!ВАЖНОЕ ПРИМЕЧАНИЕ!

Так как вопросы будут только из билетов, следует делать только те вопросы, что помечены “**ЕСТЬ В БИЛЕТАХ**“

Всё, что зачёркнуто, в билетах наших нет (Alt+Shift+5 - зачеркнуть)

Парни, для более удобного чтения и редактирования документа предлагаю ввести немного обозначений

- **Жирное выделение** - сами вопросы и параграфы.
- Красное выделение - часть вопроса, на который ещё нет ответа.
- Зелёный - отвеченный вопрос/параграф

Пишите, откуда вы взяли ответ, если это было с какого-либо сайта

Вопросы ООП

https://struchkov.dev/blog/ru/modifiers-in-java/#модификатор-abstract - тут хорошая инфа про static, final, abstract

1. Понятие класса и объекта. Определение понятий. Статический и не статический контекст класса. Члены класса. Методы, поля, конструкторы, блоки инициализации. Ключевые слова abstract и final; ЕСТЬ В БИЛЕТАХ

Класс в ООП - это шаблон, описывающий структуру и поведение объектов. Он содержит методы, поля, конструкторы и блоки инициализации. Объект - это экземпляр класса, обладающий конкретными значениями свойств и методами. Статический контекст класса относится к статическим членам, а не статический - к обычным членам, привязанным к экземпляру класса.

Метод - функция внутри класса

Поля (атрибуты) - переменные внутри класса

Конструктор класса в объектно-ориентированном программировании (ООП) — это специальный метод, который вызывается при создании нового объекта. Он используется для инициализации полей класса значениями, а также для начальных вычислений, если они необходимы. После создания объекта конструктор вызвать нельзя.

Конструктор без параметров.

Параметризованный конструктор: он принимает один или более аргументов.

Конструктор по умолчанию: он не имеет обязательных аргументов и используется при создании массивов объектов. В отсутствие явно заданного конструктора по умолчанию его код генерируется компилятором.

Блоки инициализации в ООП используются для инициализации полей объектов и классов. Они выполняются при создании объекта или загрузке класса в память.

Существует два типа блоков инициализации: Статические и Динамические. Статический блок используется инициализации статических переменных, а нестатический - для всех остальных. Блоки инициализации позволяют избежать дублирования кода инициализации в нескольких конструкторах класса. Они выполняются перед конструктором, после чего конструктор может дополнительно инициализировать поля.

final - “финальный”, т.е. неизменяемый класс или атрибут или метод (константа)

abstract - абстрактный класс, позволяющий заранее задать шаблон для более конкретных классов. (Абстрактный класс “Машина”, но конкретный класс на его основе “Мотоцикл”). Нельзя создавать экземпляр абстрактного класса. Класс является абстрактным, если хотя бы один из его методов является абстрактным

2. Основополагающие принципы ООП. Инкапсуляция. Средства реализации инкапсуляции. Модификаторы доступа; ЕСТЬ В БИЛЕТАХ

Инкапсуляция - это один из основополагающих принципов объектно-ориентированного программирования. Она заключается в сокрытии внутренней реализации объекта и предоставлении доступа к его данным и методам через специально определенные интерфейсы.

1. Модификаторы доступа: Они определяют уровень доступа к членам класса (полям, методам, конструкторам). Наиболее распространенные модификаторы:

- public: доступ ко всем классам
- private: доступ только внутри класса
- protected: доступ внутри класса и его подклассов
- internal: доступ только внутри пакета

2. Свойства (properties): Свойства позволяют управлять доступом к полям класса, предоставляя методы для чтения и записи значений. Они скрывают реализацию полей и предоставляют контролируемый доступ к ним.
3. Методы доступа (getters и setters): Методы доступа позволяют получать и изменять значения полей класса, обеспечивая инкапсуляцию данных.
4. Вложенные классы: Вложенные классы могут скрывать реализацию внутри родительского класса, предоставляя контролируемый доступ к ним.

3. Основополагающие принципы ООП. Наследование. Управление наследованием; ЕСТЬ В БИЛЕТАХ

Абстракция. В ООП абстракция означает, что для каждого объекта мы задаём минимальное количество методов, полей и описаний, которые позволят нам решить задачу. Чем меньше характеристик, тем лучше абстракция, при этом ключевые характеристики убирать нельзя.

Инкапсуляция. Это свойство системы, позволяющее объединить данные и методы, работающие с ними, в классе и скрыть детали реализации от пользователя.

Наследование. Это свойство системы, позволяющее описать новый класс на основе уже существующего класса с частично или полностью заимствующейся функциональностью. Класс, от которого производится наследование, называется базовым или родительским. Новый класс – потомком, наследником или производным классом.

Полиморфизм – это свойство системы использовать объекты с одинаковым интерфейсом без информации о типе и внутренней структуре объекта. Т.е. объекты разных классов используются одинаково и коду не нужно знать, какой класс он использует.

Наследование в объектно-ориентированном программировании (ООП) представляет собой механизм, позволяющий классу наследовать свойства и методы другого класса, называемого родительским классом или суперклассом. При этом класс, который наследует свойства и методы, называется дочерним классом или подклассом. Наследование обеспечивает повторное использование кода, упрощает его структуру и позволяет создавать иерархии классов.

1. Расширение класса: Подкласс может добавлять новые методы и свойства к тем, которые он унаследовал от суперкласса. Это позволяет расширить функциональность базового класса.
2. Переопределение методов: Подкласс может переопределить методы, унаследованные от суперкласса, чтобы изменить их поведение под конкретные требования подкласса.
3. Вызов методов суперкласса: Подкласс может вызывать методы суперкласса, чтобы использовать их функциональность в дополнение к собственным методам.
4. Множественное наследование интерфейсов: Управление наследованием также включает в себя возможность реализации множественного наследования

интерфейсов, что позволяет классу реализовывать несколько интерфейсов одновременно.

4. Основополагающие принципы ООП. Полиморфизм. Средства реализации полиморфизма; ЕСТЬ В БИЛЕТАХ

Полиморфизм в ООП означает способность объекта использовать методы с одинаковыми именами, но различной реализацией. Это позволяет объектам разных классов реагировать на одинаковые сообщения специфичным для каждого класса способом.

