МИНИСТЕРСТВО ОБРАЗОВАНИЯ И НАУКИ РЕСПУБЛИКИ БЕЛАРУСЬ

БЕЛОРУССКИЙ НАЦИОНАЛЬНЫЙ ТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ

## Факультет информационных технологий и робототехники

Кафедра программного обеспечения информационных систем

и технологий

**Отчет**

**по лабораторной работе № 7**

по дисциплине: ”Системное программирование”

на тему: ***”*** **Компиляция и отладка простейшего приложения**

**в Linux. Автоматическая сборка многофайловых**

**проектов*”***

Выполнил**:** студент группы *10702121* Нестюк Е. А.

Огризко А. В.

Принял**:** пр. Давыденко Н. В.

Минск 2023

# Лабораторная работа №7.

**Цель работы:** Изучить встроенный инструментарий для разработки приложений под семейство ОС Linux и фундаментальные основы системного программирования с использованием компиляторов gcc/g++, отладчика gdb и других для проектирования, компиляции, отладки и запуска приложений на языке программирования С/C++. Научиться грамотно проектировать и разрабатывать многофайловые проекты на языке программирования С/C++ с использованием архитектурного шаблона проектирования и разработки гибких и масштабируемых приложений Model-View-Controller (MVC). Научиться эффективно использовать специальные средства для автоматизации процесса компиляции, сборки и запуска многофайловых проектов.

## Задание 1

Используя материалы (исходные файлы) упражнения №1 проведите компиляцию однофайлового проекта с прохождением всех стадий компиляции. Для ускорения работы примените скрипты bash.

Исходные файлы программ обязательно должны содержать коментарии. Сборка проекта должна содержать файлы с результатами препроцессинга. Исследуйте фалы препроцессора, найдите в них код своей программы.

Определите размеры исходных, препроцессорных, ассемблерных, объектных и исполняемых файлов. С помощью соответствующий консольных команд определите форматы этих файлов. Результаты подтвердите скриншотами.

**Упражнение 1**

Изучите и выполните требования, представленные в лабораторной работе №2 «Компиляция и отладка простейшего приложения в Linux». Разработайте исходные коды приложений в соответствии с заданиями. Компиляцию приложений проведите с помощью консольных команд.

Попробуйте написать похожий скрипт:

#!/bin/sh

gcc -E test.c -o test.i

gcc -S test.i -o test.s

gcc -c test.s

gcc -o testprogram test.o

При разработке приложений не забудьте отразить информацию о разработчиках.

Файлы с содержанием лабораторной работы предоставлены.

**Требования**

1) Разработать модульное консольное приложение на С/С++ согласно варианту задания. Для компиляции, компоновки и выполнения программы использовать стандартный компилятор Linux gcc/g++, а для отладки - gdb.

2) Размерность массива задаётся пользователем на стадии выполнения программы. Для повышения производительности программы необходимо использовать динамическое выделение памяти.

3) Во время работы программы должны выводить на экран исходные и конечные данные.

4) При выполнении задания запрещается использовать интегрированные средства разработки (Integrated Development Environment, IDE). Рекомендуется задействовать любой текстовый редактор (к примеру, gedit) и набор компиляторов GNU Compiler Collection (GCC), в частности, компиляторы языков программирования С/C++ gcc/g++.

5) При разработке программ необходимо придерживаться соглашения по написанию кода на C/C++ (Code-Convention).

**Основное задание**

Ввести массив вещественных чисел размером N. Найти его наибольший и наименьший элементы и поменять их местами. Найти сумму и произведение всех элементов массива.

**Индивидуальное задание**

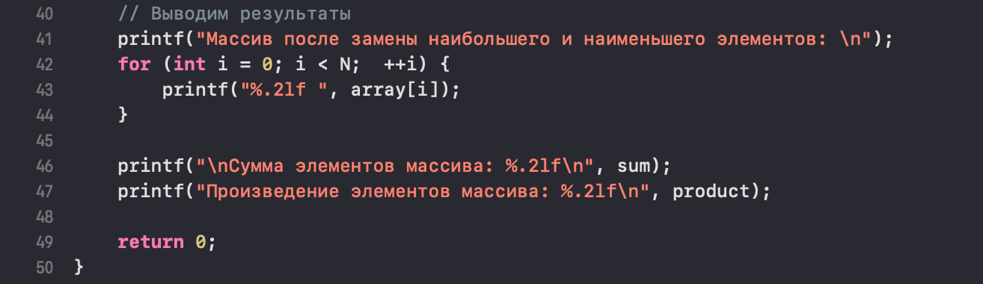
8) В одномерном массиве, состоящем из к целых элементов, вычислить: количество положительных элементов массива и сумму элементов массива, расположенных после последнего элемента, равного нулю.

**Решение**

Разработаем два приложения описанных в лабораторной работе в лабораторной работе №2 «Компиляция и отладка простейшего приложения в Linux»

