The Producer-Consumer Problem

```
#define BUFFER_SIZE N

Typedef struct {

        ...

} item

Item buffer[BUFFER_SIZE]; % bounded buffer; circular array

int in = 0; % Variable used to insert a new item in the buffer

int out = 0; % Variable used to extract an item from the buffer

int counter = 0; %Shared variable (!!!)
```

The Producer process: to produce a new item and store it in the buffer

```
while (true){

    /*Produce a new item, and store it into variable "nextProduced"

    while (counter == BUFFER_SIZE)

        ; % do nothing (because the buffer is full)

    buffer[in] = nextProduced;

    in = (in + 1) mod BUFFER_SIZE;

    counter ++;

}
```

The Consumer Process:

```
item nextConsumed;

while (true){

    while (counter == 0)

        ; % do nothing (because the buffer is empty)

    nextConsumed = buffer[out];

    out = (out + 1) mod BUFFER_SIZE;

    counter--;

    % Consume the item in nextConsumed;

}
```

Does it works? Not really, because:

- count++ could be implemented as
  register1 = counter
  register1 = register1 + 1
  counter = register1
- count-- could be implemented as
  register2 = counter
  register2 = register2 - 1
  counter = register2
- Consider this execution interleaving with “counter = 5” initially:
  - S0: producer execute register1 = counter {register1 = 5}
    S1: producer execute register1 = register1 + 1   {register1 = 6}
    S2: consumer execute register2 = counter {register2 = 5}
    S3: consumer execute register2 = register2 - 1   {register2 = 4}
    S4: producer execute counter = register1   { counter = 6 }
    S5: consumer execute counter = register2   { counter = 4}