Shared memory

This lab can be considered similar to the bounded buffer problem.

The input will be entered into the producer's terminal in the following format:

```
There are seven
Words in
this input
$
```

- Each line to be processed has words containing only English alphabet letters.
- Every line has a newline character at the end.
- Words in a line are separated by **one** space, except for the last word which has a newline after it.
- Every line has at least one word.
- The maximum possible length of a line is 50 bytes (Including the '\n' and the terminating '\0'.
- The last line always has the character `'$'` to indicate the end of the input.

There are three processes - one producer and two consumers (consumer1 and consumer2).
The code for all three should be implemented in separate files.
There will be four shared memories involved:

- Shared memory for the "buffer" of size 1024 where the producer stores the sentences. (**The producer can only store three sentences,** after that it has to wrap around.) i.e. the BUFFER_SIZE is 3.
- Shared memory for "in" variable (which is an integer). The producer uses this to track the index where the producer will store the next line.
- Shared memory for the "out1" variable (which is an integer). The consumer1 uses this to track the index from which the consumer1 will read the next line.
- Shared memory for the "out2" variable (which is an integer). The consumer2 uses this to track the index from which the consumer2 will read the next line.

The producer's task (producer.c):

1. The producer creates the above three shared memory blocks and attaches to them.
2. Read all the lines from the keyboard (till the user enters $) and store them in an array of strings.
3. Start storing these lines in the buffer, 3 at a time, and wait if the buffer is full and both the consumers haven't processed the line at the position that is to be overwritten.
4. After storing `'$'` the producer exits. So that, when the consumer sees `'$'` has been stored in the shared memory it knows it must exit.

The consumer1's task (consumer1.c). :

1. Attach to the necessary shared memory blocks.
2. Check for the buffer empty condition. If the buffer is empty, sleep for 1000 microseconds (Hint: Use usleep).
3. If the buffer is not empty, access the shared memory to count the number of words in the stored line. Words in a line are separated by one space. Print the number of words in that line (format given below)
4. If the line read is equal to **'$'**, print the total number of words in the input (format below) and then exit.

The consumer2's task (consumer2.c):

1. Attach to the necessary shared memory blocks.
2. Check for the buffer empty condition. If the buffer is empty, sleep for 1000 microseconds (Hint: Use usleep).
3. If the buffer is not empty, access the shared memory to count the number of capital letters in the stored line. Print the number of capital letters in that line (format given below).
4. If the line read is equal to **'$'**, print the total number of capital letters in the input (format below) and then exit.

Hints/Note:

- Use fgets to read a line from stdin with the '\n' at the end.
- Make sure the producer clears the 50 bytes shared memory corresponding to the index where a line is to be stored.
- Make sure all the processes detach themselves from the shared memory segment before exiting and the producer marks the shared memory segment for destruction.

./producer

```
THERE are Seven
LINES
in this input
$
Producer pid: 9115 exiting
```

./consumer1

```
Consumer pid: 9116, number of words in line 1: 3
Consumer pid: 9116, number of words in line 2: 1
Consumer pid: 9116, number of words in line 3: 3
Consumer pid: 9116, total number of words: 7
```

./consumer2

```
Consumer pid: 9117, number of capital letters in line 1: 6
Consumer pid: 9117, number of capital letters in line 2: 5
Consumer pid: 9117, number of capital letters in line 3: 0
Consumer pid: 9117, total number of words: 11
```