1. Перегрузка методов (method overloading): Позволяет создавать несколько методов с одинаковым именем, но различными параметрами. Компилятор определяет, какой метод вызывать на основе переданных аргументов.
2. Переопределение методов (method overriding): Позволяет подклассам предоставить специфичную реализацию для методов, унаследованных от суперкласса. При вызове метода у объекта подкласса будет использоваться его реализация.
3. Абстрактные классы и интерфейсы: Абстрактные классы и интерфейсы предоставляют абстрактные методы, которые должны быть реализованы в подклассах. Это позволяет использовать полиморфизм для обработки объектов различных типов через общий интерфейс.
4. Динамическое связывание (dynamic binding): Позволяет определить, какой метод вызывать во время выполнения программы, а не во время компиляции. Это обеспечивает гибкость и возможность использования полиморфизма.

5. Понятие класса и интерфейса: абстрактные классы, абстрактные методы. Отличие абстрактного класса от интерфейса; ЕСТЬ В БИЛЕТАХ

Класс в программировании представляет собой шаблон или форму для создания объектов. Он определяет состояние (поля) и поведение (методы) объектов. Экземпляры класса создаются на основе этого шаблона.

Интерфейс определяет, какие методы должны быть реализованы классами, которые его реализуют. В интерфейсе указывается только сигнатура методов (их названия, параметры и возвращаемое значение), но не их реализация. Классы могут реализовывать один или несколько интерфейсов.

Абстрактные классы в объектно-ориентированном программировании — это базовые классы, которые можно наследовать, но нельзя реализовывать. То есть на их основе нельзя создать объект. Подклассы должны реализовать все абстрактные методы или быть также объявлены как абстрактные.

Абстрактный метод — это метод без тела (реализации), который должен быть реализован в подклассах. Абстрактные методы объявляются в абстрактных классах или интерфейсах.

Отличия между абстрактным классом и интерфейсом:

- Абстрактный класс может содержать как абстрактные методы, так и обычные методы с реализацией, в то время как интерфейс содержит только сигнатуры методов без реализации.
- Класс может наследовать только один абстрактный класс, но реализовать несколько интерфейсов.
- В абстрактном классе можно иметь переменные экземпляра, в то время как в интерфейсе могут быть только константы.
- Абстрактный класс может содержать конструктор, но интерфейс не может.

В интерфейсах все методы абстрактные (без реализации), а атрибуты константы

В абстрактных классах методы могут быть не абстрактные. Они могут быть обычные поля. Они оба не могут иметь экземпляра.

6. Интерфейсы: определение, реализация, наследование; ЕСТЬ В БИЛЕТАХ

Интерфейс в ООП представляет собой контракт, определяющий набор методов, свойств и событий, которые должен реализовывать класс. Интерфейс не содержит реализацию, а только объявления членов, которые классы должны реализовать.

Реализация интерфейса:

Класс, реализующий интерфейс, должен предоставить конкретную реализацию всех методов, свойств и событий, объявленных в интерфейсе. Это обеспечивает соответствие контракту, определенному интерфейсом.

Наследование интерфейсов:

Интерфейсы могут наследовать друг от друга, образуя иерархию интерфейсов. Подинтерфейс наследует все члены родительского интерфейса и может добавлять новые члены. Классы, реализующие подинтерфейс, должны реализовывать все члены, объявленные в иерархии интерфейсов.

7. Дженерики: Определение, реализация, примеры; ЕСТЬ В БИЛЕТАХ

https://javarush.com/groups/posts/2004-teorija-dzhenerikov-v-java-ili-gde-na-praktike-stavitjh-skobki

Определение

Дженерики (от англ. "Generics") — это механизм в программировании, позволяющий создавать компоненты (классы, интерфейсы, методы), которые могут работать с любыми типами данных, обеспечивая при этом типовую безопасность. Дженерики позволяют использовать один и тот же код для работы с разными типами данных, избегая при этом необходимости дублирования кода и обеспечивая более высокий уровень абстракции.

Реализация

Дженерики поддерживаются в различных языках программирования, таких как Java, C#, C++, и других. Ниже рассмотрим, как дженерики реализуются в Java.

В Java дженерики были введены в версии 5.0. Классы и методы могут быть параметризованными типами, что позволяет использовать их с различными типами данных.

Пример класса с использованием дженериков в Java:

```
public class Box<T> {
    private T item;
    public void set(T item) {
        this.item = item;
    }
    public T get() {
        return item;
    }
    public static void main(String[] args) {
        Box<Integer> integerBox = new Box<>();
        integerBox.set(123);
        System.out.println("Integer Value: " + integerBox.get());
        Box<String> stringBox = new Box<>();
        stringBox.set("Hello");
        System.out.println("String Value: " + stringBox.get());
    }
}
```

В этом примере Box<T> — это дженерик-класс, где T представляет тип, который будет использоваться. Мы можем создавать экземпляры этого класса с любыми типами данных, такими как Integer и String.

Они также есть в шарпе, вот пример:

```
public abstract class Entity<TPrimaryKey>
{
    public TPrimaryKey Id { get; set; }
}
```

```
public abstract class Entity : Entity<int>
{
}
```

Тут вначале задаем в Entity <TPrimaryKey> айди на любой прием, а через Entity : Entity<int> задает именно числовой тип данных.

8. Интерфейсы Comparator и Comparable. Описание, различия, примеры использования; ЕСТЬ В БИЛЕТАХ

https://javarush.com/groups/posts/6459-kofe-breyk-253-v-chem-razlichie-mezhdu-comparable-i-comparator-chto-takoe-varargs-peremennihe-a

Comparable — это интерфейс, входящий в пакет java.lang и используемый для сортировки классов на основе их естественного порядка. Интерфейс Comparable должен быть реализован в классе, который будет использоваться для сортировки. Этот класс можно сортировать на основе отдельных атрибутов, таких как идентификатор, имя, отдел и так далее.