*Основное задание*





*Индивидуальное задание*



Проведем их компиляцию с помощью gcc

*Основное задание*

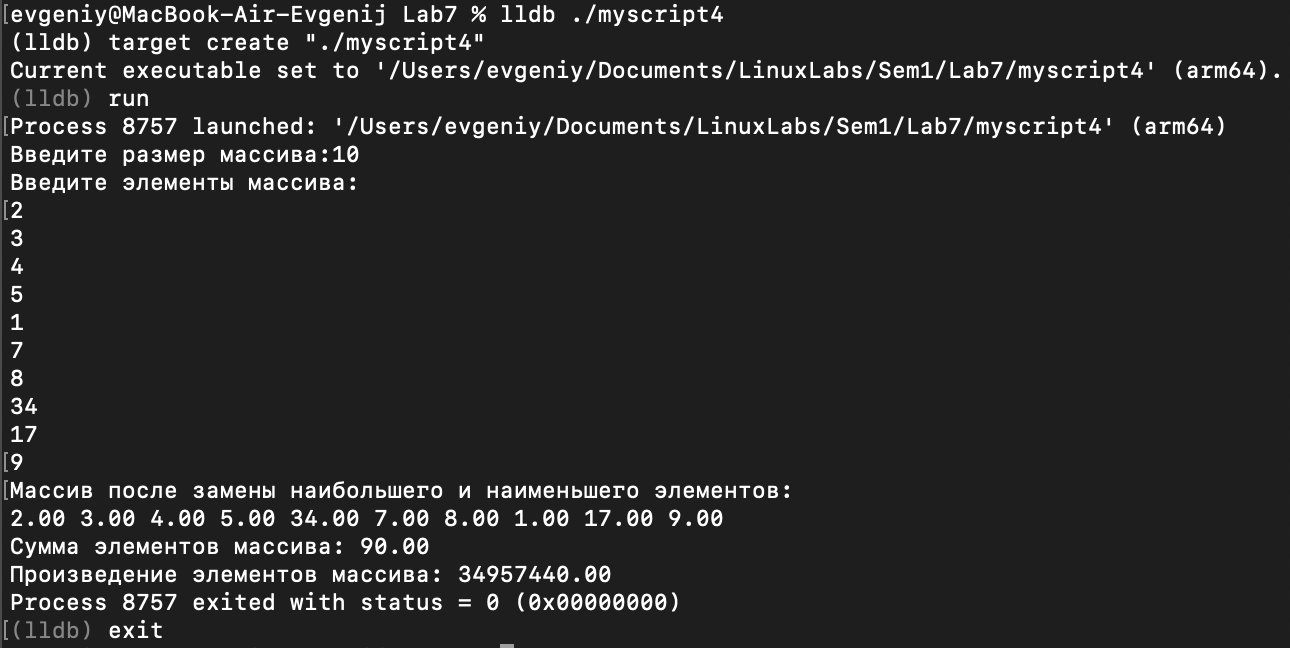


*Индивидуальное задание*

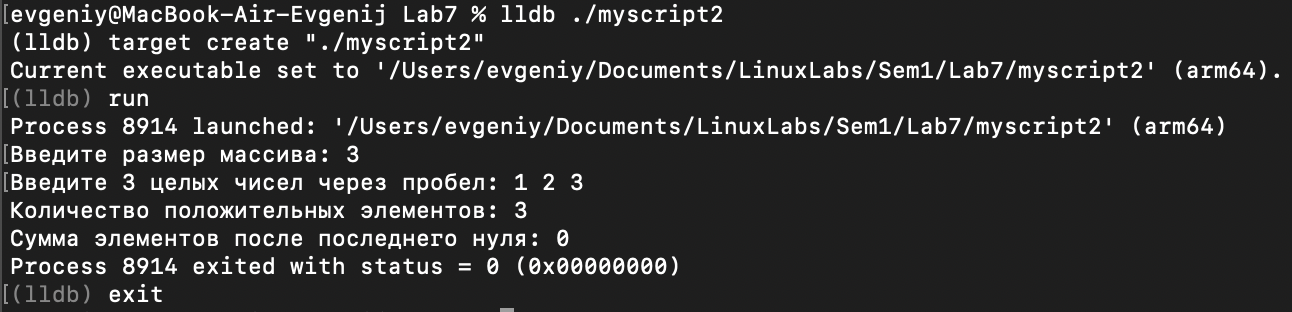


Проведем их отладку с помощью lldb

*Основное задание*

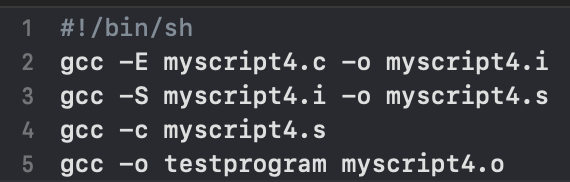


*Индивидуальное задание*



Проведем компиляцию однофайлового проекта с прохождением всех стадий компиляции. Для этого напишем скрипты.

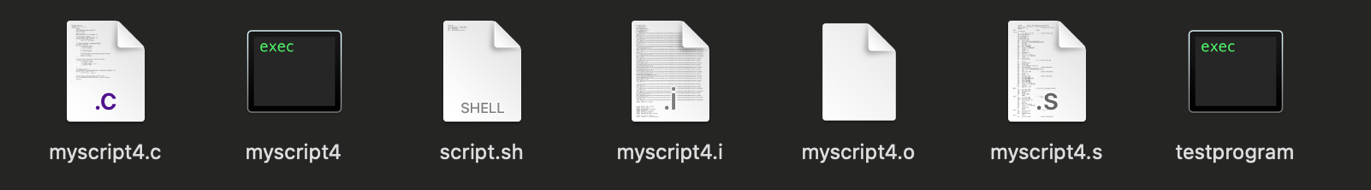
*Основное задание*



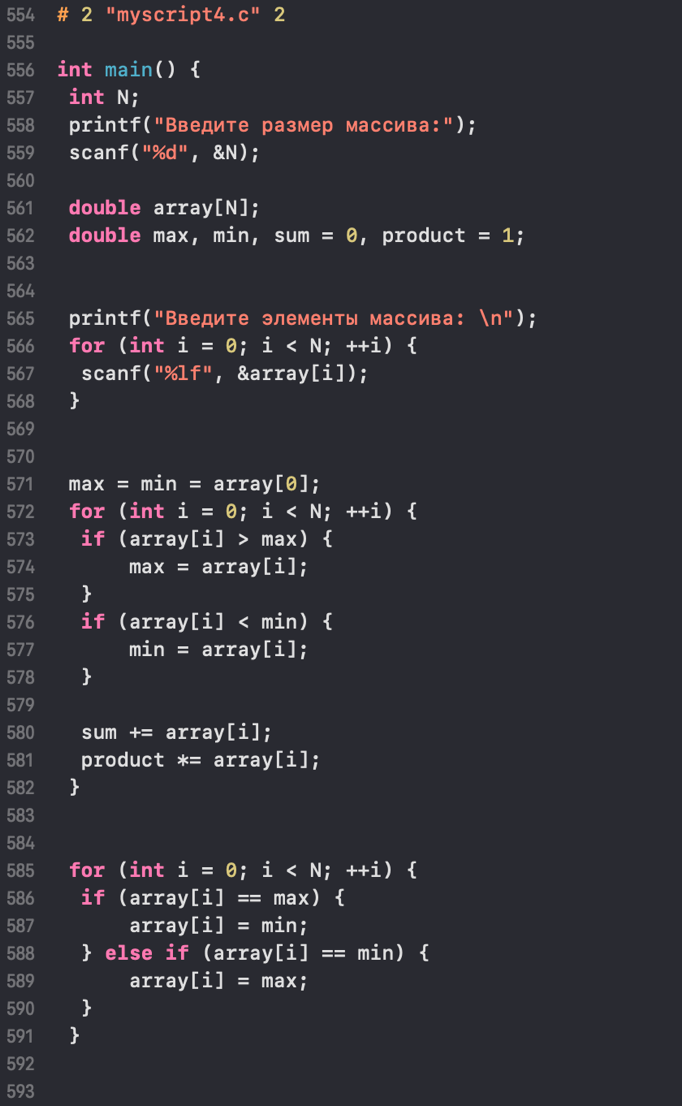
*Выполняем скрипты*

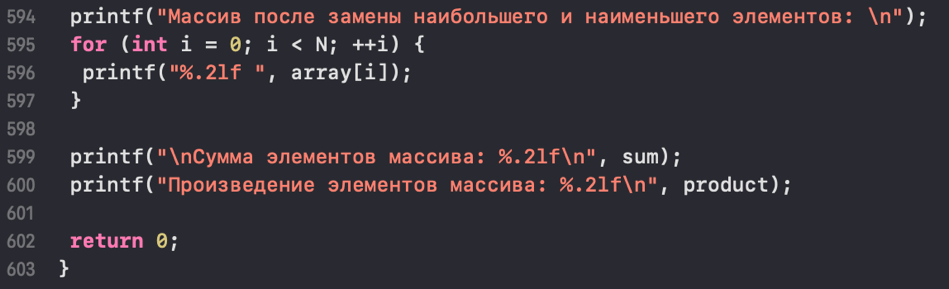


*Получаем файлы всех стадий компиляции*

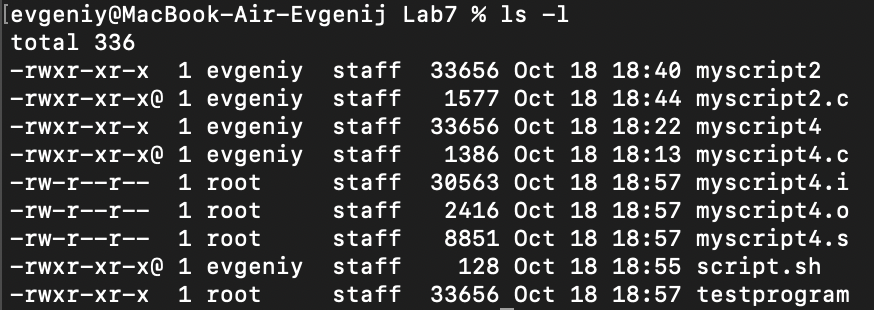


*Находим наш код в файлах препроцессора*

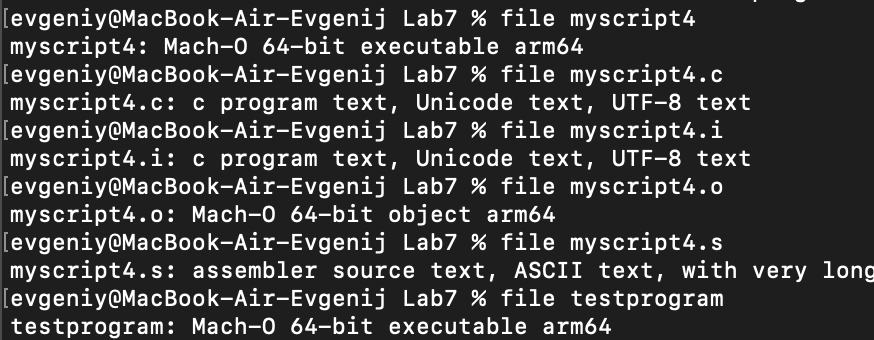




*Просмотрим размер файлов*



Просмотрим их расширения



## Задание 2.

Используя материалы (исходные файлы) упражнения №1 и №2 создайте один многофайловый проект, руководствуясь принципом Single Responsibility Principle. Каждое задание должно быть представлено отдельным исходным файлом. Для связывания файлов проекта обязательно используйте заголовочные файлы. Все файлы могут располагаться в одной директории.

