# Test Cases 377 - Lab 3

Gia Lee, Valerie So, William Robson

## Outline

**Assumptions**: This test document is to test the functioning of the producer/consumer threads and how they transfer values with the buffer. The rest of the code (like the input and argument passing) will not be tested as they were provided and assumed to be good.

**Testing the Producer Consumer Threads:**

For a test to be successful, all the values must be:

A) Accounted for, all data must make it to output.
B) Not doubled or duplicated in any way.
C) In order (relative to other values in the file it came from).

For the purposes of this test, each source file will have values labeled <host file num>000<value num>

Ex. 200014…. file 2 number 14.

Test file names follow the convention outlined in the lab instructions.

| Test # | Test Name | Input Files | Output Files |
|---|---|---|---|
| 1 | One to One | t10 | out10 |
| 2 | One to Many | t20 | out20 out21 out22 |
| 3 | Many to One | t30 t31 t32 | out30 |
| 4 | Many to Many | t40 t41 t42 | out40 out41 out42 |
| 5 | Not enough values (edge case) | t50 | out50 out51 out52 |
| 6 | No values (edge case) | t60 | out60 |

## One to One (Test 1 ./main 1 1 1)

One producer to one consumer

| Input: t10 | Output: out10 |
|---|---|

t10.dat

```
1   100001
2   100002
3   100003
4   100004
5   100005
6   100006
7   100007
8   100008
9   100009
10  100010
11  100011
12  100012
13  100013
14  100014
15  100015
```

out10.dat

```
1   100001
2   100002
3   100003
4   100004
5   100005
6   100006
7   100007
8   100008
9   100009
10  100010
11  100011
12  100012
13  100013
14  100014
15  100015
16
```

All values are accounted for, not doubled, and in order. Successful Test

## One to Many (Test 2 ./main 2 1 3)

One producer to three consumers

| Input: t20 | Output: out20 out21 out22 |
|---|---|

t20.dat

```
1   100001
2   100002
3   100003
4   100004
5   100005
6   100006
7   100007
8   100008
9   100009
10  100010
11  100011
12  100012
13  100013
14  100014
15  100015
```

out20.dat

```
1   100001
2   100002
3   100003
4   100004
5   100005
6   100009
7   100010
8   100011
9   100012
10  100013
11
```

out21.dat

```
1   100008
2   100014
3
```

out22.dat

```
1   100006
2   100007
3   100015
4
```

~/Downloads/elec37

All 15 values made it into the output.

There are no doubles across or in any file.

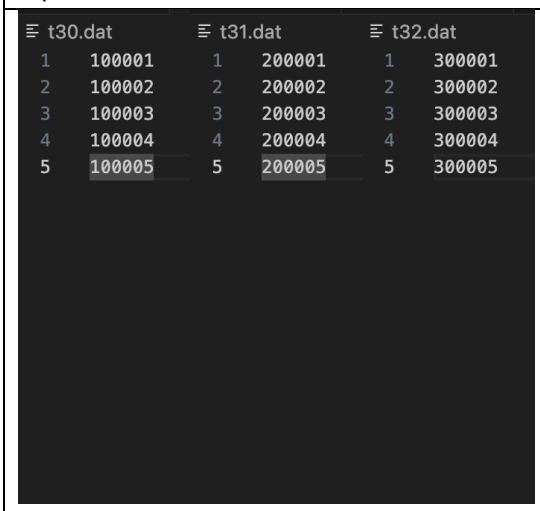
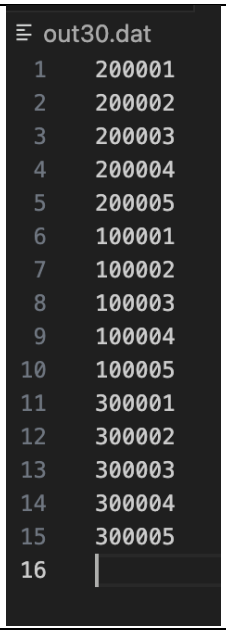
Within each output file, the values are still in numerical order as they were read from the input.

The test was successful.

--Interesting to note the first consumer did most of the work.

## Many to One (Test 3  ./main 3 3 1)

3 producers to one consumer

| Input: t30 t31 t32 | Output: out30 |
| --- | --- |
|  |  |





All 15 values made it into the output.

There are no doubles in the output file.

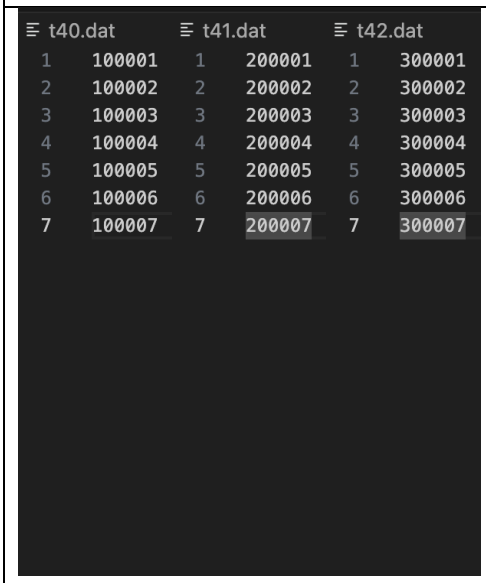
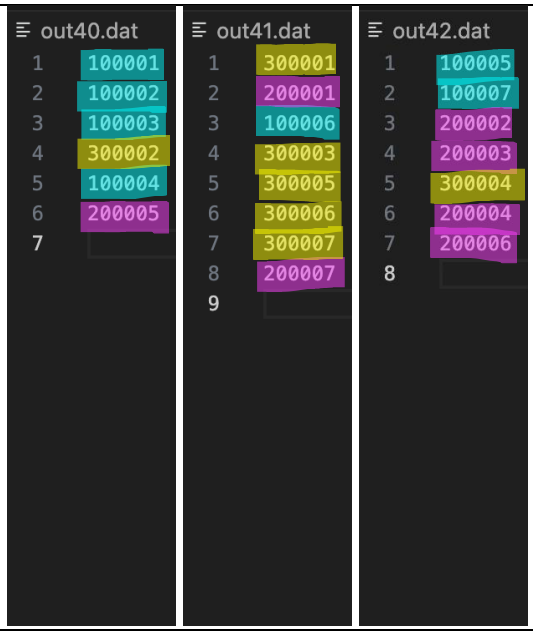
Values from each input file are in relative order they were read in with.