Comparator — это интерфейс, входящий в пакет java.util, который также используется для сортировки коллекций в Java. В отличие от Comparable, интерфейс Comparator не обязательно должен быть реализован в исходном классе, его можно реализовать и в отдельном классе.

9. Коллекции типа List. Описание, представители. Механизм работы, различия реализаций; ЕСТЬ В БИЛЕТАХ

https://javarush.com/groups/posts/3986-kofe-breyk-185-podrobnoe-rukovodstvo-po-java-collection-framework

List (список) — это упорядоченный набор объектов, каждый элемент которого занимает определенную позицию в списке. Интерфейс List расширяет интерфейс Collection и добавляет в него несколько методов для работы со списками, таких как методы доступа к элементам по их положению в списке и методы поиска и сортировки списков. List может содержать повторяющиеся элементы, доступ к этим элементам можно получить по их положению в списке.

10. Коллекции типа Map. Описание, представители. Механизм работы, различия реализаций. ЕСТЬ В БИЛЕТАХ

https://javarush.com/groups/posts/2542-otvetih-na-samihe-populjarnihe-voprosih-ob-interfeyse-map

Map — это структура данных, которая содержит набор пар “ключ-значение”. По своей структуре данных напоминает словарь, поэтому ее часто так и называют. В то же время, Map является интерфейсом, и в стандартном jdk содержит основные реализации: Hashmap, LinkedHashMap, Hashtable, TreeMap. Самая используемая реализация — Hashmap, поэтому и будем ее использовать в наших примерах.

Различия реализаций:

- HashMap: Предоставляет быстрый доступ к элементам за счет хэш-таблицы, но не гарантирует порядок элементов.
- TreeMap: Сохраняет элементы в отсортированном порядке на основе ключей.
- LinkedHashMap: Сохраняет порядок вставки элементов, что позволяет итерироваться по элементам в порядке их добавления.

	HashMap	HashTable	TreeMap
Упорядоченность элементов	нет	нет	да
null в качестве значения	да	нет	да/нет
Потокобезопасность	нет	да	нет
Алгоритмическая сложность поиска элементов	O(1)	O(1)	O(log n)
Структура данных под капотом	хэш-таблица	хэш-таблица	красно-чёрное дерево

Практика ООП

(Java или Python? Или вообще C#?)

1. Описать класс «поезд», содержащий следующие закрытые поля: название пункта назначения; номер поезда (может содержать буквы и цифры); время отправления. Предусмотреть свойства для получения состояния объекта.

Описать класс «вокзал», содержащий закрытый массив поездов. Обеспечить следующие возможности: вывод информации о поезде по номеру с помощью индекса; вывод информации о поездах, отправляющихся после введенного с клавиатуры времени; перегруженную операцию сравнения, выполняющую сравнение времени отправления двух поездов; вывод информации о поездах, отправляющихся в заданный пункт назначения. Информация должна быть отсортирована по времени отправления. Написать программу, демонстрирующую все разработанные элементы классов. ЕСТЬ В БИЛЕТАХ

```
import java.util.Arrays;
import java.util.Scanner;

public class TrainStation {
    private Train[] trains;

    public TrainStation(Train[] trains) {
        this.trains = trains;
    }

    public void displayTrainInfoByIndex(int index) {
        if (index >= 0 && index < trains.length) {
            System.out.println("Train Information:");
            System.out.println("Destination: " + trains[index].getDestination());
            System.out.println("Train Number: " + trains[index].getTrainNumber());
            System.out.println("Departure Time: " + trains[index].getDepartureTime());
        } else {
            System.out.println("Train with index " + index + " not found.");
        }
    }

    public void displayTrainsAfterTime(String time) {
        System.out.println("Trains departing after " + time + ":");
        for (Train train : trains) {
            if (train.getDepartureTime().compareTo(time) > 0) {
                System.out.println("Destination: " + train.getDestination() + ", Departure Time: " +
train.getDepartureTime());
            }
        }
    }
```

```
    public void compareTrainsDepartureTime(int index1, int index2) {
        if (index1 >= 0 && index1 < trains.length && index2 >= 0 && index2 < trains.length) {
            int result =
trains[index1].getDepartureTime().compareTo(trains[index2].getDepartureTime());
            if (result < 0) {
                System.out.println("Train 1 departs before Train 2.");
            } else if (result > 0) {
                System.out.println("Train 1 departs after Train 2.");
            } else {
                System.out.println("Both trains depart at the same time.");
            }
        } else {
            System.out.println("Invalid train indices.");
        }
    }

    public void displayTrainsToDestination(String destination) {
        System.out.println("Trains going to " + destination + ":");
        Arrays.stream(trains)
                .filter(train -> train.getDestination().equalsIgnoreCase(destination))
                .sorted((t1, t2) -> t1.getDepartureTime().compareTo(t2.getDepartureTime()))
                .forEach(train -> System.out.println("Departure Time: " + train.getDepartureTime()));
    }
}

public class Train {
    private String destination; // Название пункта назначения
    private String trainNumber; // Номер поезда
    private String departureTime; // Время отправления

    // Конструктор класса
    public Train(String destination, String trainNumber, String departureTime) {
        this.destination = destination;
        this.trainNumber = trainNumber;
        this.departureTime = departureTime;
    }

    // Методы для получения состояния объекта
    public String getDestination() {
        return destination;
    }

    public String getTrainNumber() {
        return trainNumber;
    }

    public String getDepartureTime() {
        return departureTime;
    }
}

    public static void main(String[] args) {
        Train train1 = new Train("Moscow", "A123", "08:00");
        Train train2 = new Train("St. Petersburg", "B456", "09:30");
        Train train3 = new Train("Kazan", "C789", "10:45");

        Train[] trains = {train1, train2, train3};
```

```
        TrainStation station = new TrainStation(trains);

        Scanner scanner = new Scanner(System.in);
        System.out.print("Enter the time to display trains departing after: ");
        String inputTime = scanner.next();
        station.displayTrainsAfterTime(inputTime);

        station.compareTrainsDepartureTime(0, 1);

        System.out.print("Enter the destination to display trains going to: ");
        String inputDestination = scanner.next();
        station.displayTrainsToDestination(inputDestination);

        scanner.close();
    }
```

