First the code.

Class - prod_cons.c

```c
#include <stdio.h>
#include <stdlib.h>
int mutex = 1;
int full = 0;
int empty = 10;
int x = 0;

// producer function
void producer() {
  --mutex;
  ++full;
  --empty;
  ++x;
  printf("\n Producer produces item %d ", x);
  ++mutex;
}

// consumer function

void consumer() {
  --mutex;
  --full;
  ++empty;
  printf("\n Consumer consumes item %d ", x);
  x--;
  ++mutex;
}

int main() {
```

```
    int n, i;
    printf(
        "\n Press 1 for Producer"
        "\n Press 2 for Consumer"
        "\n Press 3 for Exit");
    for (i = 1; i > 0; i++) {
      printf("\nEnter your choice");
      scanf("%d", &n);
      switch (n) {
        case 1:
          if ((mutex == 1) && (empty != 0)) {
            producer();
          } else {
            printf("Buffer is full");
          }
          break;
        case 2:
          if ((mutex == 1) && (full != 0)) {
            consumer();
          } else {
            printf("Buffer is empty");
          }
          break;
        case 3:
          exit(0);
          break;
      }
    }
  }
```

Mine - queue.c

```
#include <stdio.h>
#include <stdlib.h>
```

```c
// initialise mutex. I.e lock that is set before using shared memory.
// Lock that allows us to set  vars, then release after use.
// ============================== Init state
int mutex = 1;
int full = 0;    // i.e our slot is empty.
int empty = 10;  // -> we have 10 free slots
int x = 0;       // change the value of slot with this
// ==============================

// producer - no params. Like main (could pass args though)
void producer() {
  --mutex;  // lock. Ideally this starts our critical section
  // increase number of full. i.e we produced.
  ++full;
  // decrease empty
  --empty;
  // put values in em
  ++x;
  // print what the producer has.
  printf("\n Producer produces item %d",
         x);  // It's producing now, and in turn reducing our empty slots

  printf("\nProducer: x=%d, full=%d, empty=%d\n", x, full, empty);
  // release lock
  ++mutex;
}
void consumer() {
  --mutex;
  // decrease full
  --full;
  // increase empty
  ++empty;
  // print what the consumer has. Right before consumption. Cuz produce retains
  // state, consume destroys.
  printf("\nConsumer consumes item %d",
         x);  // It's consuming now, and in turn increasing our empty slots.
              // What it shows is saying: Producer will produce item4 (or x+1)
```

```
                // if consumer gobbledup 3 (or x)
    printf("\nConsumer: x=%d, full=%d, empty=%d\n", x, full, empty);
    // remove values
    --x;
    // release lock
    ++mutex;
  }

  // Note: our mutex is always the same when no critical section is occuring. So,
  // both can do what they want
  void main() {
    // look at two cases. Producer. Consumer. Both.
    // we don't know n times producers/processes will use.
    int choice, n;
    // this should be out of the for loop to avoid irregular loop - put it once so
    // we remember. WE could make it multiple for entropy
    printf("\n Enter how many times you want the process to run");
    scanf("%d", &n);
    for (size_t i = 1; i > 0;
         i++)  // infinite loop?? or one time loop?? - Infinite. Switch breaks. We
               // never leave the loop. Replace with while true.
    {
      // Menu. We might want to consume only or produce or both.
      printf(
          "\n Enter 1 for producer, 2 for consumer, 3 for both, and 4 to exit");
      scanf("%d", &choice);
      switch (choice) {
        case 1:
          if ((mutex == 1) && (empty != 0)) {
            // copilot init: (mutex == 1) && (full == 0) && (empty ==
            // 10). But last two are implied by empty!=0
            for (size_t i = 1; i <= n; i++) {
              producer();
            }

          } else
            printf("\n Buffer is full");  // it doesn't necessarily have to be
```

```
                                        // producer. Anyone else could occupy it
          break;
        case 2:
          if ((mutex == 1) && (full != 0)) {
            for (size_t i = 1; i <= n; i++) {
              consumer();
            }
          } else
            printf("\n Buffer is empty");
          break;
        case 3:
          if ((mutex == 1) && (full != 0) &&
              (empty != 0)) {  // ha! i'd like to see this execute! It does cuz they cover up for each other
            for (size_t i = 1; i <= n; i++) {
              producer();
              consumer();
            }
          } else
            printf("\n Buffer is either fu");
          break;
        default:
          printf("\n ERR: Invalid choice");
          exit(0);
          break;  // this is unreachable
      };
    }
  }
```

Trial one - running both works!