Все задания должны вызываться консольным меню. Для выхода из приложения предусмотрите отдельный параметр.

Проведите автосборку проект с прохождением всех стадий компиляции с использованием утилиты make.

**Упражнение 2**

Изучите и выполните требования, представленные в лабораторной работе №3 «Многофайловые проекты и шаблон MVC». Разработайте исходные коды приложений в соответствии с заданиями.

Компиляцию приложений проведите с помощью консольных команд.

При разработке приложений не забудьте отразить информацию о разработчиках.

Файлы с содержанием лабораторной работы предоставлены.

1) Разработать многофайловый консольный проект на С/С++ согласно варианту задания с использованием шаблона (паттерна) проектирования MVC.

2) Размерность матрицы в индивидуальном задании задаётся пользователем на стадии выполнения программы. Для повышения производительности программы использовать динамическое выделение памяти.

3) ЗАПРЕЩАЕТСЯ в программе использовать под любым предлогом ГЛОБАЛЬНЫЕ переменные!

4) Каждое задание оформить в виде отдельной бизнес-функции.

5) Все функции должны быть сгруппированы по соответствующим отдельным файлам.

6) Все функции должны быть самодостаточные, т.е. при их разработке необходимо придерживаться принципа Single Responsibility Principle.

7) При выполнении задания запрещается использовать интегрированные средства разработки (Integrated Development Environment, IDE). Рекомендуется задействовать любой текстовый редактор (к примеру, gedit) и набор компиляторов GNU Compiler Collection (GCC), в частности, компиляторы языков программирования С/C++ gcc/g++, а для отладки – gdb-отладчик.

8) При разработке программ необходимо придерживаться соглашения по написанию кода на C/C++ (Code-Convention).

**Основное задание**

Разработать интерактивную программу «Try to Guess the Number» («Попробуй угадать число»), которая эмулирует классическую игру на отгадывание числа.

Суть программы (игры) сводиться к следующему: компьютер генерирует случайное число из диапазона, к примеру, от 1 до 100, а пользователь (далее игрок) пытается отгадать число за минимальное количество попыток. При каждой очередной попытке компьютер «подсказывает» игроку, как соизмеряется предложенный вариант игрока с действительным загаданным компьютером числом: загаданное число больше или меньше указанного (higher/lower). Как только игрок отгадывает число, компьютер должен «поздравить» его с выводом на экран угаданного числа и количество затраченных игроком попыток.

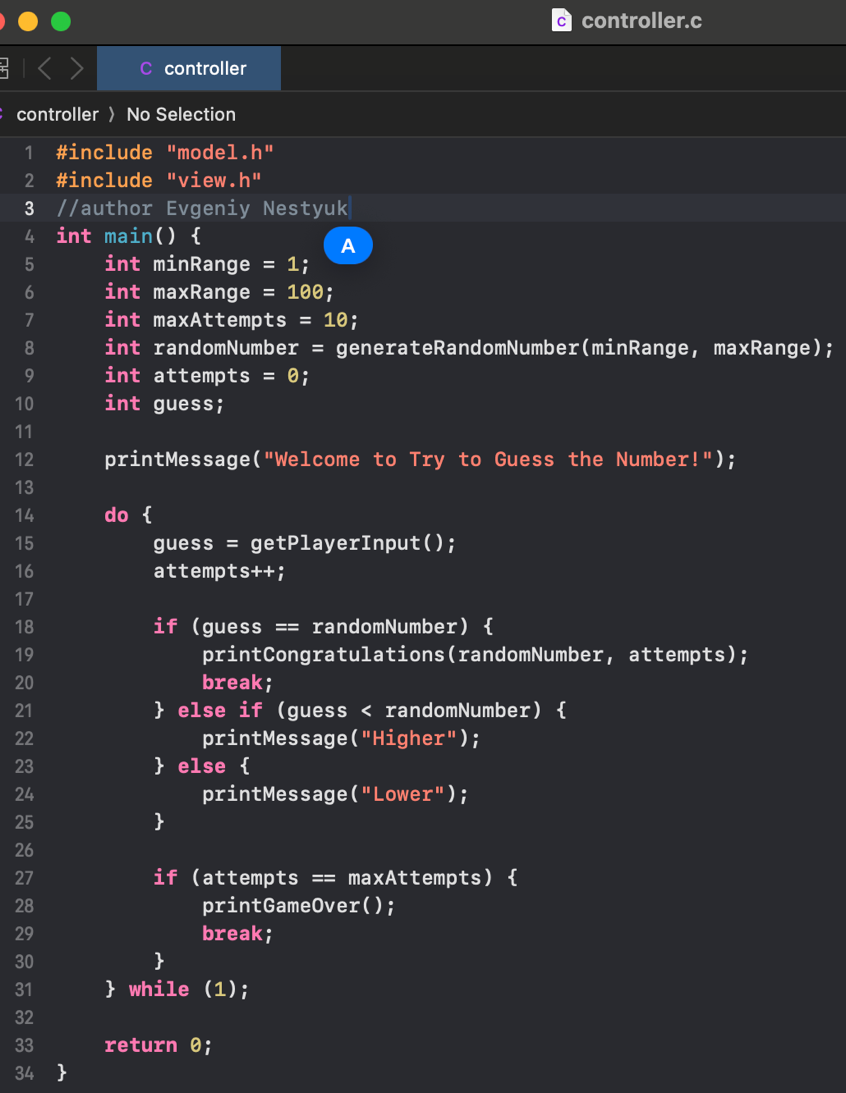
Далее компьютер может «предложить» повторно сыграть игру или выйти из программы.

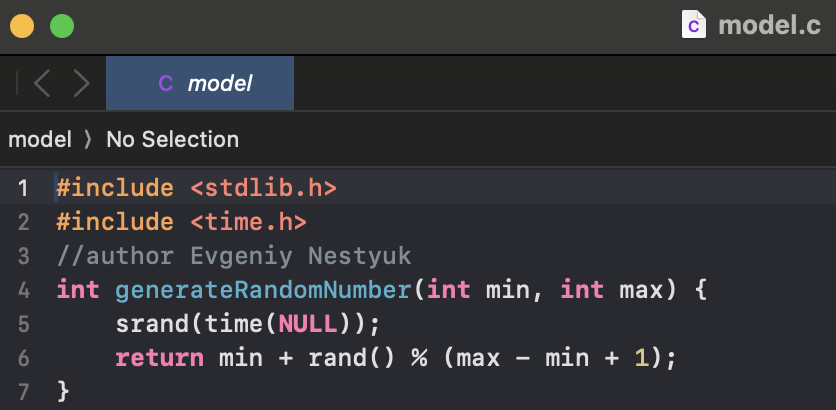
Для универсальности предложенной программы можно добавить возможность выбора диапазона генерирования компьютером случайных чисел, а также задания ограничения на количество попыток. В случае, если игрок не укладывается в заданное количество попыток (т.е. проигрывает), программа должна выводить суровую надпись «Game Over».

