NAME:

1.) Betrachten Sie folgenden ‚Bounded Buffer' Lösung:

```
int buffer[max];


void *producer(void *arg) {
  for (i=0; i<loops; i++) {
    Pthread_mutex_lock(&mutex) ;            // p1
    while (count == max)                    // p2
       Pthread_cond_wait(&empty, &mutex);   // p3
    put(i);                                 // p4
    Pthread_cond_signal(&fill);             // p5
    Pthread_mutex_unlock(&mutex) ;          // p6
  }
}

void *consumer(void *arg) {
  while (1) {
      Pthread_mutex_lock(&mutex) ;          // c1
      while (count == 0)                    // c2
        Pthread_cond_wait(&fill, &mutex);   // c3
      int tmp = get();                      // c4
      Pthread_cond_signal(&empty) ;         // c5
      Pthread_mutex_unlock(&mutex);         // c6
      printf("%d\n", tmp);
      }
  }
```

Gehen Sie im weiteren von folgenden Annahmen aus:

- **ein Thread stoppt nur dann, wenn er entweder durch eine Condition Variable oder einen Lock (Mutex) blockiert wird.**
- **Mögliche Threadwechsel durch Interrupts, wie z.B. Ablauf der Zeitschreibe (Scheduling) tritt nicht auf!**

Ein möglicher Trace (Zeitablauf von links nach rechts) von den beiden Threads Producer ‚a' und Consumer ‚a' kann folgendermassen angegeben werden:

```
Thread Pa: 1,2,4,5,6,1,2,3
Thread Ca:                  1,2,4,5,6,1,2
```

Fragen [6P gesamt]: Geben Sie im weiteren den Trace für folgende Szenarien an:

Trace 1 (0,5P):

Ein Producer (*Pa*) und ein Consumer (*Ca*), `max=1`

Pa startet zuerst. Ablauf stoppt, wenn Consumer *Ca* ein Element ‚konsumiert' hat.

Thread Pa: 1,2,4,5,6,1,2,3

Thread Ca: 1,2,4

Trace 2 (0,5P):

Ein Producer (*Pa*) und ein Consumer (*Ca*), `max=3`

Pa startet zuerst. Ablauf stoppt, wenn Consumer *Ca* ein Element ‚konsumiert' hat.

Thread Pa: 1,2,4,5,6 1,2,4,5,6 1,2,4,5,6, 1,2,3

Thread Ca: 1,2,4

Trace 3 (1P):

Ein Producer (*Pa*) und ein Consumer (*Ca*), `max=1`

Ca startet zuerst. Ablauf stoppt, wenn Consumer *Ca* ein Element ‚konsumiert' hat.

Thread Pa: 1,2,4,5,6,1,2,3

Thread Ca: 1,2,3 2[1],4,

Trace 4 (1P):

Ein Producer (*Pa*) und zwei Consumer (*Ca,Cb*), `max=1`

Ca startet zuerst, dann Cb und dann Pa. Ablauf stoppt, wenn Producer P*a* ein Element ‚produziert' hat.

Thread Pa: 1,2,4

Thread Ca: 1,2,3

Thread Cb: 1,2,3

---

[1] RE-CHECK !!

Trace 5 (1,5P):

Die while Loops im Code werden für diese Trace Aufgabe mit if Statements ersetzt.
Zeigen Sie mit einem Producer (*Pa*) und zwei Consumer (*Ca,Cb*) und einem `max=` Wert Ihrer Wahl, dass dies zu einem Problem führt.

max = 1

Thread Pa: **1,2,4,5**[2]**,6,1,2,3**

Thread Ca: 1,2,3(wait) **4**[3]

Thread Cb: **1,2,4**[4]**,5,6,1,2,3**

Trace 6 (1,5P):

Nun werden wieder while Loops im Code verwendet, aber nur eine(!) Condition Variable statt der bisherigen 2 CVs.
Zeigen Sie mit einem Producer (*Pa*) und zwei Consumer (*Ca,Cb*) und einem `max=` Wert Ihrer Wahl, dass dies zu einem Problem führen kann (Erklärung!)

max = 1

Pa: **1,2,4,5,6,1,2,3(wait)**

Ca: **1,2,3(wait)** **2,4,5(signal),6,1,2,3(wait)**

Cb: 1,2,3(wait) **2,3(wait)**

Erklärung: signal() aber wer wird zuerst ausgeführt?, Producer sollte, aber Consumer wird zuerst ausgeführt. Alle 3 Threads am Ende im 'Wartezustand' … rien ne va plus.

---

[2] Produces Item

[3] Versucht Item zu Lesen (hat sich ja vorher auf leeren Puffer schlafen gelegt)

[4] Consumes Item