# **Лабораторная работа № 08**

# **Тема: Асинхронное программирование**

**Цель:**

* Знакомство с асинхронным программированием;
* Получение практических навыков в параллельной обработке данных;
* Получение практических навыков в синхронизации потоков;

## **Порядок выполнения работы**

1. Ознакомиться с теоретическим материалом.

2. Получить у преподавателя вариант задания.

3. Реализовать задание своего варианта в соответствии с поставленными требованиями.

4. Подготовить тестовые наборы данных.

5. Создать репозиторий на GitHub.

6. Отправить файлы лабораторной работы в репозиторий.

7. Отчитаться по выполненной работе путём демонстрации работающей программы на тестовых наборах данных (как подготовленных самостоятельно, так и предложенных преподавателем) и ответов на вопросы преподавателя (как из числа контрольных, так и по реализации программы).

## **Требования к программе**

Разработать программу на языке C++ согласно варианту задания. Программа на C++ должна собираться с помощью системы сборки CMake. Программа должна получать данные из стандартного ввода и выводить данные в стандартный вывод.

Необходимо настроить сборку лабораторной работы с помощью CMake. Собранная программа должна называться **oop\_exercise\_08** (в случае использования Windows **oop\_exercise\_08.exe)**

Необходимо зарегистрироваться на GitHub (если студент уже имеет регистрацию на GitHub то можно использовать ее) и создать репозитарий для задания лабораторной работы.

Преподавателю необходимо предъявить ссылку на публичный репозиторий на Github. Имя репозитория должно быть<https://github.com/login/oop_exercise_08>

Где login – логин, выбранный студентом для своего репозитория на Github.

Репозиторий должен содержать файлы:

· main.cpp //файл с заданием работы

· CMakeLists.txt // файл с конфигураций CMake

· report.doc // отчет о лабораторной работе

Создать приложение, которое будет считывать из стандартного ввода данные фигур, согласно варианту задания, выводить их характеристики на экран и записывать в файл. Фигуры могут задаваться как своими вершинами, так и другими характеристиками (например, координата центра, количество точек и радиус).

Программа должна:

1. Осуществлять ввод из стандартного ввода данных фигур, согласно варианту задания;

2. Программа должна создавать классы, соответствующие введенным данных фигур;

3. Программа должна содержать внутренний буфер, в который помещаются фигуры. Для создания буфера допускается использовать стандартные контейнеры STL. Размер буфера задается параметром командной строки. Например, для буфера размером 10 фигур: **oop\_exercise\_08 10**

4. При накоплении буфера они должны запускаться на асинхронную обработку, после чего буфер должен очищаться;

5. Обработка должна производиться в отдельном потоке;

6. Реализовать два обработчика, которые должны обрабатывать данные буфера:

a. Вывод информации о фигурах в буфере на экран;

b. Вывод информации о фигурах в буфере в файл. Для каждого буфера должен создаваться файл с уникальным именем.

7. Оба обработчика должны обрабатывать каждый введенный буфер. Т.е. после каждого заполнения буфера его содержимое должно выводиться как на экран, так и в файл.

8. Обработчики должны быть реализованы в виде лямбда-функций и должны хранится в специальном массиве обработчиков. Откуда и должны последовательно вызываться в потоке – обработчике.

9. В программе должно быть ровно два потока (thread). Один основной (main) и второй для обработчиков;

10. В программе должен явно прослеживаться шаблон Publish-Subscribe. Каждый обработчик должен быть реализован как отдельный подписчик.

11. Реализовать в основном потоке (main) ожидание обработки буфера в потоке-обработчике. Т.е. после отправки буфера на обработку основной поток должен ждать, пока поток обработчик выведет данные на экран и запишет в файл.

**Варианты заданий (выпуклые равносторонние фигуры вращения):**

|  |  |  |  |
| --- | --- | --- | --- |
| **Вариант** | **Фигура №1** | **Фигура №2** | **Фигура №3** |
| 1. | Треугольник | Квадрат | Прямоугольник |
| 2. | Квадрат | Прямоугольник | Трапеция |
| 3. | Прямоугольник | Трапеция | Ромб |
| 4. | Трапеция | Ромб | 5-угольник |
| 5. | Ромб | 5-угольник | 6-угольник |
| 6. | 5-угольник | 6-угольник | 8-угольник |
| 7. | 6-угольник | 8-угольник | Треугольник |
| 8. | 8-угольник | Треугольник | Квадрат |
| 9. | Треугольник | Квадрат | Прямоугольник |
| 10. | Квадрат | Прямоугольник | Трапеция |
| 11. | Прямоугольник | Трапеция | Ромб |
| 12. | Трапеция | Ромб | 5-угольник |
| 13. | Ромб | 5-угольник | 6-угольник |
| 14. | 5-угольник | 6-угольник | 8-угольник |
| 15. | 6-угольник | 8-угольник | Треугольник |
| 16. | 8-угольник | Треугольник | Квадрат |
| 17. | Треугольник | Квадрат | Прямоугольник |
| 18. | Квадрат | Прямоугольник | Трапеция |
| 19. | Прямоугольник | Трапеция | Ромб |
| 20. | Трапеция | Ромб | 5-угольник |
| 21. | Ромб | 5-угольник | 6-угольник |
| 22. | 5-угольник | 6-угольник | 8-угольник |
| 23. | 6-угольник | 8-угольник | Треугольник |
| 24. | 8-угольник | Треугольник | Квадрат |
| 25. | Треугольник | Квадрат | Прямоугольник |
| 26. | Квадрат | Прямоугольник | Трапеция |
| 27. | Прямоугольник | Трапеция | Ромб |
| 28. | Трапеция | Ромб | 5-угольник |
| 29. | Ромб | 5-угольник | 6-угольник |
| 30. | 5-угольник | 6-угольник | 8-угольник |
| 31. | 6-угольник | 8-угольник | Треугольник |
| 32. | 8-угольник | Треугольник | Квадрат |
| 33. | Треугольник | Квадрат | Прямоугольник |
| 34. | Квадрат | Прямоугольник | Трапеция |
| 35. | Прямоугольник | Трапеция | Ромб |
| 36. | Трапеция | Ромб | 5-угольник |

Отчет

1. Код программы на языке C++.

2. Ссылка на репозиторий на GitHub.

3. Набор testcases.

4. Результаты выполнения тестов.

5. Объяснение результатов работы программы.