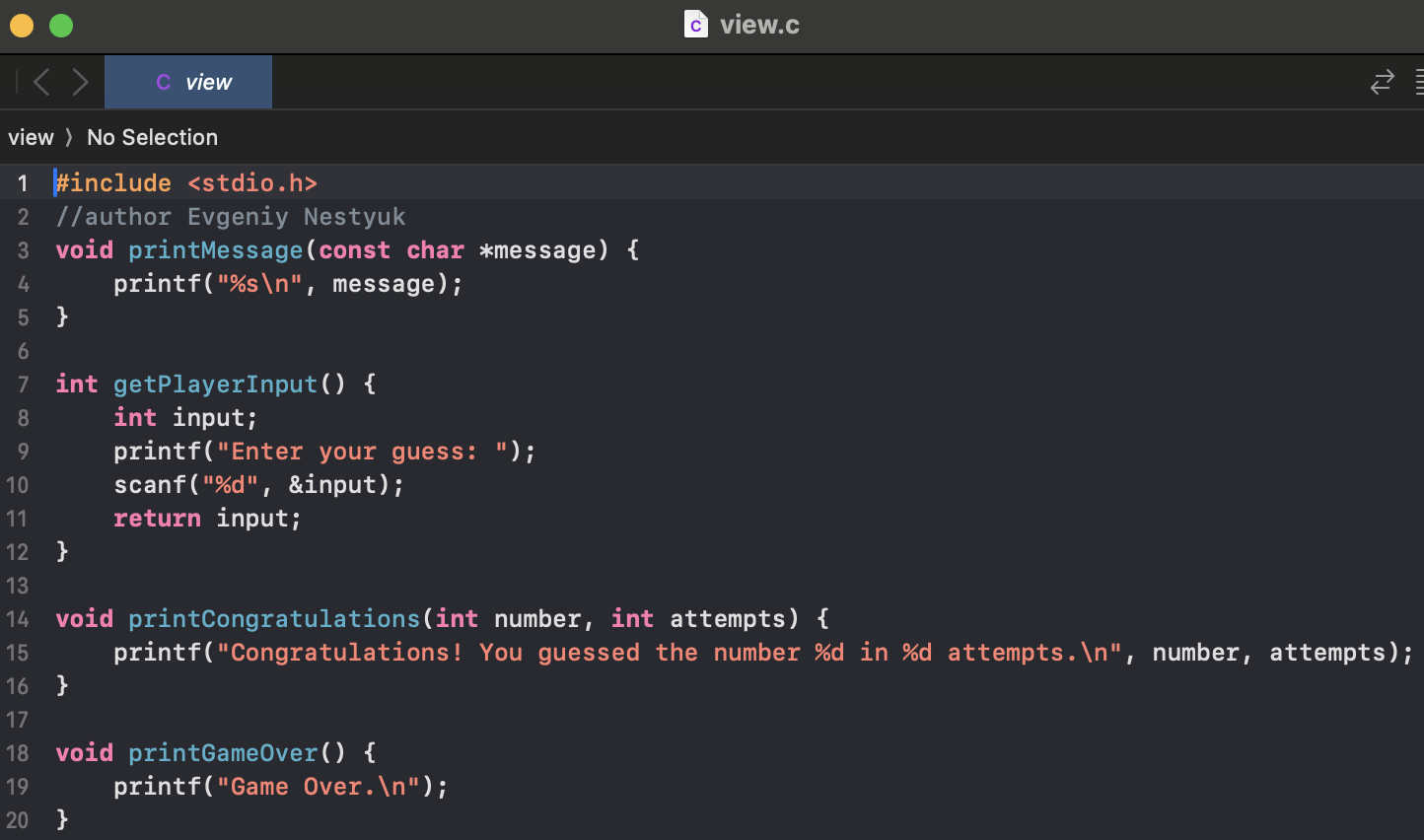
8) Найти номер строки заданной матрицы, в которой находится самая длинная серия (последовательность одинаковых элементов)

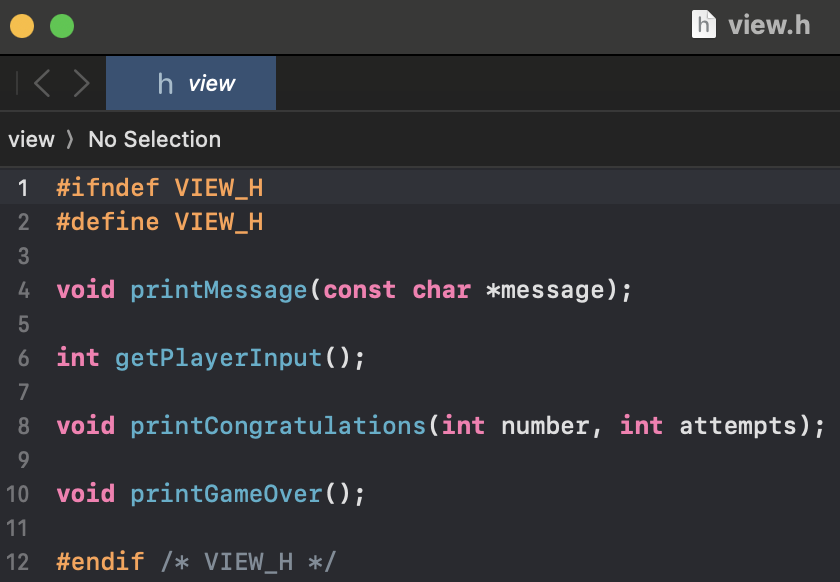
Создадим в одной директории файлы с исходным кодом model.c controller.c view.c, а также заголовки view.h model.h

