Лабораторные работы
по прикладному программированию

INTRO

Уважаемые студенты,

Добро пожаловать на курс по языку программирования Go! В этом курсе вы будете выполнять ряд лабораторных работ, направленных на развитие ваших навыков программирования и создания реальных приложений. Для успешного завершения курса и оценки ваших лабораторных работ, важно соблюдать следующие правила:

1. **Репозиторий на GitHub:** Все ваши лабораторные работы должны быть размещены в репозитории на GitHub. Это позволит нам легко проверять и комментировать ваши работы, а также даст вам возможность отслеживать свой прогресс.

2. **Структура проекта:** В каждом репозитории создайте отдельные папки для каждой лабораторной работы. Внутри папок должны быть `README`-файлы, описывающие каждое задание, и исходный код с примерами решений.

3. **Код и документация:** Ваш код должен быть четко структурирован и документирован. Убедитесь, что ваш код компилируется и работает корректно. В `README`-файле укажите инструкции по запуску и тестированию вашего кода.

4. **Ссылки на репозитории:** По мере выполнения лабораторных работ, предоставляйте ссылки на ваши репозитории в соответствующих отчетах или заданиях. Это поможет нам оперативно оценить ваши работы.

5. **Периодические обновления:** Регулярно обновляйте свои репозитории, добавляя новые лабораторные работы и исправляя ошибки по мере их обнаружения. Это покажет вашу активность и прогресс в изучении курса.

Если у вас возникнут вопросы или трудности с выполнением заданий, не стесняйтесь обращаться за помощью. Удачи в изучении Go и успешного выполнения лабораторных работ!

Лабораторная работа 1

1. Написать программу, которая выводит текущее время и дату.
2. Создать переменные различных типов (`int`, `float64`, `string`, `bool`) и вывести их на экран.
3. Использовать краткую форму объявления переменных для создания и вывода переменных.
4. Написать программу для выполнения арифметических операций с двумя целыми числами и выводом результатов.
5. Реализовать функцию для вычисления суммы и разности двух чисел с плавающей запятой.
6. Написать программу, которая вычисляет среднее значение трёх чисел.

Лабораторная работа 2

1. Написать программу, которая определяет, является ли введенное пользователем число четным или нечетным.
2. Реализовать функцию, которая принимает число и возвращает «Positive», «Negative» или «Zero».
3. Написать программу, которая выводит все числа от 1 до 10 с помощью цикла `for`.
4. Написать функцию, которая принимает строку и возвращает ее длину.
5. Создать структуру `Rectangle` и реализовать метод для вычисления площади прямоугольника.
6. Написать функцию, которая принимает два целых числа и возвращает их среднее значение.

Лабораторная работа 3

1. Создать пакет `mathutils` с функцией для вычисления факториала числа.
2. Использовать созданный пакет для вычисления факториала введенного пользователем числа.
3. Создать пакет `stringutils` с функцией для переворота строки и использовать его в основной программе.

4. Написать программу, которая создает массив из 5 целых чисел, заполняет его значениями и выводит их на экран.

5. Создать срез из массива и выполнить операции добавления и удаления элементов.

6. Написать программу, которая создает срез из строк и находит самую длинную строку.

Лабораторная работа 4

1. Написать программу, которая создает карту с именами людей и их возрастами. Добавить нового человека и вывести все записи на экран.

2. Реализовать функцию, которая принимает карту и возвращает средний возраст всех людей в карте.

3. Написать программу, которая удаляет запись из карты по заданному имени.

4. Написать программу, которая считывает строку с ввода и выводит её в верхнем регистре.

5. Написать программу, которая считывает несколько чисел, введенных пользователем, и выводит их сумму.

6. Написать программу, которая считывает массив целых чисел и выводит их в обратном порядке.

Лабораторная работа 5

1. Создать структуру `Person` с полями `name` и `age`. Реализовать метод для вывода информации о человеке.

2. Реализовать метод `birthday` для структуры `Person`, который увеличивает возраст на 1 год.

3. Создать структуру `Circle` с полем `radius` и метод для вычисления площади круга.

