CSC 262
Homework #6
Due at the start of class on Wednesday, March 28.

1. **Mutual Exclusion**. The software-based mutual exclusion protocol discussed in class works with only two processes. Suppose that we have *n* processes that must have mutually exclusive access to a critical section. The *bakery algorithm* works in this setting:

```
1:  entry_i:  choosing[i]:=true;
2:    number[i]:=max(number[0]..number[n-1])+1;
3:    choosing[i]:=false;
4:    for j:=0 to n-1 do
5:      while (choosing[j]) do skip;
6:      while (number[j]!=0)
7:      and (number[j],j)<(number[i]),i) do skip;
          // note:  (a,b)<(c,d) means (a<c) or (a=c and b<d)
8:    end;


9:  exit_i:  number[i]:=0;
```

Examine the algorithm carefully. You may wish to run a few simulations by hand, imagining what can happen under different circumstances.

a. Suppose a process is in the critical section, and one or more additional processes are waiting for it at lines 6-7. If another new process begins the entry protocol at line 1, what can you say about the number assigned to it relative to the numbers already assigned? What can you say about when it will enter its critical section, relative to the processes already waiting?

b. Why is the test in line 7 **`(number[j],j) < (number[i]),i)`** instead of just **`number[j] < number[i]`**? Give a set of circumstances that could result in two processes being assigned the same number.

c. Consider the simpler entry protocol below. Explain how it could fail to provide mutual exclusion, and therefore why the more complex protocol above is necessary. (Hint: consider the atomic steps necessary to achieve line 1 for one process, and what might be interleaved between those steps.)

```
1:  entry_i:  number[i]:=max(number[0]..number[n-1])+1;
2:    for j:=0 to n-1 do
3:      while (number[j]!=0)
4:      and (number[j],j)<(number[i]),i) do skip;
5:        // note:  (a,b)<(c,d) means (a<c) or (a=c and b<d)
6:    end;
```

d. Put the observations above together to form a coherent argument that the bakery algorithm ensures mutual exclusion of the critical sections for all n processes. (Devise a formal proof for extra credit.)

2. **Interlock Instructions**. Consider the following interlock instruction, which is can be executed atomically in hardware:

```
INC(s,x):  ⟨x:=s; s:=s+1⟩
```

a. This should simplify the distribution of bakery tickets. Use it to give a new, simpler implementation of the bakery algorithm from question 1. Show the old protocol in one color ink and your modifications and edits in another color. Make as many simplifications as possible while maintaining the integrity of the protocol.

b. Discuss whether the value of `s` is bounded (i.e., whether it has a maximum value). Explain the practical consequences of this and how the boundedness of `s` interacts with the argument for of mutual exclusion you gave in part a of this problem. (No need to modify your answer given above, however.) Can you suggest any way of addressing this issue?