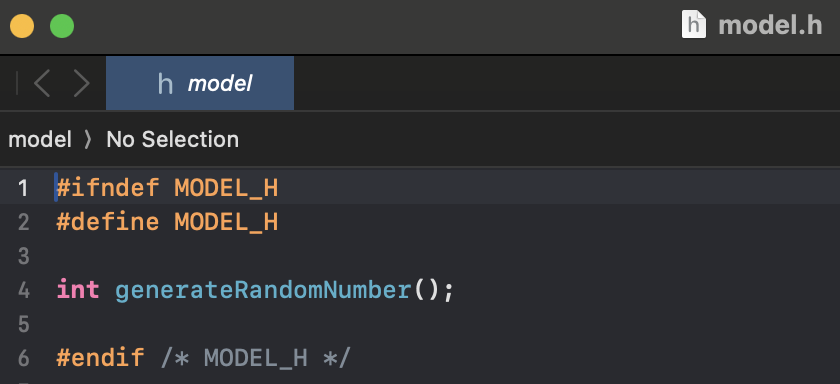
*Основное задание*

**

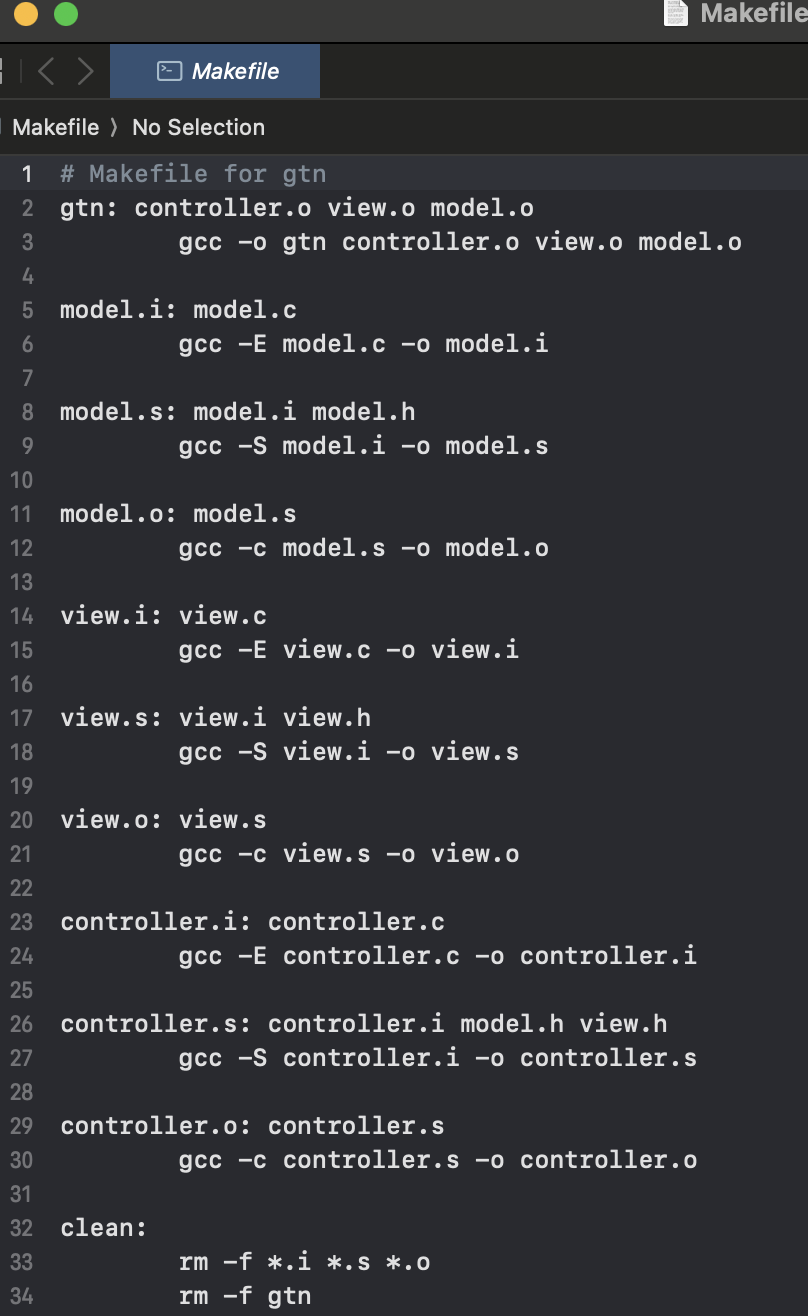




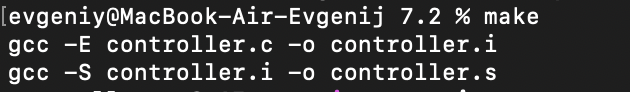


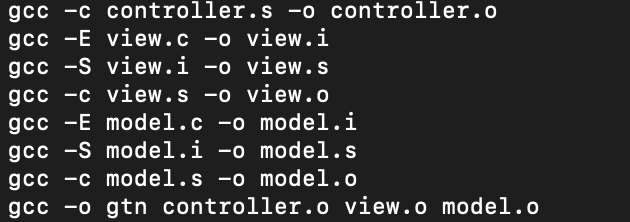


*Напишем Makefile с явным прохождением всех этапов компиляции*

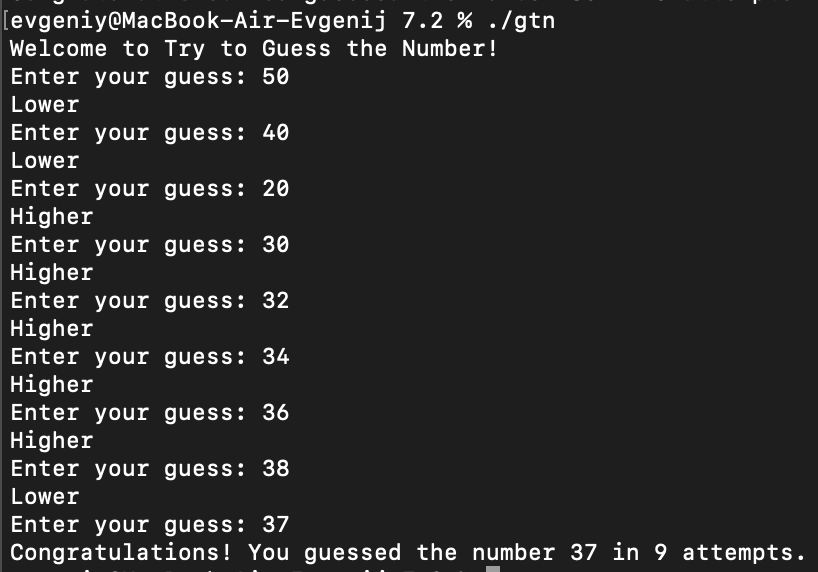


*Выполним команду make*



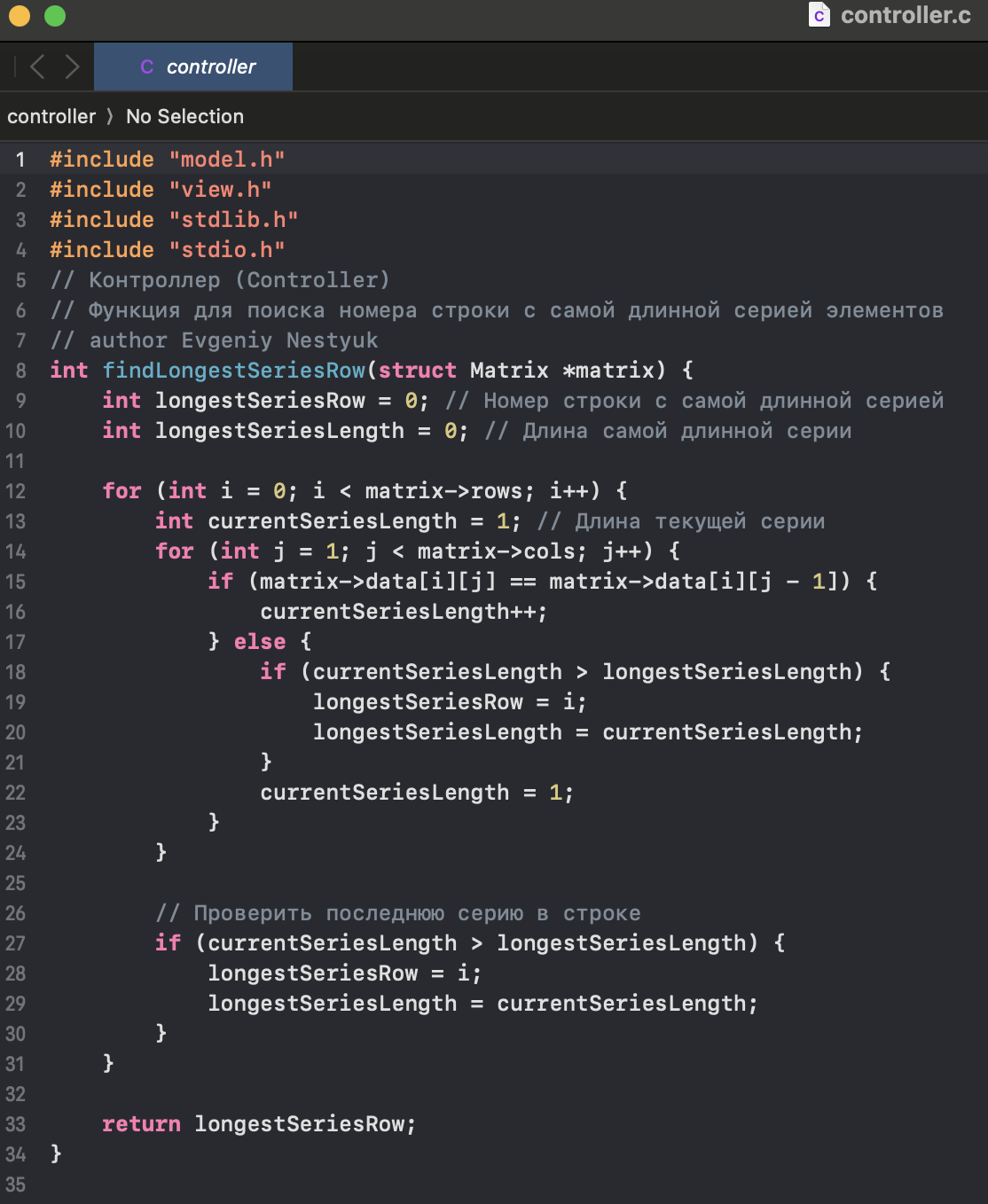


*Протестируем работу программы*

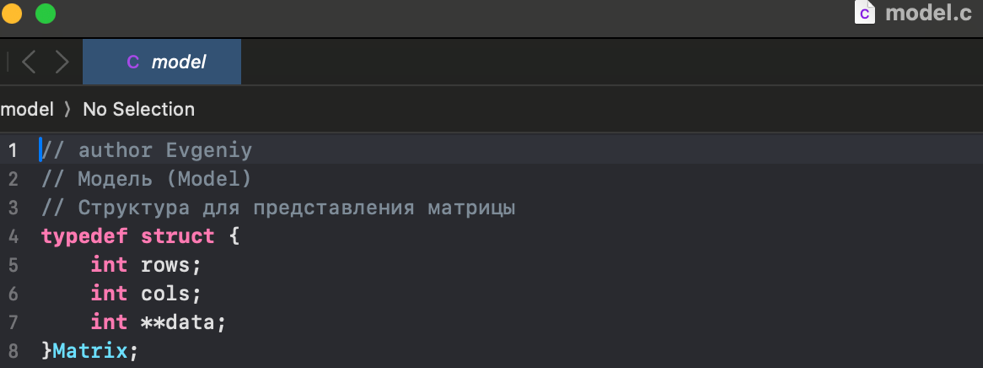


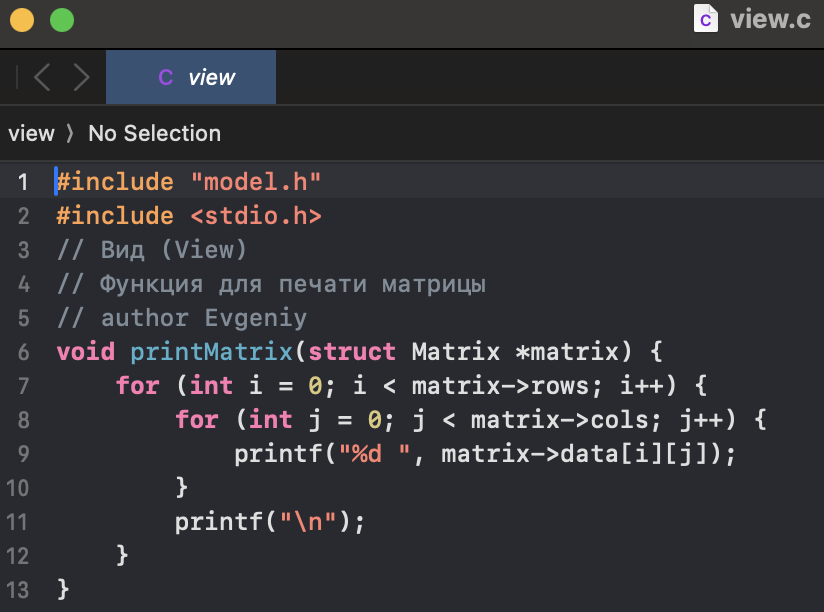
Индивидуальное задание

*Создаем файлы по аналогии*

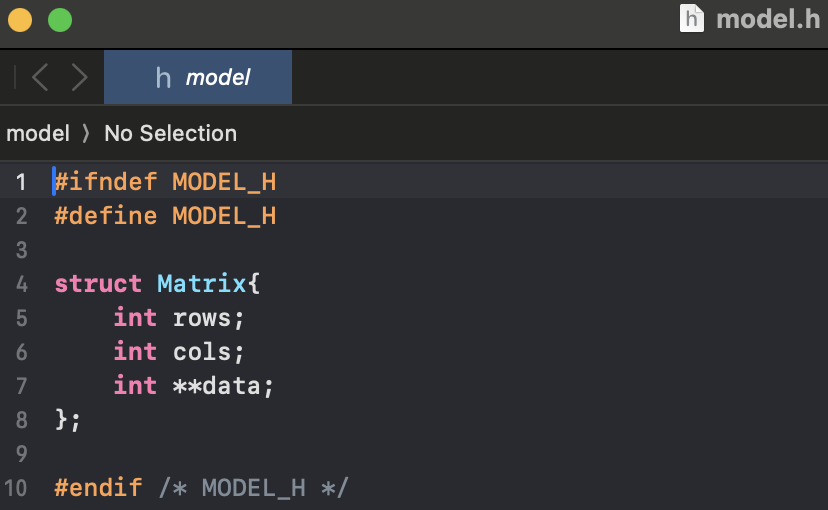


