1. The pseudocode of Figure 1 illustrates the basic `push()` and `pop()` operations of an array-based stack. Assuming that this algorithm could be used in a concurrent environment, answer the following questions:

```
push(item) {
  if (top < SIZE) {
    stack[top] = item;
    top++;
  }
  else
    ERROR
}

pop() {
  if (!is_empty()) {
    top--;
    return stack[top];
  }
  else
      ERROR
}

is_empty() {
  if (top == 0)
    return true;
  else
    return false;
}
```

Figure 1 Array-based stack

   a. What data have a race condition?

   b. How could the race condition be fixed?

2. Assume that a system has multiple processing cores. For each of the following scenarios, describe which is a better locking mechanism—a *spinlock* or *a mutex lock where waiting processes sleep while waiting* for the lock to become available:

- The lock is to be held for a short duration.
- The lock is to be held for a long duration.
- A thread may be put to sleep while holding the lock.

3. Assume that a context switch takes T time. Suggest an upper bound (in terms of T) for holding a spinlock. If the spinlock is held for any longer, a mutex lock (where waiting threads are put to sleep) is a better alternative.

4. A multithreaded web server wishes to keep track of the number of requests it services (known as ***hits***). Consider the two following strategies to prevent a race condition on the variable `hits`. The first strategy is to use a basic mutex lock when updating hits:

```
int hits;
mutex_lock hit_lock;
hit_lock.acquire();
hits++;
hit_lock.release();
```

A second strategy is to use an atomic integer:

```
Atomic_t hits;
Atomic_inc(&hits);
```

Explain which of these two strategies is more efficient.

5. Demonstrate that monitors and semaphores are equivalent to the degree that they can be used to implement solutions to the same types of synchronization problems.

6. Describe how the `signal()` operation associated with monitors differs from the corresponding operation defined for semaphores.

7*. Consider a system consisting of processes P_1, P_2, …, P_n, each of which has a unique priority number. Write a monitor that allocates three identical printers to these processes, using the priority numbers for deciding the order of allocation.

8*. A file is to be shared among different processes, each of which has a unique number. The file can be accessed simultaneously by several processes, subject to the following constraint: the sum of all unique numbers associated with all the processes currently accessing the file must be less than n. Write a monitor to coordinate access to the file.

9. Consider the following code

```
const int n = 50;
int tally;
void total()
{
  int count;
    for (count = 1; count<= n; count++) {
    tally++;
  }
}
void main()
{
  tally = 0;
  parbegin (total (), total ());
  write (tally);
}
```

Determine the proper lower bound and upper bound on the final value of the shared variable `tally` output by this concurrent program. Assume processes can execute at any relative speed, and a value can only be incremented after it has been loaded into a register by a separate machine instruction.
Note that the construct `parbegin (P1, P2, . . . , Pn)` means the following: suspend the execution of the main program; initiate concurrent execution of procedures P1, P2, . . . , Pn; when all of P1, P2, . . . , Pn have terminated, resume the main program.

10. The code on the side is a software solution to the mutual exclusion problem for two processes. It shares two variables:

```
int turn;
Boolean blocked[2]
```

`turn` is initialized to 0 and `blocked` values are initialized to `false`. Find a counterexample that demonstrates that this solution is incorrect.

```
boolean blocked [2];
int turn;
void P (int id)
{
  while (true) {
    blocked[id] = true;
    while (turn != id) {
      while (blocked[1-id]);
      turn = id;
    }
    /* critical section */
    blocked[id] = false;
    /* remainder */
  }
}
```

11. Provide an example using the producer-consumer problem that **illustrates** why atomic variables are not enough for synchronization for all circumstances.
    Note: Your answer should include which variable to be atomic and in which part the use of atomic variable is not enough for synchronization.
12. On modern multi-core computing systems *spinlocks* are widely used in many Operating systems. What is the advantage that spinlocks provide? And When this advantage applies?
13. In the following code, three processes produce output using the routine `printf` and synchronize using two semaphores **L** and **R**.

```
//process 1
while( 1 ){
    L.wait();
    printf("C");
    R.signal();
}
```

```
//process 2
while( 1 ){
    R.wait();
    printf("A");
    printf("B");
    R.signal();
}
```

```
//process 3
while( 1 ){
    R.wait();
    printf("D");
}
```

Semaphore L is initialized to 3 and semaphore R is initialized to 0.

a. What is the smallest number of A's that might be printed when this set of processes runs?
b. Is CABABDDCABCABD a possible output sequence when this set of processes runs?
c. Is CABACDBCABDD a possible output sequence when this set of processes runs?