COMP3520 Assignment 1 – Discussion Document

Question 1

Part A

Pseudocode — Teacher

```
lock(mu)
print "waiting for all students"
while arrived_cnt < N:
    wait(all_arrived_cv)                      // students -> teacher

print "assign groups"
compute G[g].size for g=0..M-1
permute students randomly
for each group g in order:
  for each student sid in g:
    group_of_student[sid] = g
    print assignment
    next_sid = sid
    broadcast(assigned_one_cv)                // teacher -> students
    while ack_sid != sid:
        wait(ack_cv)                          // student sid -> teacher

rooms_phase_started = true
broadcast(rooms_phase_cv)                     // teacher -> tutors
print "waiting for rooms"

while next_gid < M:
  while empty_rooms queue is empty:
      wait(empty_room_cv)                     // tutors -> teacher
  rid = er_pop()
  gid = next_gid++
  G[gid].called = true
  G[gid].room_assigned = rid
  R[rid].current_group = gid
  print "room available for gid"
  broadcast(G[gid].called_cv)                 // teacher -> students in gid
  signal(R[rid].assigned_cv)                  // teacher -> that room's tutor

for each g:
  while G[g].left_cnt < G[g].size:
      wait(G[g].all_left_cv)                  // last student -> teacher

all_groups_done = true
for each room r: broadcast(R[r].assigned_cv)  // let idle tutors exit
print "all left"
unlock(mu)
```

Pseudocode — Tutor (one per room rid)

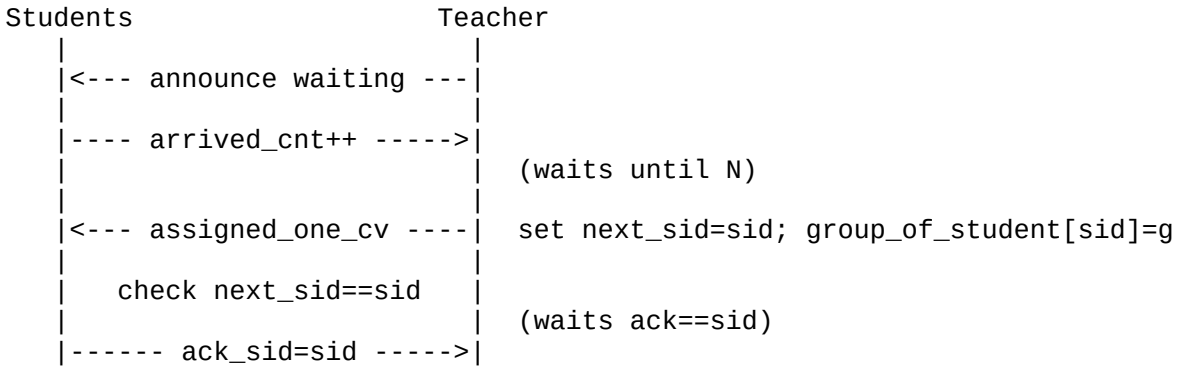
```
lock(mu)
while !rooms_phase_started:
    wait(rooms_phase_cv)                        // teacher -> tutors

R[rid].current_group = -1
print "room ready"
er_push(rid); signal(empty_room_cv)             // tutor -> teacher

while !all_groups_done:
  while R[rid].current_group == -1 && !all_groups_done:
      wait(R[rid].assigned_cv)                  // teacher -> this tutor
  if all_groups_done: break

  gid = R[rid].current_group
  while G[gid].entered_cnt < G[gid].size:
      wait(G[gid].all_entered_cv)               // last entering student -> tutor
  print "all in"

  unlock(mu)
  sleep(duration in 1..Tlim)                    // exercise
  lock(mu)

  print "done"
  G[gid].completed = true
  broadcast(G[gid].completed_cv)                // tutor -> students

  while G[gid].left_cnt < G[gid].size:
      wait(G[gid].all_left_cv)                  // last leaving student -> tutor

  G[gid].called = false
  R[rid].current_group = -1
  print "room ready"
  er_push(rid); signal(empty_room_cv)           // tutor -> teacher

  if (++groups_finished == M):
      all_groups_done = true
      for all rooms: broadcast(R[...].assigned_cv)

print "tutor bye"
unlock(mu)
```

Pseudocode — Student (one per sid)

```
lock(mu)
print "arrived"
if (++arrived_cnt == N):
    broadcast(all_arrived_cv)                   // students -> teacher
while group_of_student[sid] == -1 || next_sid != sid:
    wait(assigned_one_cv)                       // teacher -> students

gid = group_of_student[sid]
print "know my group"
ack_sid = sid; signal(ack_cv)                   // student -> teacher
while !G[gid].called:
    wait(G[gid].called_cv)                      // teacher -> group
rid = G[gid].room_assigned
print "entering"

if (++G[gid].entered_cnt == G[gid].size):
    signal(G[gid].all_entered_cv)               // last enterer -> tutor

while !G[gid].completed:
    wait(G[gid].completed_cv)                   // tutor -> students
print "bye"

if (++G[gid].left_cnt == G[gid].size):
    broadcast(G[gid].all_left_cv)               // last leaver -> tutor & teacher
unlock(mu)
return
```

Diagrams (text)

Arrival & assignment handshake



Part B

Mutex

- `pthread_mutex_t mu` - protects all shared state below and is held for every condvar wait/signal.