Для питонистов:

```python
class Train:
    def __init__(self, destination, train_number, departure_time):
        self._destination = destination
        self._train_number = train_number
        self._departure_time = departure_time

    @property
    def destination(self):
        return self._destination

    @property
    def train_number(self):
        return self._train_number

    @property
    def departure_time(self):
        return self._departure_time

class Station:
    def __init__(self):
        self.trains = []

    def add_train(self, train):
        self.trains.append(train)

    def get_train_by_number(self, number):
        for train in self.trains:
            if train.train_number == number:
                return (train.train_number,train.destination, train.departure_time)
        return None

    def get_trains_after_time(self, time):
        return [(train.train_number,train.destination, train.departure_time) for
train in self.trains if train.departure_time > time]
```

```
        def compare_departure_times(self, train1, train2):
            if train1.departure_time < train2.departure_time:
                return f"{train1.departure_time}"+" < "+f"{train2.departure_time}"
            else: return f"{train1.departure_time}"+" > "+f"{train2.departure_time}"

        def get_trains_to_destination(self, destination):
            trains_list = []
            # sorted_list = sorted([train for train in self.trains if train.destination
== destination], key=lambda x: x.departure_time)
            sorted_list = sorted(self.trains, key=lambda x: x.departure_time)
            for i in sorted_list:
                if i.destination == destination: trains_list.append(i)
             return [(train.train_number,train.destination, train.departure_time) for
train in trains_list]

    # Creating instances of Train
    train1 = Train("Moscow", "123A", "08:00")
    train2 = Train("Saint Petersburg", "456B", "09:30")
    train3 = Train("Kiev", "789C", "10:15")
    train4 = Train("Moscow", "123B", "09:00")

    # Creating an instance of Station and adding trains to it
    station = Station()
    station.add_train(train1)
    station.add_train(train2)
    station.add_train(train3)
    station.add_train(train4)

    print(station.trains)

    # Demonstrating functionality
    while True:
        cmd = ""
        print('''
              1: вывод информации о поезде по номеру с помощью индекса;
              2: вывод информации о поездах, отправляющихся после введенного с
клавиатуры времени;
              3: вывод информации о сравнение времени отправления двух поездов;
              4: вывод информации о поездах, отправляющихся в заданный пункт
назначения;
              0: выход
              '''
        )
        cmd = input('введите команду: ')
        match cmd:
            case "1":
                print(station.get_train_by_number(input("Номер поезда: ")))
            case "2":
```

```
                print(station.get_trains_after_time(input("Время(чч:мм): ")))
            case "3":
                print(station.compare_departure_times(train1, train2))
            case "4":
                print(station.get_trains_to_destination(input("Место назначения: ")))
            case "0": break
            case _:
                print("Code not found")

print(station.get_train_by_number("123A"))
print(station.get_trains_after_time("09:00"))
print(station.compare_departure_times(train1, train2))
print(station.get_trains_to_destination("Moscow"))
```

2. Описать класс «товар», содержащий следующие закрытые поля: название товара; название магазина, в котором продается товар; стоимость товара в рублях. Предусмотреть свойства для получения состояния объекта.

Описать класс «склад», содержащий закрытый массив товаров. Обеспечить следующие возможности: вывод информации о товаре по номеру с помощью индекса; вывод на экран информации о товаре, название которого введено с клавиатуры; если таких товаров нет, выдать соответствующее сообщение; сортировку товаров по названию магазина, по наименованию и по цене; перегруженную операцию сложения товаров, выполняющую сложение их цен. Написать программу, демонстрирующую все разработанные элементы классов. ЕСТЬ В БИЛЕТАХ

```
using System;
using System.Linq;

namespace Tovari
{
    class Program
    {
        static void Main(string[] args)
        {
            Sclad sclad = new Sclad(new Products[] {
                new Products("Молоко", "Пятёрочка", 59),
                new Products("Колбаса", "Монетка", 209),
                new Products("Хлеб", "Магнит", 49)
            });

            Console.WriteLine("Вывод 2 элемента\n");
            Console.WriteLine(sclad.Print(2));
            Console.Write("Введите название товара: ");
            Console.WriteLine(sclad.Print(Console.ReadLine()));
            Console.WriteLine("--------------------------------");
            Console.WriteLine("Сортировка по названию товаров\n");
            foreach (Products product in sclad.Sort_name())
                Console.WriteLine(product.Sostoiyanie);
            Console.WriteLine("--------------------------------");
```

```
            Console.WriteLine("Сортировка по названию магазинов\n");
            foreach (Products product in sclad.Sort_name_magazin())
                Console.WriteLine(product.Sostoiyanie);
            Console.WriteLine("--------------------------------");
            Console.WriteLine("Сортировка по цене товаров\n");
            foreach (Products product in sclad.Sort_price())
                Console.WriteLine(product.Sostoiyanie);
            Console.WriteLine("--------------------------------");
            Console.WriteLine("Общая сумма = " + sclad.Summa());
            Console.WriteLine("Сумма 2 и 3 товара = "+ sclad.Summa(1,2));
        }
    }

    class Products
    {
        string name; //название товара
        string name_magazin; //название магазина
        int price; // цена товара

        public string Name => name;
        public string Name_magazin => name_magazin;
        public int Price => price;

        public string Sostoiyanie=>  // свойство сотояние обьекта
            "Товар: " + name+"\n"+
            "Магазин: " + name_magazin + "\n" +
            "Цена: " + price+"р"+"\n";
        public Products(string name, string name_magazin, int price)
        {
            this.name = name;
            this.name_magazin = name_magazin;
            this.price = price;

        }
    }

    class Sclad
    {
        Products[] products;

        public Sclad(Products[] products)
        {
            this.products= products;
        }

        public string Print(int index)
        {
            try
            {
                return products[index].Sostoiyanie;
```

```
            }
            catch
            {
                return "Таково товара нет";
            }

        }

        public string Print(string name)
        {
            Products inf = Array.Find(products, product => product.Name ==
name);
            return inf!=null? inf.Sostoiyanie: "Таково товара нет";
        }

        public Products[] Sort_name()
        {
            return products.OrderBy(product => product.Name).ToArray();
        }

        public Products[] Sort_name_magazin()
        {
            return products.OrderBy(product =>
product.Name_magazin).ToArray();
        }

        public Products[] Sort_price()
        {
            return products.OrderBy(product => product.Price).ToArray();
        }

        public int Summa()
        {
            return products.Sum(product=> product.Price);
        }

        public int Summa(int index1, int index2)
        {
            try
            {
                return products[index1].Price+products[index2].Price;
            }
            catch
            {
                return 0;
            }

        }
    }
}
```