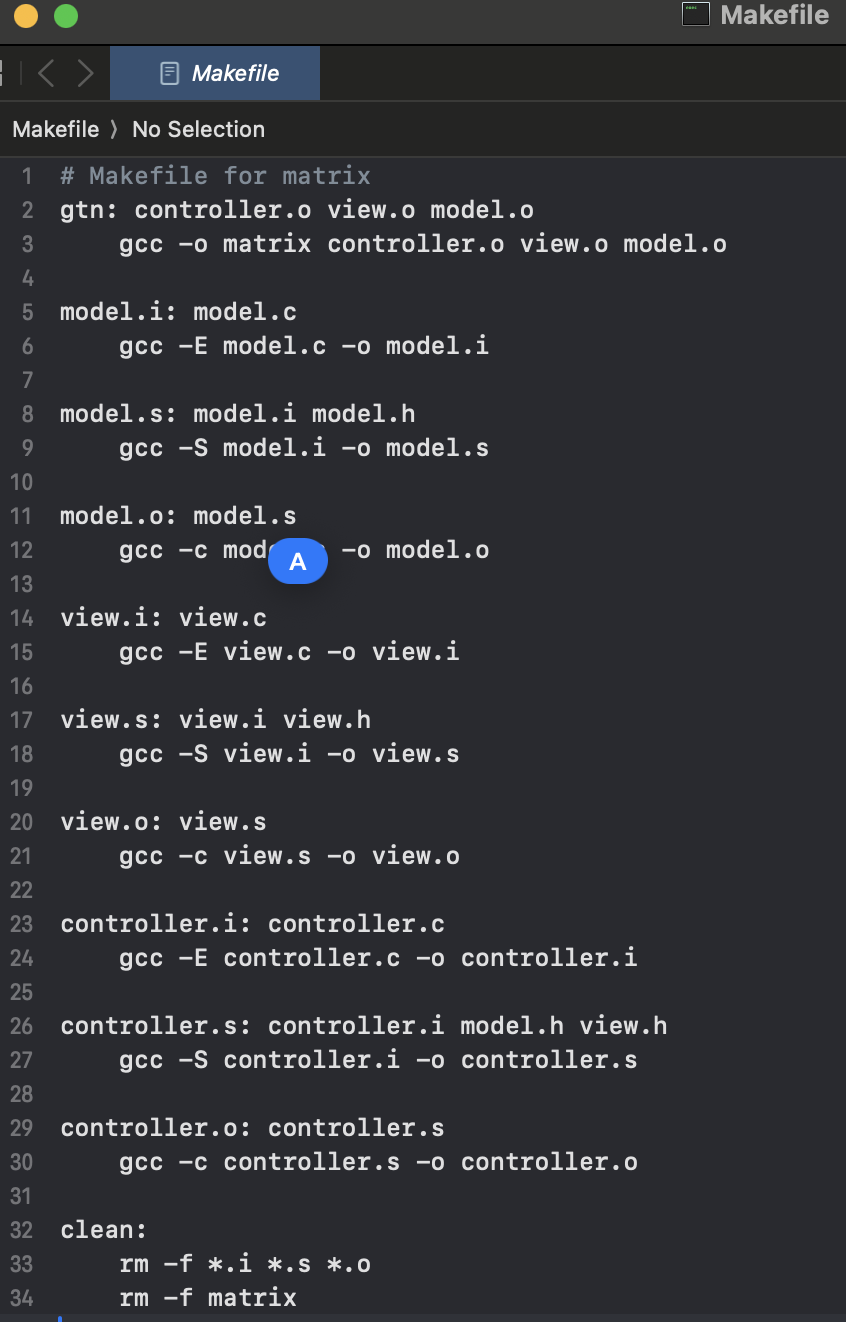




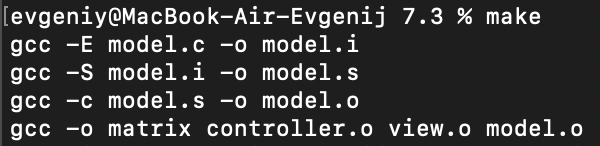








*Выполним команду make*



*Протестируем работу программы*



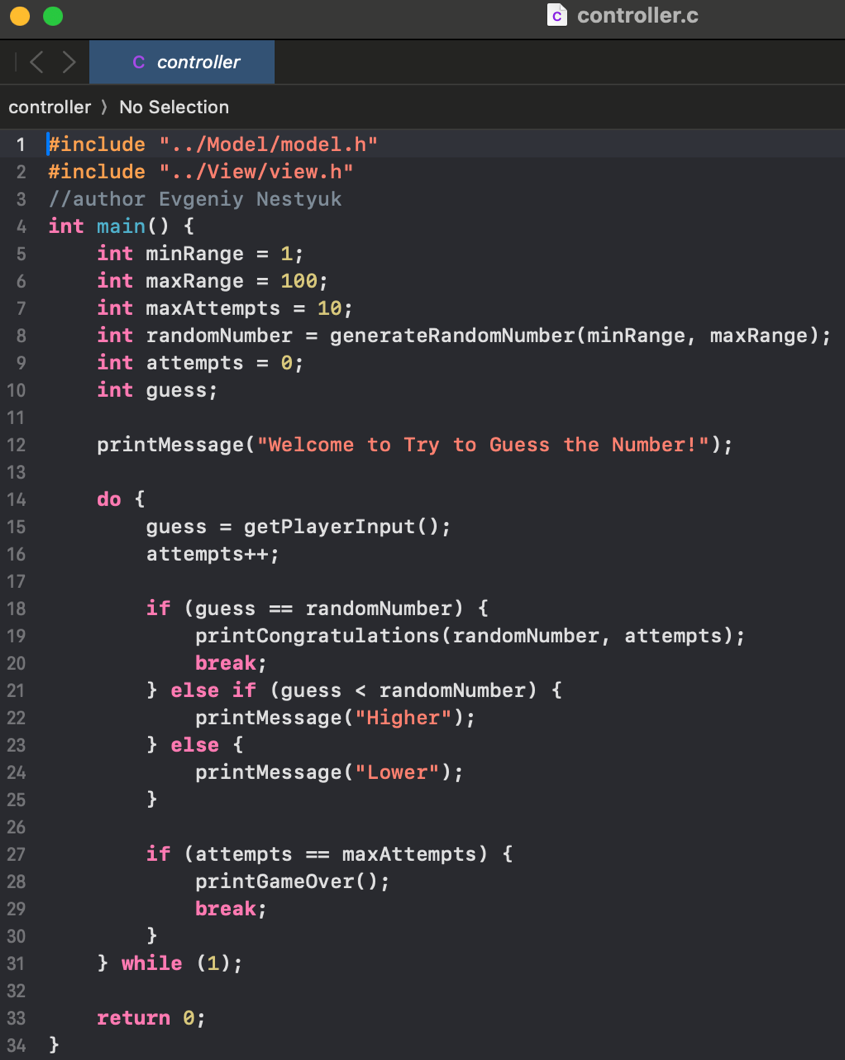
## Задание 3

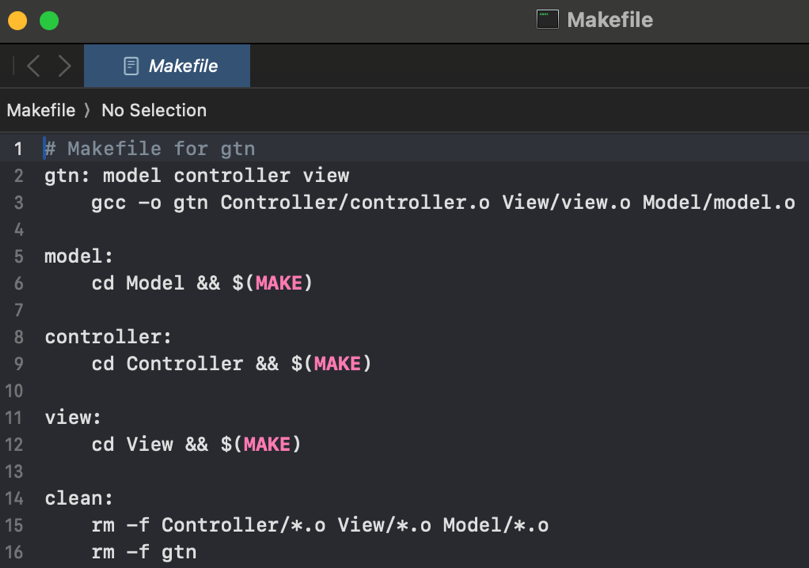
Используя материалы (исходные файлы) задания №2 создайте один многофайловый проект, руководствуясь принципом Single Responsibility Principle. Каждое задание должно быть представлено отдельным исходным файлом. Используйте архитектурный шаблон проектирования MVC. Здесь под шаблоном MVC понимается файлы или группа файлов в трех отдельных директориях (папках), а не просто три отдельных файла.

