) One suggestion for bounding the value of *number[i]* is to replace the statement

```
number[i] = max(number) + 1;
```

with the statement

```
// numThreads is # of threads in the program
number[i] = (max(number) + 1) % numThreads;
```

Is this suggestion correct? Explain.