для питонюг:

```
class Product:
    def __init__(self, store_name, product_name, cost):
        self.__store_name = store_name
        self.__product_name = product_name
        self.__cost = cost

    @property
    def store_name(self):
        return self.__store_name

    @property
    def product_name(self):
        return self.__product_name

    @property
    def cost(self):
        return self.__cost

class Warehouse:
    def __init__(self):
        self.__products = []

    def add_product(self, product):
        self.__products.append(product)

    def show_product_by_index(self, index):
        if 0 <= index < len(self.__products):
            print(f"Store Name: {self.__products[index].store_name}")
            print(f"Product Name: {self.__products[index].product_name}")
            print(f"Cost: {self.__products[index].cost} rub.")
        else:
            print("No such product exists.")

    def find_product_by_name(self, name):
        for product in self.__products:
            if product.product_name == name:
                print(f"Store Name: {product.store_name}")
                print(f"Product Name: {product.product_name}")
                print(f"Cost: {product.cost} rub.")
                break
        else:
            print("No such product exists.")

    def sort_by_store(self):
        self.__products.sort(key=lambda x: x.store_name)
        for i in self.__products:
            print(i.store_name, i.product_name,i.cost)
```

```
        def sort_by_name(self):
            self.__products.sort(key=lambda x: x.product_name)
            for i in self.__products:
                print(i.store_name, i.product_name,i.cost)
        def sort_by_cost(self):
            self.__products.sort(key=lambda x: x.cost)
            for i in self.__products:
                print(i.store_name, i.product_name,i.cost)

        def calculate_total_cost(self):
            total_cost = sum([product.cost for product in self.__products])
            print(f"Total Cost of All Products: {total_cost} rub.")


    # Create products
    store1 = Product("Store 1", "Product 1", 100)
    store2 = Product("Store 2", "Product 2", 200)
    store3 = Product("Store 3", "Product 3", 300)

    # Add products to warehouse
    warehouse = Warehouse()
    warehouse.add_product(store1)
    warehouse.add_product(store2)
    warehouse.add_product(store3)

    while True:
        cmd = ""
        print('''
              1: вывод информации о товаре по номеру с помощью индекса;
                  2: вывод на экран информации о товаре, название которого введено с клавиатуры;
                  3: вывод информации о сортировке товаров по названию магазина, по наименованию и по цене;
              4: вывод информации о сумме цен товаров;
              0: выход
              '''
        )
        cmd = input('введите команду: ')
        match cmd:
            case "1":
                warehouse.show_product_by_index(input("Номер товара: "))
            case "2":
                warehouse.find_product_by_name(input("Название товара: "))
            case "3":
                print("Сортировка по названию магазина")
                warehouse.sort_by_store()
                print("Сортировка по наименованию")
```

```
                warehouse.sort_by_name()
                print("Сортировка по цене")
                warehouse.sort_by_cost()
            case "4":
                warehouse.calculate_total_cost()
            case "0": break
            case _:
                print("Code not found")
```

Вопросы БД (Базы данных)

1 Необходимость проектирования баз данных, цели проектирования, этапы проектирования. **ЕСТЬ В БИЛЕТАХ**

Хорошо спроектированная база данных необходима для обеспечения согласованности информации, устранения избыточных данных, эффективного выполнения запросов и повышения производительности базы данных. Методологический подход к проектированию базы данных сэкономит вам время на этапе разработки базы данных.

Основная цель проектирования базы данных - это сокращение избыточности хранимых данных, а следовательно, экономия объема используемой памяти, уменьшение затрат на многократные операции обновления избыточных копий и устранение возможности возникновения противоречий из-за хранения в разных местах сведений об одном и том же объекте

1. Анализ требований: В этом этапе проектировщики баз данных собирают и анализируют требования к данным и функциональности системы, определяют бизнес-правила и основные потребности пользователей.
2. Логическое проектирование: На этом этапе создается концептуальная модель базы данных, которая описывает структуру данных независимо от конкретной СУБД. Используются сущности, атрибуты и связи между ними.
3. Физическое проектирование: Здесь концептуальная модель преобразуется в физическую модель, которая определяет способ хранения данных в базе данных. Включает выбор типов данных, индексов, оптимизацию запросов и другие аспекты, связанные с производительностью и эффективностью.
4. Реализация и тестирование: На этом этапе создается база данных с использованием выбранной СУБД, загружаются данные, создаются таблицы, индексы и другие структуры. После этого проводятся тесты для проверки правильности работы базы данных.
5. Эксплуатация и сопровождение: После внедрения базы данных она подвергается эксплуатации, администрированию, резервному копированию данных и обновлениям. В случае необходимости вносятся изменения и улучшения в базу данных.

2. Основные понятия реляционной базы данных: сущность, атрибут, ключ, запись, связь. **ЕСТЬ В БИЛЕТАХ**

1. Сущность (Entity): Объект, который имеет определенные свойства и может быть описан с помощью атрибутов. Сущности могут быть объектами реального мира, такими как люди, организации, или абстрактными, такими как заказы или товары.
2. Атрибут (Attribute): Характеристика или свойство сущности, которая может иметь определенное значение. Атрибуты могут быть числовыми, текстовыми, дате или логическими.
3. Ключ (Key): Уникальное значение, которое идентифицирует запись в базе данных. Ключи могут быть простыми (например, идентификатор пользователя) или сложными (например, сочетание нескольких атрибутов).
4. Запись (Tuple): Сетка атрибутов, которая описывает конкретное состояние сущности. Записи могут быть добавлены, изменены или удалены в базе данных.
5. Связь (Relationship): Отношение между сущностями, которое описывает, как они связаны друг с другом. Связи могут быть один к одному, один ко многим или многим к многим.