Все задания должны вызываться консольным меню. Для выхода из приложения предусмотрите отдельный параметр. Для связывания файлов проекта обязательно используйте заголовочные файлы.

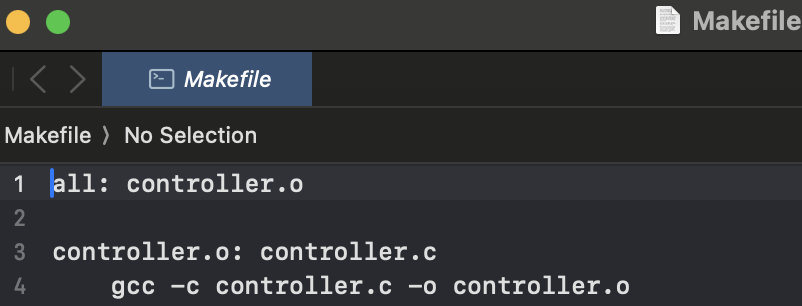
Проведите рекурсивную автосборку проект с прохождением всех стадий компиляции с использованием утилиты make.

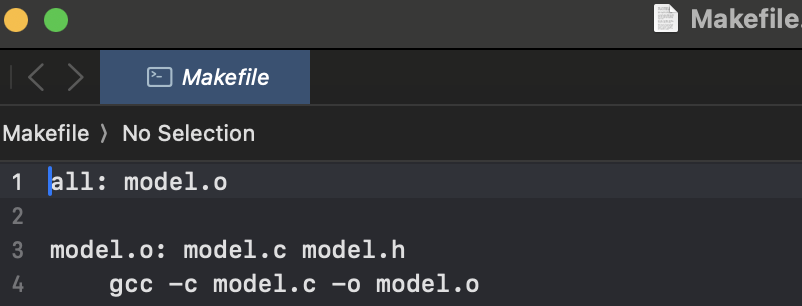
*Расположим файлы из задания 2 по раздельным директориям и немного изменим подключение заголовков в controller.c и изменим Makefile для рекурсивной сборки*

