To execute the code for the Operational Research Projects in Xcode, follow these steps:
Prerequisites
- Xcode installed on your macOS device.
Steps to Execute
2. **Open the Xcode Project**:
   - Launch Xcode.
   - Open the cloned project by selecting `File` -> `Open` and navigating to the project folder.
3. **Build and Run**:
   - Select a target (e.g., a simulator or a physical device) from the target device menu in the Xcode toolbar.
   - Press `Cmd + R` or select `Product` -> `Run` to build and run the project.
4. **Explore the Simulations**:
   - Use the interface to input parameters for the simulations.
   - Click the appropriate buttons (e.g., "Show Results") to trigger the simulations and display the results.
5. **View Results**:
   - Explore the results displayed in the application interface, which may include tables, charts, or other visualizations depending on the simulation.
6. **Interact with the Application**:
   - Experiment with different input parameters to see how the simulations behave.
   - Use any provided functionalities (e.g., fitness tests, data analysis) to further analyze the results.
7. **Review and Modify Code** (Optional):
   - Explore the codebase to understand how the simulations are implemented.
   - Modify the code as needed to customize simulations or add new features.
8. **Build and Run Again** (Optional):
   - After making changes to the code, rebuild and run the project to see the effects of your modifications.
9. **Save and Share Results**:
   - If desired, save the results of the simulations or share them with others for review and discussion.
10. **Close the Project**:
    - When finished, close the project in Xcode by selecting `File` -> `Close Project`.

By following these steps, you can execute and explore the Operational Research Projects in Xcode, gaining insights into queuing models, random number generation, and other simulations relevant to operational research.

Here's a documentation outline for your Xcode project:
Operational Research Projects Documentation
5th Semester
Queuing Model
- **Model**
  - `Result()`: Represents the result of a queuing simulation.
  - `FitTest`: Performs a fitness test on the queuing model.
- **View**
  - `QueuingView()`: Manages the interface for the queuing model simulation inputs and results display.
- **ViewModel**
  - `QueuingViewModel()`: Handles the logic for the queuing model simulation.
Random Number Generator
- **View**
  - `RandomNumberView()`: Manages the interface for generating random numbers.
- **ViewModel**
  - `RandomViewModel()`: Handles the logic for generating random numbers.
6th Semester
MM1Priority
- **Model**
  - `Customer`: Represents a customer in the M/M/1 priority queuing model.
  - `GrantChartData`: Stores data for generating Grantt Chart.
- **View**
  - `MM1PriorityView()`: Manages the interface for the M/M/1 priority queuing model simulation inputs and results display.

- **ViewModel**
  - `MM1PriorityViewModel()`: Handles the logic for the M/M/1 priority queuing model simulation.
LCG (Linear Congruential Generator)
- **Model**
  - `LCGRow`: Represents a row in the LCG table.
  - `LCGInput`: Represents input parameters for the LCG calculation.
- **View**
  - `LCGView()`: Manages the interface for the LCG simulation inputs and results display.
- **ViewModel**
  - `LCGViewModel()`: Handles the logic for the LCG simulation.
This structure outlines the key components of your project, organizing them by semester and functionality.