Worksheet 5.5: Questions on Assignment 1

Consider the following functions that implement the producer and the consumer in Assignment 1. Assuming that the rest of the code is the same as in the give templates, answer the following questions. Each question is **independent** of other questions.

```
void Producer(int bufSize, int itemCnt, int randSeed){
        int i, in = 0, out = 0, val;

1        for (i=0; i<itemCnt; i++) {
2                while((GetIn()+1)%bufSize == GetOut());
3                val = GetRand(0, 1000);
4                WriteAtBufIndex(in, val);
5                in = (in + 1)%bufSize;
6                SetIn(in);
        }
}
void Consumer(){
//Code to open shared memory block and map it to gShmPtr
1    int bufSize = GetBufSize();
2    int itemCnt = GetItemCnt();
3    int in = GetIn();
4    int out = GetOut();
5    for(i=0; i<itemCnt; i++){
6          while(GetIn() == GetOut());
7          val = ReadAtBufIndex(out);
8          out = (out + 1)%bufSize;
9          SetOut(out);
    }
```

1. Which line in the above code does each of the following? Make sure you specify whether that line is in the Producer or in the Consumer

   Waiting if the buffer is empty — Line 6 in Consumer

   Updating the value of *in* in the shared memory buffer — Line 6 in Producer

2. Assuming 4 items and a buffer size of 3, which of the following outputs are correct and which are incorrect? See the prints in the above code. Circle Right or Wrong.

Prod 0	Prod 0	Prod 0	Prod 0
Cons 0	Prod 1	Prod 1	Prod 1
Prod 1	Cons 0	Cons 0	Cons 0
Prod 2	Prod 2	Prod 2	Cons 1
Cons 1	Cons 1	Prod 3	Prod 2
Prod 3	Prod 3	Cons 1	Cons 2
Cons 2	Cons 2	Cons 2	Cons 3
Cons 3	Cons 3	Cons 3	Prod 3
Right Wrong	**Right** Wrong	Right **Wrong**	Right **Wrong**

3. Use the table below to trace the consequences of making the following changes**,** assuming that everything else remains the same. Assume that there is a **single** producer and a **single** consumer. Let the buffer size be **m**, the number of items be **n**, **n** > **m**. **Inconsistent** answers will get zero credit.

Change	Will it work correctly?	Items Produced	Items Consumed	Explanation
Delete ***SetIn()*** on Line 6 of the Producer	No	n	0	in won't get updated in shared mem. So, cons. would think that nothing has been produced, and would thus consume nothing. Prod. would forget that it has produced an item, and would thus keep producing items but it will overwrite some items.

Change	Will it work correctly?	Items Produced	Items Consumed	Explanation
Delete ***SetIn()*** on Line 6 of the Producer and change ***GetIn()*** on Line 2 of Producer to in.	No	m-1	0	in won't get updated in shared mem. So, cons. would think that nothing has been produced, and would thus consume nothing. Prod. would keep producing until it fills the buf with m-1 items. Then prod. and cons. will be waiting for each other forever.

4. What's the range of possible values that may be returned by **GetIn()** on Line 3 and **GetOut()** on Line 4 of the **Consumer**? Assume a single producer and a single consumer. Let the buffer size be **m**, the number of items be **n**, **n** > **m**.

   Range of all possible values returned by GetIn() on Line 3 in Consumer: 0 to m-1

   Range of all possible values returned by GetOut() on Line 4 in Consumer: 0

5. Suppose that we modify the problem such that there is one producer and multiple consumers with the following specifications:
   - The parent process forks multiple child process, each child running the same **Consumer** code.
   - The producer (in the parent process) produces itemCnt items, and an item may be consumed by any consumer. An item can be consumed **only once** (once one consumer consumes it, the buffer cell may be used by the producer to produce new items).
   - After reading an item, a consumer spends a significant amount of time processing the item. So, the consumer code will have a function named ProcessItem() somewhere after ReadAtBufIndex() (you need to figure out the best place for it). ProcessItem takes significant time but does independent work that each consumer can do in parallel with other consumers.

- When all produced items have been consumed, each consumer should detect this (you will need to figure out a mechanism for doing this) and terminate itself by calling exit(0).

  Modify the above code to solve this single-producer multiple-consumer problem. Don't worry about the parent code that forks multiple child processes. Only give the necessary changes to the Consumer and Producer functions (if any) and describe any changes that need to be made to the header of the shared memory block. Your code will be graded on both correctness and efficiency. Correct but inefficient code will get only partial credit.

Key idea: Need to add to the shared memory header a fifth field that tracks the number of items consumed. Since multiple processes will be updating this shared variable as well as the *out* variable, we must protect access to these variables using a lock or a semaphore.

Let the name of that shared variable be ConsumedCnt
Assume that we have a SetConsumedCnt() and a GetConsumedCnt() functions.

Let sem be a mutex semaphore to protect access to the shared variables. sem must be initialized to 1.

```
while(1){
            sem_wait(&sem);
            if (GetConsumedCnt() == itemCnt)
            { sem_post(&sem); exit(0);}
            out = GetOut();
            while(GetIn() == out);
            val = ReadAtBufIndex(out);
            SetConsumedCnt(GetConsumedCnt() + 1);
            out = (out + 1) % bufSize;
            SetOut(out);
            sem_post(sem);
            ProcessItem(val);
}
```