Note: YOU MUST DO THE HOMEWORK BY YOURSELF. If you have difficulties in solving a question you may discuss it with friends, BUT you MUST phrase, write and formulate the answers by yourself, after you understand the solution. The language of the solution must be entirely your own.

Notice 2: Unless your hand writing is extremely clear, you should type (Word, or LaTeX, etc.) and print your homework.

Notice 3: Put your name and id number on each page of the solution.
1. Let us define $r$-bounded waiting for a given mutual exclusion algorithm to mean that if $D_A^j \rightarrow D_B^k$ then $CS_A^j \rightarrow CS_B^{k+r}$ Show that the Filter lock allows some threads to overtake others an arbitrary number of times, that is, there is no value of $r$ for which it provides $r$-bounded waiting.

2. The $l$-exclusion problem is a variant of the starvation-free mutual exclusion problem. We make two changes: as many as $l$ threads may be in the critical section at the same time, and fewer than $l$ threads might fail (by halting) in the critical section. An implementation must satisfy the following conditions:

   **$l$-Exclusion:** At any time, at most $l$ threads are in the critical section.

   **$l$-Starvation-Freedom:** If less than $l$ threads are in the critical section, each thread that wants to enter it eventually succeeds (even if some threads halt).

Modify the Filter mutual exclusion algorithm to turn it into an $l$-exclusion algorithm. Hint: notice that there are two conditions to be satisfied.
3. **Bakery Mutual Exclusion:** In each of the following subsections (a, b, and c) we replace one line of the Bakery Mutual Exclusion Algorithm that we saw in class with a code segment. The replaced line

$$label[i] = 1 + \max\{label[j], \mid 1 \leq j \leq n\}$$

With the code shown (showing more assembly like actual implementation). For each subsection, is the replacement correct?, i.e., is the resulting algorithm correct, prove your answer.

(a)  
```
local1 = 0
for local2 = 1 to n do
    local3 = label[local2]
    if local1 < local3 then local1 = local3 fi
od
label[i] = 1 + local1
```

(b)  
```
local1 = i
for local2 = 1 to n do
    if label[local1] < label[local2] then local1 = local2 fi
od
label[i] = 1 + label[local1]
```

(c)  
```
local1 = i
for local2 = 1 to n do
    if label[local1] \leq label[local2] then local1 = local2 fi
od
label[i] = 1 + label[local1]
```

4. Let $A$ be an arbitrary deadlock-free mutual exclusion algorithm in which the number of steps in the exit code of $A$ depends on the activity of the other processes (e.g., it has a loop with a wait statement). In the following, $A$ is modified, so that the new algorithm satisfies the requirement that the exit code be of constant number of steps. The modified algorithm uses one additional shared bit, called $flag$, is used.

```
1 entry code of original $A$;
2 while flag do skip od;
3 flag := true;
4 exit code of original $A$; /* Whose step complexity may depend on other processes */
5 critical section;
6 flag := false;
```

Figure 1: The modified Mutual Exclusion Algorithm
Does the modified algorithm satisfy deadlock-freedom and/or mutual exclusion (for any A)? Prove your answers.