Err with this: Consumer lags behind producer, see trial2 Used sh for output for some highlighting

```
> Executing task in folder c: ./queue.out <


 Enter how many times you want the process to run2

 Enter 1 for producer, 2 for consumer, 3 for both, and 4 to exit1

 Producer produces item 1
Producer: x=1, full=1, empty=9

 Producer produces item 2
Producer: x=2, full=2, empty=8

 Enter 1 for producer, 2 for consumer, 3 for both, and 4 to exit1

 Producer produces item 3
Producer: x=3, full=3, empty=7

 Producer produces item 4
Producer: x=4, full=4, empty=6

 Enter 1 for producer, 2 for consumer, 3 for both, and 4 to exit2

Consumer consumes item 3
Consumer: x=3, full=3, empty=7

Consumer consumes item 2
Consumer: x=2, full=2, empty=8

 Enter 1 for producer, 2 for consumer, 3 for both, and 4 to exit1

 Producer produces item 3
Producer: x=3, full=3, empty=7

 Producer produces item 4
Producer: x=4, full=4, empty=6
```

```
 Enter 1 for producer, 2 for consumer, 3 for both, and 4 to exit3

 Producer produces item 5
Producer: x=5, full=5, empty=5

Consumer consumes item 4
Consumer: x=4, full=4, empty=6

 Producer produces item 5
Producer: x=5, full=5, empty=5

Consumer consumes item 4
Consumer: x=4, full=4, empty=6

 Enter 1 for producer, 2 for consumer, 3 for both, and 4 to exit4

 ERR: Invalid choice
```

From the outputs the following outliers should be noted:

- The last run where we do both consumer and producer is idempotent (doesn't change the overall state of the system) as expected. i.e our last consume is the same as the last produce before we started offf
- The other two runs of consumer/producer change the state of the system permanently.

Inputs: n=2, `choice={1,2,3,4}` In order.

Further confirmation can be done by testing another round of consume after the last choice in this trial.

Trial two

- Code from class runs as expected with a couple syntax errors (function names) and formatting (\E instead of \nE)

i.e here: `printf("\nEnter your choice");`

here: `produce(); -> producer();`

and here `consume(); -> consumer();`

- Found an error with my code i.e I was printing x after the mutations. The consumer should consume what the producer creates. Mine does things with +1 or -1.

Out from class

```
> Executing task in folder c: ./prod_cons.out <


 Press 1 for Producer
 Press 2 for Consumer
 Press 3 for Exit
Enter your choice1

 Producer produces item 1
Enter your choice1

 Producer produces item 2
Enter your choice2

 Consumer consumes item 2
Enter your choice2

 Consumer consumes item 1
Enter your choice1

 Producer produces item 1
Enter your choice1

 Producer produces item 2
Enter your choice2

 Consumer consumes item 2
```

```
Enter your choice3
```

All processes run once only. So, following from the earlier input names: Input: `n=1`, `choice={1,1,2,2,1,1,2,3}` . In order.

Refactoring a bit on my end: Same input, same out. (4 to exit on my end)

```
Terminal will be reused by tasks, press any key to close it.

> Executing task in folder c: ./queue.out <


 Enter how many times you want the process to run1

 Enter 1 for producer, 2 for consumer, 3 for both, and 4 to exit1

 Producer produces item 1
Producer: x=1, full=1, empty=9

 Enter 1 for producer, 2 for consumer, 3 for both, and 4 to exit1

 Producer produces item 2
Producer: x=2, full=2, empty=8

 Enter 1 for producer, 2 for consumer, 3 for both, and 4 to exit2

Consumer consumes item 2
Consumer: x=2, full=1, empty=9

 Enter 1 for producer, 2 for consumer, 3 for both, and 4 to exit2

Consumer consumes item 1
Consumer: x=1, full=0, empty=10
```

```
 Enter 1 for producer, 2 for consumer, 3 for both, and 4 to exit1

 Producer produces item 1
Producer: x=1, full=1, empty=9

 Enter 1 for producer, 2 for consumer, 3 for both, and 4 to exit1

 Producer produces item 2
Producer: x=2, full=2, empty=8

 Enter 1 for producer, 2 for consumer, 3 for both, and 4 to exit2

Consumer consumes item 2
Consumer: x=2, full=1, empty=9

 Enter 1 for producer, 2 for consumer, 3 for both, and 4 to exit4

 ERR: Invalid choice
Terminal will be reused by tasks, press any key to close it.
```