The test was successful.

--Interesting to note the producers read in their values completely before the next producer started.

## Many to Many (Test 4 ./main 4 3 3)

3 producers to 3 consumers

| Input: t40 t41 t42 | Output: out40 out41 out42 |
| --- | --- |
|  |  |





All 21 values made it into the output.

There are no doubles in or across the output files.

Values from each input file are in relative order they were read in with. (see highlighter)

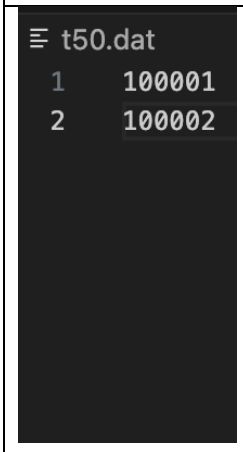
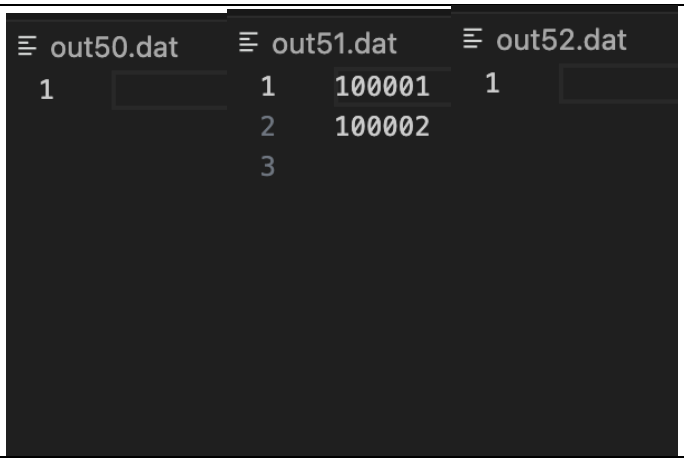
The test was successful.

--Here things are much more jumbled up, consumers and producers secured time seemingly randomly.

## Not enough values (Test 5 ./main 5 1 3)

Testing edge case where some consumers may not get any values.

1 producer (with only 2 values) to 3 consumers

| Input: t50 | Output: out50 out51 out52 |
|---|---|
|  |  |



The test succeeded, all values copied, no doubles, in proper order.

Importantly the consumer threads were able to start up, open an output file, and close, despite the fact that they never grabbed a value from the buffer. No errors, and consumers acted as intended.

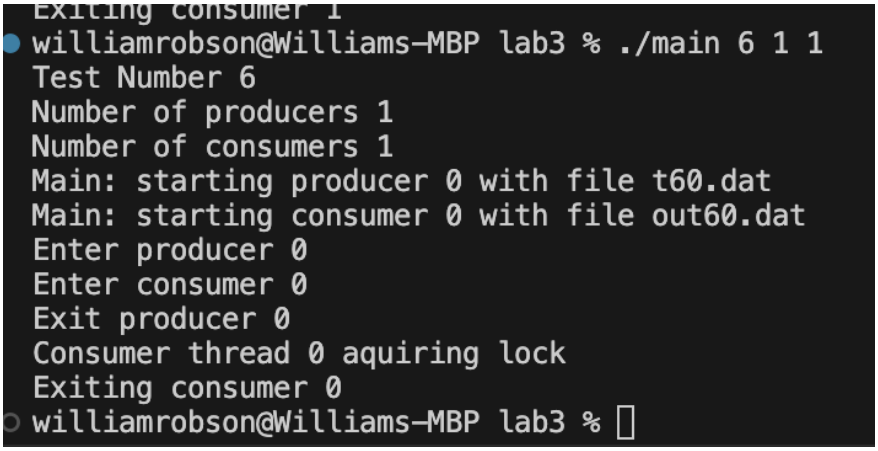
## No values (Test 6 ./main 6 1 1)

Edge case where producer has no values.

Producer and Consumer enter and exit as intended with no errors.

The producer handles itself well, seeing no more values (the file is empty) it exits as usual.

Out60 is generated with no values.




## Conclusion

The producer/consumer thread will properly pass values through the buffer as intended, moving all the values in the order they are read, completely, and without duplication or deletion.

No matter if it is one-to-one, one-to-many, many-to-one, or many-to-many, the values be output intact. Even if the producer or consumer thread is starved of values, it handles fine without error.

This code is well tested, and functions as intended.