3. Виды моделей данных. Иерархическая и сетевая модели. Достоинства и недостатки. **ЕСТЬ В БИЛЕТАХ**

Иерархическая и сетевая модели данных являются двумя из первых моделей данных, используемых в базах данных. Вот их основные характеристики:

Иерархическая модель данных:

Достоинства:

- Простота структуры: Данные организованы в виде древовидной структуры с родительскими и дочерними узлами.
- Эффективность при работе с иерархическими данными: Подходит для представления связей типа один ко многим.

Недостатки:

- Жесткость структуры: Требует заранее определенной структуры, что затрудняет изменения.
- Ограниченность: Неудобно представлять сложные связи между данными.

Сетевая модель данных:

Достоинства:

- Гибкость: Позволяет представлять сложные связи между данными.
- Эффективность при работе с многими ко многим: Подходит для представления сложных связей.

Недостатки:

- Сложность: Требует более сложной структуры и понимания для работы с данными.
- Сложности при обновлении: Может быть сложно поддерживать целостность данных при изменениях.

Иерархическая модель данных подходит для простых структур данных с четко определенными связями, в то время как сетевая модель данных более гибка и позволяет представлять сложные связи, но требует более сложной структуры и обработки.

4. Виды моделей данных. Реляционная модель данных, основные понятия и элементы. ЕСТЬ В БИЛЕТАХ

Реляционная модель данных является одной из основных моделей данных в базах данных. Основные понятия и элементы реляционной модели данных включают:

Таблицы: В реляционной модели данные хранятся в виде таблиц, где каждая строка представляет собой запись, а каждый столбец - атрибут или поле.

Отношения: Отношения в реляционной модели представляют связи между таблицами на основе общих ключей. Они обеспечивают связь между данными из разных таблиц.

Ключи: В реляционной модели ключи играют важную роль. Ключи могут быть первичными (Primary Key), уникальными и внешними (Foreign Key), обеспечивая уникальность и связи между данными.

Запросы: Для извлечения данных из реляционной базы данных используются SQL-запросы. Они позволяют выполнять операции выборки, вставки, обновления и удаления данных.

Нормализация: Процесс нормализации в реляционной модели помогает устранить избыточность данных и обеспечить целостность и эффективность хранения данных.

Целостность данных: Реляционная модель обеспечивает целостность данных с помощью ограничений целостности, которые гарантируют правильность и согласованность данных.

Реляционная модель данных является широко используемой и эффективной для хранения и управления данными в базах данных, обеспечивая структурированное хранение и обработку информации.

5. Операции реляционной алгебры применительно к базам данных: проекция, выборка, соединение, объединение, пересечение, вычитание, умножение. ЕСТЬ В БИЛЕТАХ

Проекция (Projection): Операция проекции выбирает определенные атрибуты из таблицы, оставляя остальные атрибуты не измененными.

Выборка (Selection): Операция выборки выбирает записи из таблицы, удовлетворяющие определенным условиям.

Соединение (Join): Операция соединения объединяет записи из двух или более таблиц на основе общих ключей.

Объединение (Union): Операция объединения объединяет записи из двух или более таблиц, не учитывая дубликаты.

Пересечение (Intersection): Операция пересечения выбирает записи, которые есть в обеих таблицах.

Вычитание (Difference): Операция вычитания выбирает записи из первой таблицы, которые не есть в второй таблице.

Умножение (Cartesian Product): Операция умножения создает новую таблицу, содержащую все возможные комбинации записей из двух или более таблиц.

6. Понятие ключа отношения. Виды ключей. Правила выбора ключа. ЕСТЬ В БИЛЕТАХ

1. Первичный ключ (Primary Key): Уникальный ключ, который однозначно идентифицирует каждую запись в таблице. Значения первичного ключа не могут быть пустыми и не могут повторяться.
2. Уникальный ключ (Unique Key): Ключ, который гарантирует уникальность значений, но может содержать пустые значения. Позволяет исключить дубликаты, но не обязательно идентифицирует записи.
3. Внешний ключ (Foreign Key): Ключ, который устанавливает связь между двумя таблицами. Значение внешнего ключа связано с значением первичного ключа в другой таблице.

Правила выбора ключа включают:

- Ключ должен быть уникальным и однозначно идентифицировать записи.
- Первичный ключ должен быть простым и стабильным, не подверженным изменениям.
- Использование естественных ключей (например, идентификатор клиента) или создание искусственных ключей (например, автоинкрементный идентификатор).
- При использовании составного ключа, убедитесь, что комбинация атрибутов уникальна и не содержит пустых значений.

7. Организация параллельной работы устройств ввода-вывода и процессора. Виды зависимостей. ЕСТЬ В БИЛЕТАХ

Функциональная зависимость - это свойство отношения между атрибутами в базе данных, которое означает, что значение одного или более атрибутов определяет значение других атрибутов в этом отношении. Другими словами, функциональная зависимость указывает, что при изменении значения одного атрибута в базе данных, изменятся и значения других атрибутов.

Виды функциональных зависимостей:

1. Простая функциональная зависимость - это зависимость, где один атрибут определяет значение другого атрибута в отношении.
2. Сложная функциональная зависимость - это зависимость, где значение одного или нескольких атрибутов определяет значение другого атрибута.
3. Транзитивная функциональная зависимость - это зависимость, где значение одного атрибута определяет значение другого атрибута через третий атрибут.
4. Нетривиальная функциональная зависимость - это зависимость, где значение одного или нескольких атрибутов определяет значение другого атрибута, и эта зависимость не может быть выражена через другие функциональные зависимости.