Condition variables

- `all_arrived_cv` — arrival barrier: students broadcast when `arrived_cnt==N`; teacher waits
- `assigned_one_cv` / `ack_cv` — *per-student handshake* so printed assignment ⇄ student acknowledgement are ordered and not lost.
- `rooms_phase_cv` — gate that holds tutors until teacher finishes assignment.
- `empty_room_cv` — room-availability queue notification: tutors signal when a room becomes free; teacher waits/pops.
- Per-room: `R[r].assigned_cv` — wake exactly the tutor of the room the teacher just assigned.
- Per-group:
  - `G[g].called_cv` — teacher calls a group to its room.
  - `G[g].all_entered_cv` — last entering student wakes tutor to start.
  - `G[g].completed_cv` — tutor announces completion; students may “Bye”.
  - `G[g].all_left_cv` — last leaving student wakes both tutor & teacher.

Core globals / shared fields

- **Counts/gates:** `arrived_cnt`, `next_gid`, `groups_finished`, `all_groups_done`, `rooms_phase_started`.
- **Assignment state:** `group_of_student[]`, `next_sid`, `ack_sid`.
- **Room state:** `R[r].current_group`.
- **Group state:** `G[g].size`, `room_assigned`, `entered_cnt`, `left_cnt`, `called`, `completed`.
- **Empty-room queue:** `empty_rooms[]`, `er_head`, `er_tail`, guarded by `mu` and signaled via `empty_room_cv`.

Each variable/condvar is used to enforce one step in the overall order:

1) **All arrive** → 2) **Per-student assignment (ordered prints)** → 3) **Tutors advertise rooms** → 4) **Teacher pairs next group with a free room** → 5) **All group members enter** → 6) **Tutor signals completion** → 7) **All leave** → 8) **Tutor re-queues room** → 9) **Repeat until all groups finish** → 10) **Clean shutdown**.

Question 2

Part A

Key invariants (proved by reasoning & validated in runs):

- *Per student:* `arrived` → `know my group` → `entering` → `bye`.
- *Per group:*
  - No entry before `G[gid].called == true` and a room is assigned.
  - Exactly one `"Tutor all_in"` after `entered_cnt == size`.
  - `"Tutor done"` happens before any `"Bye"`.
  - `left_cnt` never exceeds `entered_cnt` and ends at `size`.
- *Per room:* `R[rid].current_group` is either `-1` or a single `gid` (no overlap).
- *Global:* `groups_finished` reaches `M`; then all threads exit.

Test matrix (covering edge cases & interleavings):

Scenario	Purpose	Example input
Serial baseline	End-to-end flow	3 1 1 1
Single-room queueing	Check round-robin groups	6 2 1 2
Parallel rooms	Cross-room interleavings	9 3 2 2
Many tiny groups	Stress per-group condvars	10 10 3 1
More rooms than groups	Tutors idle cleanly	8 2 4 1
Stress	Look for hangs/missed signals	200 40 10 1
Tlim boundary	Ensure duration ≥ 1	(we now require Tlim>0)

Why the outputs are “correct” despite nondeterminism:

We don’t enforce a total order of lines—only the *necessary* happens-before relations with condition variables. The above invariants precisely define those relations and are guaranteed by (a) holding `mu` around every shared update and (b) the `while`-loops on all waits (protect against spurious wakeups and missed signals).

Part B

Bugs / limitations & remedies

the tutor’s duration can be simplified to:

```
int duration = (rand() % Tlim) + 1;  // 1..Tlim
```

The empty-room queue is bounded (total pushes ≤ `K + M`) and we allocate `K + M + 8.` That’s safe for this design; if the logic evolves, convert to a circular buffer.

Could destroy condvars/mutex and `free()` arrays at the end (not required, but neat).

Manual: compile & run

Requirements: POSIX pthreads (Linux/macOS). Use `gcc`

```
gcc main.c -o lab_sim -pthread
./lab_sim
# then type: N M K Tlim   (all > 0)
# example: 12 3 2 3
```