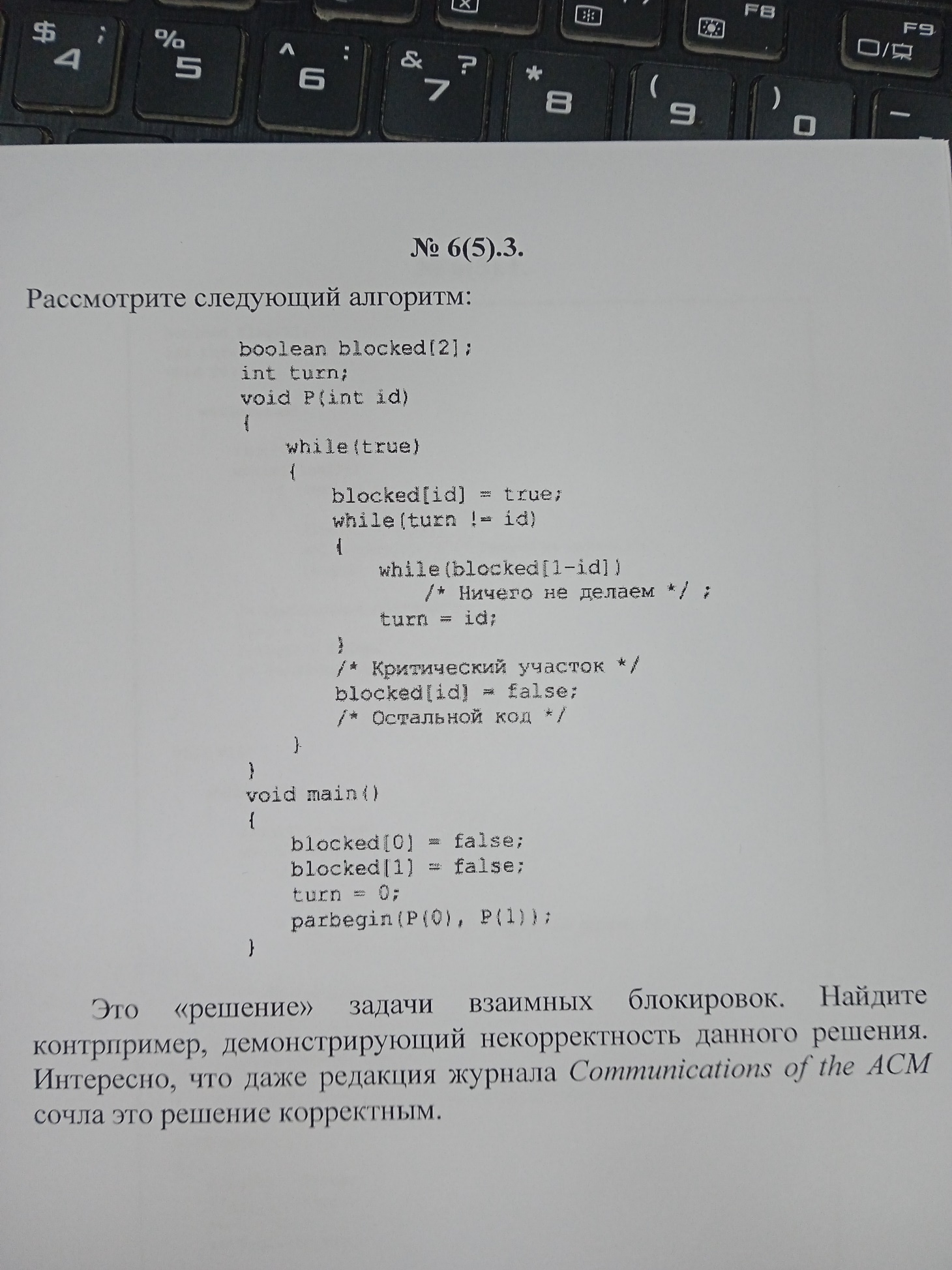
**Отчет по лабораторной работе №6**

**Вариант №6(5).3.**

**Аладко Анастасии Дмитриевны**

**12, МСС**

**Задача**

******

**Выполнение**

Это решение похоже на алгоритм Деккера, но отличается от него и содержит ошибки.

Рассмотрим возможную ситуацию:

turn = 0, blocked[0] = true, blocked[1] = false и пускай 0-й процесс выполнит критическую секцию и одной из возможных ситуаций будет то, что он заново зайдёт в цикл и установит blocked[0] = true сразу после того, как 1й процесс выйдет из ожидания blocked[0] = true, потом 0-й процесс войдёт в критическую секцию, 1й процесс изменит очередь и тоже войдёт в критическую секцию. Таким образом, если скорости выполнения процессов отличаются, то **взаимоисключение не гарантируется**.

Рассмотрим возможную ситуацию:

turn = 0, blocked[0] = true, blocked[1] = true, и 1й процесс ждёт, пока blocked[0] будет false. Возможна ситуация, когда 0-й процесс выполняется во много раз быстрее 1го, тогда он будет постоянно перехватывать инициативу и устанавливать blocked[0] = true до того, как 1-й успеет выйти из ожидания. Т.е. здесь возможно **голодание** одного из процессов, если 2й выполняется гораздо быстрее.

**Вывод**

Хоть этот код и похож на алгоритм Деккера, но он содержит кучу проблем, таких как нарушение **взаимоисключения**, **голодание** процесса. Хоть в условии и написано, что это «решение» задачи взаимных блокировок, то в таком решении нет никакого смысла уже только потому, что нарушается **взаимоисключение** и оба процесса могут войти в критическую секцию одновременно, а из-за некорректного обращения к критическим ресурсам могут быть большие проблемы.