Примеры использования различных видов функциональных зависимостей в базе данных:

1. Простая функциональная зависимость: в базе данных организации есть таблица "сотрудники", в которой один из атрибутов - "адрес". Значение атрибута "адрес" зависит от значения атрибута "сотрудник", поэтому мы можем говорить о простой функциональной зависимости между этими двумя атрибутами.
2. Сложная функциональная зависимость: в базе данных онлайн-магазина есть таблица "заказы", в которой один из атрибутов - "сумма заказа". Значение атрибута "сумма заказа" зависит от значений атрибутов "стоимость товара" и "количество товара" для каждого отдельного заказа, поэтому мы можем говорить о сложной функциональной зависимости между этими атрибутами.
3. Транзитивная функциональная зависимость: в базе данных университета есть таблица "студенты", в которой один из атрибутов - "факультет". Значение атрибута

"факультет" можно определить через атрибут "кафедра", который, в свою очередь, зависит от атрибута "направление", поэтому мы можем говорить о транзитивной функциональной зависимости между этими тремя атрибутами.

4. Нетривиальная функциональная зависимость: в базе данных банка есть таблица "клиенты", в которой атрибут "номер счета" зависит от атрибутов "тип счета" и "филиал банка". Таким образом, мы не можем выразить эту зависимость через другие функциональные зависимости, и поэтому мы говорим о нетривиальной функциональной зависимости между этими атрибутами.

8. Понятие нормализации базы данных. Нормальные формы. Требования 1НФ, 2НФ и 3НФ. ЕСТЬ В БИЛЕТАХ
(https://habr.com/ru/articles/254773/)

Нормализация базы данных - это процесс структурирования данных в базе данных в соответствии с набором правил, называемых нормальными формами. Цель нормализации - устранение избыточности данных, обеспечение целостности и упрощение структуры базы данных

Первая нормальная форма (1НФ):

- Все поля содержат атомарные (неделимые) значения
- Все записи уникальны (нет дубликатов)
- Каждая запись содержит одинаковое количество полей

Например, есть таблица «Автомобили»:

Фирма	Модели
BMW	M5, X5M, M1
Nissan	GT-R

Нарушение нормализации 1НФ происходит в моделях BMW, т.к. в одной ячейке содержится список из 3 элементов: M5, X5M, M1, т.е. он не является атомарным. Преобразуем таблицу к 1НФ:

Фирма	Модели
BMW	M5
BMW	X5M
BMW	M1
Nissan	GT-R

Вторая нормальная форма (2НФ)

- Она находится в 1НФ
- Все не ключевые поля полностью зависят от первичного ключа

Модель	Фирма	Цена	Скидка
M5	BMW	5500000	5%
X5M	BMW	6000000	5%
M1	BMW	2500000	5%
GT-R	Nissan	5000000	10%

Таблица находится в первой нормальной форме, но не во второй. Цена машины зависит от модели и фирмы. Скидка зависят от фирмы, то есть зависимость от первичного ключа неполная. Исправляется это путем декомпозиции на два отношения, в которых не ключевые атрибуты зависят от ПК.

Модель	Фирма	Цена
M5	BMW	5500000
X5M	BMW	6000000
M1	BMW	2500000
GT-R	Nissan	5000000

Фирма	Скидка
BMW	5%
Nissan	10%

Третья нормальная форма (3НФ)

- Она находится во 2НФ
- Нет транзитивных зависимостей, т.е. не ключевые поля не зависят друг от друга

Модель	Магазин	Телефон
BMW	Риал-авто	87-33-98
Audi	Риал-авто	87-33-98
Nissan	Некст-Авто	94-54-12

Таблица находится во 2НФ, но не в 3НФ.
В отношении атрибут «Модель» является первичным ключом. Личных телефонов у автомобилей нет, и телефон зависит исключительно от магазина.
Таким образом, в отношении существуют следующие функциональные зависимости: Модель → Магазин, Магазин → Телефон, Модель → Телефон.
Зависимость Модель → Телефон является транзитивной, следовательно, отношение не находится в 3НФ.
В результате разделения исходного отношения получаются два отношения, находящиеся в 3НФ:

Магазин	Телефон
Риал-авто	87-33-98
Некст-Авто	94-54-12

Модель	Магазин
BMW	Риал-авто
Audi	Риал-авто
Nissan	Некст-Авто

9. Понятие связи между сущностями. Характеристики связи. **ЕСТЬ В БИЛЕТАХ**

Связь между сущностями в базе данных - это ассоциация между двумя или более сущностями, которая определяет, как информация из этих сущностей будет связана друг с другом. Связи используются для определения отношений между сущностями в базе данных и для обеспечения целостности и согласованности данных.

Характеристики связи между сущностями:

1. Тип связи: определяет, как информация из одной сущности связана с информацией из другой сущности. Например, отношение «один к одному», «один ко многим», «многие ко многим».
2. Кардинальность связи: определяет, сколько записей из одной сущности связано с записями из другой сущности. Например, если у нас есть отношение «один ко многим» между таблицами «клиенты» и «заказы», то кардинальность связи для таблицы «заказы» будет «много», а для таблицы «клиенты» - «один».
3. Направленность связи: определяет, можно ли переходить от одной сущности к другой в обоих направлениях. Если связь между двумя таблицами является направленной, то мы можем переходить от одной таблицы к другой только в одном направлении.
4. Взаимосвязь: определяет, зависит ли связь между сущностями от существования других связанных сущностей или от состояния других таблиц в БД.
5. Атрибуты связи: некоторые связи могут иметь атрибуты, которые относятся к связи между двумя сущностями, а не к любой из сущностей самостоятельно.

10. Приведение связи типа «многие-ко-многим» к типу «один-ко многим». **ЕСТЬ В БИЛЕТАХ**

Связи типа "многие-ко-многим" часто представляются в базах данных как промежуточная таблица, которая содержит связи между двумя связанными таблицами. Перевод связи типа "многие-ко-многим" к типу "один-ко-многим" можно выполнить следующим образом:

1. Создание новой таблицы: необходимо создать новую таблицу, которая будет представлять одну из связанных таблиц в связи типа "многие-ко-многим". В эту таблицу будут добавлены поля, которые ранее были связаны с промежуточной таблицей.
2. Добавление внешнего ключа: добавляем внешний ключ к новой таблице, указывающий на первичный ключ таблицы, с которой она была связана ранее.
3. Обновление связи таблиц: обновляем связь между таблицами. Теперь вместо связи "многие-ко-многим" у нас есть две связи "один-ко-многим".
4. Удаление промежуточной таблицы: после выполнения первых трех шагов мы можем удалить промежуточную таблицу, которая больше не нужна.