4. Создать интерфейс `Shape` с методом `Area()`. Реализовать этот интерфейс для структур `Rectangle` и `Circle`.

5. Реализовать функцию, которая принимает срез интерфейсов `Shape` и выводит площадь каждого объекта.

6. Создать интерфейс `Stringer` и реализовать его для структуры `Book`, которая хранит информацию о книге.

Лабораторная работа 6

1. Создание и запуск горутин:

   - Напишите программу, которая параллельно выполняет три функции (например, расчёт факториала, генерация случайных чисел и вычисление суммы числового ряда);
   - Каждая функция должна выполняться в своей горутине.
   - Добавьте использование `time.Sleep()` для имитации задержек и продемонстрируйте параллельное выполнение.

2. Использование каналов для передачи данных:

   - Реализуйте приложение, в котором одна горутина генерирует последовательность чисел (например, первые 10 чисел Фибоначчи) и отправляет их в канал.
   - Другая горутина должна считывать данные из канала и выводить их на экран.
   - Добавьте блокировку чтения из канала с помощью `close()` и объясните её роль.

3. Применение `select` для управления каналами:

   - Создайте две горутины, одна из которых будет генерировать случайные числа, а другая — отправлять сообщения об их чётности/нечётности.
   - Используйте конструкцию `select` для приёма данных из обоих каналов и вывода результатов в консоль.
   - Продемонстрируйте, как `select` управляет многоканальными операциями.

4. Синхронизация с помощью мьютексов:

   - Реализуйте программу, в которой несколько горутин увеличивают общую переменную-счётчик.
   - Используйте мьютексы (`sync.Mutex`) для предотвращения гонки данных.

5. Разработка многопоточного калькулятора:

   - Напишите многопоточный калькулятор, который одновременно может обрабатывать запросы на выполнение простых операций (`+`, `-`, `*`, `/`).
   - Используйте каналы для отправки запросов и возврата результатов.

- Организуйте взаимодействие между клиентскими запросами и серверной частью калькулятора с помощью горутин.

6. Создание пула воркеров:

   - Реализуйте пул воркеров, обрабатывающих задачи (например, чтение строк из файла и их реверсирование).
   - Количество воркеров задаётся пользователем.
   - Распределение задач и сбор результатов осуществляется через каналы.
   - Выведите результаты работы воркеров в итоговый файл или в консоль.

Лабораторная работа 7

1. Создание TCP-сервера:

   - Реализуйте простой TCP-сервер, который слушает указанный порт и принимает входящие соединения.
   - Сервер должен считывать сообщения от клиента и выводить их на экран.
   - По завершении работы клиенту отправляется ответ с подтверждением получения сообщения.

2. Реализация TCP-клиента:

   - Разработайте TCP-клиента, который подключается к вашему серверу.
   - Клиент должен отправлять сообщение, введённое пользователем, и ожидать ответа.
   - После получения ответа от сервера клиент завершает соединение.

3. Асинхронная обработка клиентских соединений:

   - Добавьте в сервер многопоточную обработку нескольких клиентских соединений.
   - Используйте горутины для обработки каждого нового соединения.
   - Реализуйте механизм graceful shutdown: сервер должен корректно завершать все активные соединения при остановке.

4. Создание HTTP-сервера:

   - Реализуйте базовый HTTP-сервер с обработкой простейших `GET` и `POST` запросов.

- Сервер должен поддерживать два пути:
  - `GET/hello` — возвращает приветственное сообщение;
  - `POST/data` — принимает данные в формате JSON и выводит их содержимое в консоль.

5. Добавление маршрутизации и middleware:

   - Реализуйте обработку нескольких маршрутов и добавьте middleware для логирования входящих запросов.
   - Middleware должен логировать метод, URL, и время выполнения каждого запроса.

6. Веб-сокеты:

   - Реализуйте сервер на основе веб-сокетов для чата.
   - Клиенты должны подключаться к серверу, отправлять и получать сообщения.
   - Сервер должен поддерживать несколько клиентов и рассылать им сообщения, отправленные любым подключённым клиентом.

