Exam 3

Question 1-

1- Not in our syllabus.
2- A simple lock have can't retain a process. Indeed, a process which read a simple lock during a program either enter in the block or not. But there is no way for the process to stay blocked in the simple lock and he can go. The programmer can't have any control on it.
A re-entrant lock may solve this problem by define a line where the process should return.
Here an example:

```
1       if(blablabla) {
2               blablabla
3               blublublu
4               bliblibli
5               blobloblo
6               blebleble
7               bloubloublou
8               blaiblaiblai
9               GO TO line 1
10      }
```

3-
a- TRUE
b- FALSE
c- TRUE
d- FALSE
4-
a-
Shared:
turn[1] = turn[2] = turn[3] = turn[4] = {false}
turn[0] = true

code for i:

```
while(turn[i] == false) {}
<CS>
turn[i] = false
if (i = 0)
        turn[2] = true
else if (i = 2)
        turn[4] = true
else if (i = 4)
        turn[1] = true
else if (i = 1)
        turn[3] = true
```

b-
Shared:
int turn_to = 0
semaphore mutex = 0

code for i:

```
if (i != 0  && turn_to == 0)
        down(mutex)
if (i != 2  && turn_to == 2)
        down(mutex)
if (i != 4  && turn_to == 4)
        down(mutex)
if (i != 1  && turn_to == 1)
        down(mutex)
if (i != 3  && turn_to == 3)
        down(mutex)
<CS>
if(turn_to == 0)
        turn_to = 2
else if(turn_to == 2)
        turn_to = 4
else if(turn_to == 4)
        turn_to = 1
else if(turn_to == 1)
        turn_to = 3
else if(turn_to == 3)
        turn_to = -1
up(mutex)
```

c-
Shared:
semaphore zero = 1
one = 0
two = 0
three = 0
four = 0

code for process i:

```
if(i == 0)
        down(zero)
if (i == 1)
        down(one)
if (i == 2)
        down(two)
if (i == 3)
        down(three)
if (i == 4)
        down(four)
<CS>
if(i == 0)
        up(two)
```

```
if (i == 1)
        up(three)
if (i == 2)
        up(four)
if (i == 4)
        down(one)
```

Question 2-

```
shared:
int  turn = 0;
```

code for process 0:

```
<entry code>
while(compare_and_swap(turn, 0, -1)) {}
<CS>
<exit code>
turn = 0
```

code for process 1:

```
while(compare_and_swap(turn, 0, -1)) {}
<CS>
<exit code>
turn = 0
```

This question is already solved in the assignment 6.

Question 3-

1- Only one process, two processes doesn't provide starvation freedom.
a. mutual exclusion and deadlock free, and starvation free. (since he is alone)
b. k = 2, not process 0 may be stuck forever in the entry code as the next scenario shows:
process 1 enter in the CS, then, while process 0 is blocking in the down(R) since R = 0.
Then, process 1 reinitialize R to 1 with down(R) but he can going faster that 0 and once again make down(R). Repeating this scenario we don't provide starvation freedom.
2- Only one process, two processes doesn't provide mutual exclusion.
a. mutual exclusion and deadlock free, and starvation free. (since he is alone)
b. trivial.
3- Only one, same as (1).
4- v1 = 9, v2 = 90, v3 = 636