Algorithm 3.6: Second attempt	
boolean wantp ← false, wantq ← false	
p	**q**
loop forever	loop forever
p1: non-critical section	q1: non-critical section
p2: await wantq = false	q2: await wantp = false
p3: wantp ← true	q3: wantq ← true
p4: critical section	q4: critical section
p5: wantp ← false	q5: wantq ← false

```
bool wantp = false, wantq = false;
byte critical = 0;

active proctype p() {
   do
    ::
     !wantq;
      wantp = true;
      critical++;
      assert (critical == 1);
      critical--;
      wantp = false;
   od
}

active proctype q() {
   do
     ::
      !wantp;
      wantq = true;
      critical++;
      assert (critical == 1);
      critical--;
      wantq = false;
   od
}
```

Algorithm 3.8: Third attempt	
boolean wantp ← false, wantq ← false	
p	**q**
loop forever	loop forever
p1: non-critical section	q1: non-critical section
p2: wantp ← true	q2: wantq ← true
p3: await wantq = false	q3: await wantp = false
p4: critical section	q4: critical section
p5: wantp ← false	q5: wantq ← false

```
bool wantp = false, wantq = false;
byte critical = 0;

active proctype p() {
  do
    ::
    wantp = true;
    !wantq;
    critical++;
    assert (critical == 1);
    critical--;
    wantp = false;
  od
}

active proctype q() {
  do
    ::
    wantq = true;
    !wantp;
    critical++;
    assert (critical == 1);
    critical--;
    wantq = false;
  od
}
```

Algorithm 3.9: Fourth attempt	
boolean wantp ← false, wantq ← false	
p	**q**
loop forever	loop forever
p1: non-critical section	q1: non-critical section
p2: wantp ← true	q2: wantq ← true
p3: while wantq	q3: while wantp
p4: wantp ← false	q4: wantq ← false
p5: wantp ← true	q5: wantq ← true
p6: critical section	q6: critical section
p7: wantp ← false	q7: wantq ← false

```
bool wantp = false, wantq = false;
bool csp = false, csq = false;

ltl liveness { []<>csp && []<>csq }

active proctype p() {
   do
   :: wantp = true;
      do
      :: !wantq -> break;
      :: wantq ->
          wantp = false;
          wantp = true
      od;
      csp = true;
      assert (!(csp && csq));
      csp = false;
     wantp = false;
   od
}
active proctype q() {
   do
   :: wantq = true;
      do
      :: !wantp -> break;
      :: wantp ->
          wantq = false;
          wantq = true
      od;
      csq = true;
      assert (!(csp && csq));
      csq = false;
            wantq = false;
   od
}
```

Algorithm 3.13: Peterson's algorithm	
boolean wantp ← false, wantq ← false integer last ← 1	
p	**q**
loop forever	loop forever
p1: non-critical section	q1: non-critical section
p2: wantp ← true	q2: wantq ← true
p3: last ← 1	q3: last ← 2
p4: await wantq = false or last = 2	q4: await wantp = false or last = 1
p5: critical section	q5: critical section
p6: wantp ← false	q6: wantq ← false

Algorithm 5.1: Bakery algorithm (two processes)	
integer np ← 0, nq ← 0	
p	**q**
loop forever	loop forever
p1: non-critical section	q1: non-critical section
p2: np ← nq + 1	q2: nq ← np + 1
p3: await nq = 0 or np ≤ nq	q3: await np = 0 or nq < np
p4: critical section	q4: critical section
p5: np ← 0	q5: nq ← 0

```
#define NOSTARVE
#include "critical.h"
byte    np = 0, nq = 0;

ltl liveness { []<>nostarve }

active proctype p() {
          do
            :: np = nq + 1;
            ((nq == 0) || (np <= nq));
            critical_section('p');
            np = 0
        od
}
active proctype q() {
          do
            :: nq = np + 1;
            ((np == 0) || (nq < np));
            critical_section('q');
            nq = 0
          od
}
```