Zadanie 3.

Algorytmy **wzajemnego wykluczania** (w skrócie często nazywane mutex, z ang. mutual exclusion) są używane w przetwarzaniu współbieżnym w celu uniknięcia równoczesnego użycia wspólnego zasobu (np. zmiennej globalnej) przez różne wątki/procesy w częściach kodu zwanych sekcjami krytycznymi. Sekcja krytyczna jest fragmentem kodu, w którym wątki (lub procesy) odwołują się do wspólnego zasobu.

Rozważmy sytuację:

1. Oba wątki są przed wykonaniem instrukcji z linii 6.
2. Wątek z `id=1` idzie dalej. Wchodzi do `while`, bo spełnia warunek `turn != id` (`turn` jest równe 0). Kolejnego `while` omija, bo `blocked[0] = false`. Zatrzymujemy się przed instrukcją z 10. linii.
3. Wątek 0 zaczyna działać. Nie wchodzi do `while`, bo `turn = id = 0`. Zatem teraz wątek 0 jest przed wykonaniem sekcji krytycznej.
4. Wątek 1 znów zaczyna działać. Wykonuje instrukcję `turn = id`, przez co wychodzi również z `while` z linii 7 i zaczyna wykonywać sekcje krytyczną.
5. Oba wątki weszły do sekcji krytycznej- mamy race condition.

Zadanie 4.

Proof by contradiction: assume both P0 and P1 are in their CS (critical section)

- then `flag[0] = flag[1] = true`
- the test for entry cannot have been true for both processes at the same time (because turn favors one); therefore one process must have entered its CS first (without loss of generality, say P0)
- but this means that P1 could not have found `turn = 1` and therefore could not have entered its CS (i.e. contradiction)