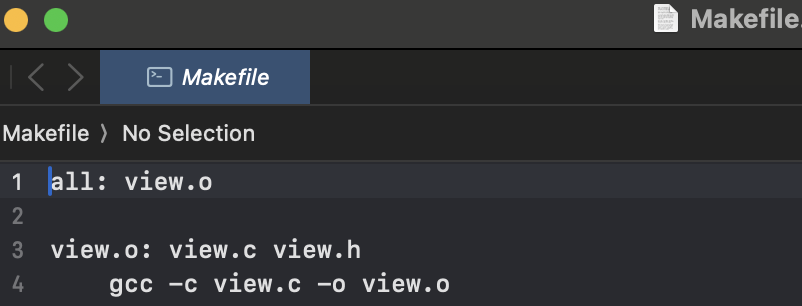




*Добавим в каждую директорию свои Makefile*



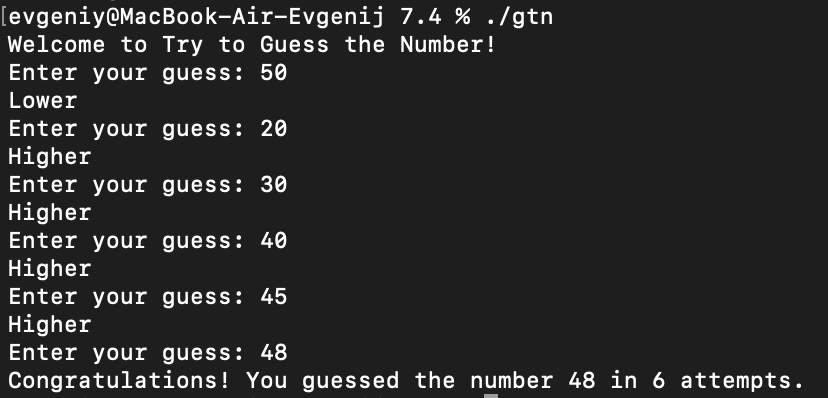




*Собираем проект*



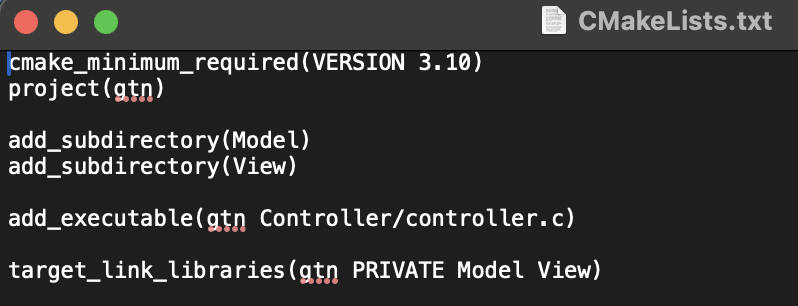
*Выполняем его*



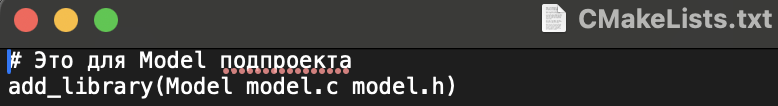
## Задание «Бонус»

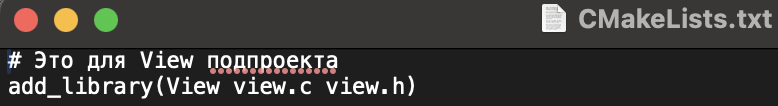
Проведите рекурсивную автосборку с использованием утилиты cmake проекта с исходными файлами задания №3.

*Создадим файл CMakeLists.txt*

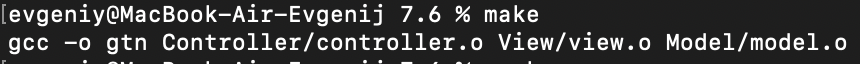


*Файлы с аналогичным названием создадим в папках Model и View*

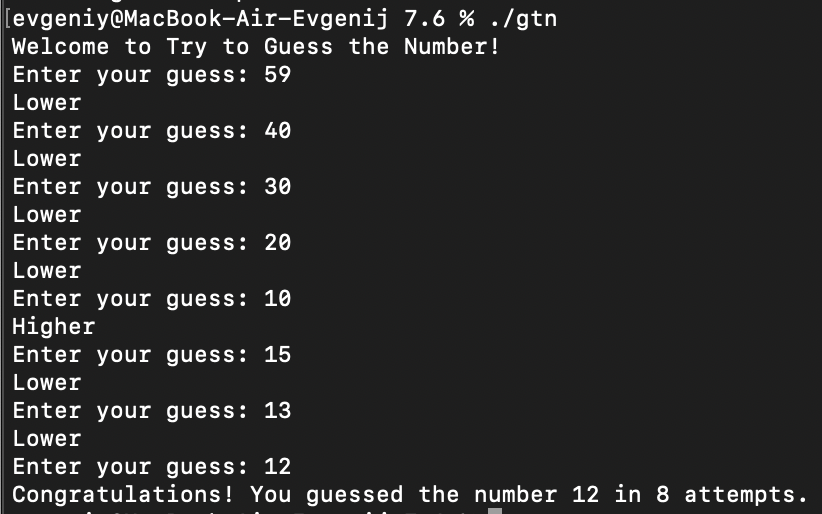




*Генерируем настроек сборки на основе инструкций, записанных в файлах CMakeLists.txt*



*Выполним проект*



**Контрольные вопросы:**

1) Перечислите основные инструментарии для разработки программ?

2) Что такое исходные файлы программы?

3) Что такое исполняемые файлы программы?

4) Какой командой можно определить тип файла? Определите типы

файлов вашей сборки проекта?

5) Назовите основные стадии компиляции программы?

6) В чем различие компиляции и интерпритации программы?

7) Приведите известные вам примеры компилируемых и

интерпритируемых языков программирования?

8) Что такое препроцессор и зачем он нужен?

9) В чем разница между понятиями «аргумент функции» и «параметр

функции» ?

10) В чем разница использования двух форм директивы: #include

<filename> или #include “filename”?

11) Что такое автосборка? Какие приемущества она дает?

12) Какой командой можно определить определить формат файла?

13) Что такое утилита make?

14) В чем разница автосборки скриптами bash и утилитой make?

1. **Основные инструментарии для разработки программ:**
   * Компиляторы (например, GCC, Clang)
   * Текстовые редакторы (например, Vim, Nano, Emacs)
   * Интегрированные среды разработки (IDE) (например, Visual Studio Code, JetBrains IDEs)
   * Отладчики (например, GDB)
   * Системы контроля версий (например, Git)
   * Утилиты сборки (например, make, CMake)
2. **Исходные файлы программы:** Исходные файлы программы - это файлы, содержащие исходный код программы, который разработчик пишет. Это обычно файлы с расширениями, соответствующими языку программирования (например, .c для языка C, .py для Python).
3. **Исполняемые файлы программы:** Исполняемые файлы - это бинарные файлы, содержащие машинный код программы, который компьютер может выполнять как программу. В Linux они часто не имеют расширения или имеют расширение .out, .bin, .elf и другие.
4. **Команда для определения типа файла:** Команда file. Например, file myprogram покажет информацию о типе файла myprogram.
5. **Основные стадии компиляции программы:**
   * Препроцессинг
   * Компиляция
   * Ассемблирование
   * Связывание (линковка)
6. **Различие между компиляцией и интерпретацией программы:** При компиляции исходный код преобразуется в машинный код один раз, создавая исполняемый файл. В интерпретации исходный код выполняется построчно интерпретатором во время выполнения. Примеры компилируемых языков: C, C++, примеры интерпретируемых языков: Python, JavaScript.
7. **Примеры компилируемых и интерпретируемых языков программирования:**
   * Компилируемые: C, C++, Rust
   * Интерпретируемые: Python, JavaScript, Ruby
8. **Препроцессор и его назначение:** Препроцессор - это часть компилятора, которая выполняет текстовую обработку исходного кода перед компиляцией. Он позволяет включать файлы, определять макросы и выполнять другие текстовые преобразования.
9. **Разница между аргументом и параметром функции:**
   * Аргумент функции - это значение, передаваемое функции при вызове.
   * Параметр функции - это переменная, объявленная в определении функции, которая получает значение аргумента при вызове функции.
10. **Разница между #include <filename> и #include "filename":** <filename> используется для включения системных заголовочных файлов, а "filename" для включения пользовательских заголовочных файлов.
11. **Автосборка (сборка проекта):** Это процесс автоматизации сборки проекта с использованием утилиты, такой как make. Преимущества включают автоматизацию, управление зависимостями и сокращение ошибок при сборке.
12. **Команда для определения формата файла:** Команда file (например, file myfile).
13. **Утилита make:** make - это утилита для автоматизации процесса сборки программных проектов. Она использует файл Makefile, который содержит инструкции о том, как собрать проект.
14. **Разница между автосборкой скриптами bash и утилитой make:** Скрипты bash могут быть менее удобными для сложных проектов, так как make обрабатывает зависимости между файлами автоматически и обновляет только необходимые компоненты, что делает процесс более эффективным.