Лабораторная работа 8

1. Построение базового REST API:

   - Реализуйте сервер, поддерживающий маршруты:
     - `GET/users` — получение списка пользователей;
     - `GET/users/{id}` — получение информации о конкретном пользователе;
     - `POST/users` — добавление нового пользователя;
     - `PUT/users/{id}` — обновление информации о пользователе;
     - `DELETE/users/{id}` — удаление пользователя.

2. Подключение базы данных:

   - Добавьте базу данных (например, PostgreSQL или MongoDB) для хранения информации о пользователях;
   - Модифицируйте сервер для взаимодействия с базой данных.

3. Обработка ошибок и валидация данных:

   - Реализуйте централизованную обработку ошибок.
   - Добавьте валидацию данных при создании и обновлении пользователей.

4. Пагинация и фильтрация:

    - Добавьте поддержку пагинации и фильтрации по параметрам запроса (например, поиск пользователей по имени или возрасту).

5. Тестирование API:

    - Реализуйте unit-тесты для каждого маршрута.
    - Проверьте корректность работы при различных вводных данных.

6. Документация API:

    - Создайте документацию для разработанного API с описанием маршрутов, методов, ожидаемых параметров и примеров запросов.

Лабораторная работа 9

1. Создание клиентской программы:

    - Напишите клиентское приложение, которое отправляет запросы на ваш сервер.
    - Реализуйте простое меню с возможностью выполнения CRUD операций.

2. Обработка ответов:

    - Реализуйте механизм обработки ответов и ошибок.
    - Добавьте функции для форматирования и вывода полученных данных на экран.

3. Интерфейс пользователя:

    - Разработайте консольный интерфейс с меню для пользователя.
    - Включите возможность добавления, удаления и обновления информации о пользователях.

4. Авторизация пользователя:

    - Добавьте поддержку авторизации пользователя.
    - Клиент должен сохранять токен сессии и передавать его в заголовках последующих запросов.

5. Поддержка нескольких клиентов:

    - Модифицируйте клиентскую часть так, чтобы несколько клиентов могли одновременно взаимодействовать с сервером.

6. Тестирование и отладка:

    - Проведите тестирование взаимодействия клиента и сервера.
    - Проверьте работу всех маршрутов и функций.

Лабораторная работа 10

1. Хэширование данных:

    - Разработайте утилиту, которая принимает на вход строку и вычисляет её хэш с использованием алгоритма `SHA-256`.
    - Реализуйте возможность выбора нескольких хэш-функций (например, `MD5`, `SHA-256`, `SHA-512`).
    - Включите в утилиту проверку целостности данных: пользователю предлагается ввести строку и её хэш, после чего утилита должна подтвердить или опровергнуть их соответствие.

2. Симметричное шифрование:

    - Реализуйте программу, шифрующую переданные данные с помощью алгоритма `AES`.
    - Пользователь должен указать строку и секретный ключ.
    - Программа должна зашифровать строку и предоставить возможность расшифровать её при вводе того же ключа.

3. Асимметричное шифрование и цифровая подпись:

    - Создайте пару ключей (открытый и закрытый) и сохраните их в файл.
    - Реализуйте программу, которая подписывает сообщение с помощью закрытого ключа и проверяет подпись с использованием открытого ключа.
    - Продемонстрируйте пример передачи подписанных сообщений между двумя сторонами.

4. Реализация защищённого канала передачи данных (TLS):

    - Модифицируйте TCP-сервер и клиент из предыдущих лабораторных работ для работы через защищённый канал с использованием TLS.
    - Сервер должен поддерживать установку безопасного соединения, а клиент — проверять сертификат сервера перед отправкой данных.
    - Реализуйте взаимную аутентификацию на уровне сертификатов.

5. Защита REST API:

   - Добавьте поддержку аутентификации с помощью токенов (например, JWT) для REST API.
   - Реализуйте маршруты, требующие аутентификации, и проверьте их работу с валидными и невалидными токенами.
   - Включите ограничение доступа к ресурсам на основе ролей (`admin`, `user`).

6. Защита от CSRF и управление сессиями:

   - Добавьте защиту от CSRF-атак в ваше REST API.
   - Реализуйте механизм сессий для авторизованных пользователей.