Например, если у нас есть таблицы "книги" и "авторы", и мы имеем связь между ними типа "многие-ко-многим", то мы можем создать новую таблицу "авторские права", которая будет содержать поля из обеих таблиц (например, id книги и id автора). Далее, мы добавляем внешний ключ к таблице "книги" и обновляем связи между таблицами, таким образом, что теперь у нас есть связь "один-ко-многим" между таблицами "авторы" и "авторские права", а также между таблицами "книги" и "авторские права". Далее, мы можем удалить промежуточную таблицу "авторство".

~~**11. СУБД Access. Характеристики, применение. Состав базы данных в СУБД Access.** **НЕТ В БИЛЕТАХ**~~

~~Microsoft Access - это реляционная СУБД, разработанная корпорацией Microsoft. Она была создана для создания, хранения, анализа и управления базами данных. Access является одним из наиболее распространенных инструментов для создания баз данных на компьютерах с Windows.~~

~~Характеристики и применение СУБД Access:~~

~~1. Access имеет простой и легко понятный интерфейс пользователя, что делает ее привлекательной для начинающих пользователей.~~

~~2. Access интегрируется с другими приложениями Microsoft Office, позволяя пользователям обмениваться данными и использовать данные из других приложений.~~

~~3. Access поддерживает большой набор типов данных, включая текстовые поля, даты, числа, денежные единицы, фотографии и другие.~~

~~4. Access содержит средства для генерации отчетов и форм, что упрощает создание и отображение данных для конечных пользователей.~~

~~5. Access поддерживает SQL-запросы и предоставляет инструменты для модификации данных и создания связей между таблицами.~~

~~6. Access может использоваться для создания баз данных различных типов и размеров, от малых локальных баз данных до крупных корпоративных систем.~~

~~Состав базы данных в СУБД Access:~~

~~1. Таблицы: хранят данные в виде строк и столбцов, как в Excel.~~

~~2. Отношения: определяют связи между таблицами и работают на основе ключей.~~

~~3. Запросы: позволяют отбирать и фильтровать данные из нескольких таблиц.~~

~~4. Формы: дают возможность пользователю вводить, изменять и просматривать данные в базе данных.~~

~~5. Отчеты: позволяют создавать различные отчеты на основе данных из таблиц в базе данных.~~

~~6. Макросы: представляют собой автоматизированные действия, которые можно выполнить при определенных условиях.~~

~~7. Модули: содержат программный код на языке VBA, который дает возможность создавать более сложные функции и процедуры для работы с базой данных.~~

12. Предложение SELECT языка SQL и его элементы. **ЕСТЬ В БИЛЕТАХ**

(ответы от Perplexity)

Предложение SELECT в языке SQL используется для выборки данных из базы данных.

Ключевое слово FROM, за которым следует название таблицы, из которой нужно выбрать данные

```
SELECT * FROM users
```

Предложение WHERE позволяет добавить условие фильтрации выбираемых записей

```
SELECT * FROM users WHERE age >= 18
```

Ключевое слово DISTINCT позволяет вернуть только уникальные значения из выбранных полей

```
SELECT DISTINCT country FROM users
```

Предложение ORDER BY позволяет отсортировать результаты по одному или нескольким полям в порядке возрастания (ASC) или убывания (DESC)

```
SELECT * FROM users ORDER BY age DESC, name ASC
```

Предложение GROUP BY используется вместе с агрегатными функциями (SUM, AVG, COUNT и т.д.) для группировки результатов по одному или нескольким полям

```
SELECT country, COUNT(*) AS total
FROM users
GROUP BY country
```

HAVING используется для фильтрации результатов, сгруппированных с помощью GROUP BY, по условию

```
SELECT country, COUNT(*) AS total
FROM users
GROUP BY country
HAVING COUNT(*) > 100
```

13. Виды запросов к базе данных и их реализация на языке SQL. **ЕСТЬ В БИЛЕТАХ**

(ответы от Perplexity)

Оператор SELECT используется для выборки данных из таблиц базы данных

Оператор INSERT используется для добавления новых записей в таблицу

```
INSERT INTO users (first_name, last_name, age)
VALUES ('Иван', 'Петров', 25)
```

Оператор UPDATE используется для изменения существующих записей в таблице

```
UPDATE users
SET age = age + 1
WHERE id = 1
```

Оператор DELETE используется для удаления записей из таблицы

```
DELETE FROM users
WHERE id = 1
```

Оператор CREATE TABLE используется для создания новых таблиц в базе данных

```
CREATE TABLE products (
  id INT PRIMARY KEY,
  name VARCHAR(50),
  price DECIMAL(10,2)
)
```

14. Подзапросы в языке SQL. Назначение, виды, порядок выполнения. ЕСТЬ В БИЛЕТАХ

(ответы от Perplexity)

В языке SQL подзапросы (subqueries) - это запросы, вложенные в другой запрос. Они используются для получения данных из множества таблиц или для фильтрации результатов основного запроса

Подзапросы позволяют:

- Получать данные из нескольких таблиц
- Фильтровать результаты основного запроса
- Использовать результаты подзапроса в операторах сравнения

Подзапросы бывают двух типов:

- Коррелированные подзапросы: значение подзапроса зависит от значения, производимого внешним оператором select, который содержит этот подзапрос
- Некоррелированные подзапросы: значение подзапроса не зависит от внешнего оператора select

Выполнение подзапроса происходит в следующем порядке:

1. Выполнение подзапроса
2. Получение результатов подзапроса
3. Использование результатов подзапроса в основном запросе

ПРИМЕР. Выдать информацию о поставщиках, рейтинг которых выше рейтинга хотя бы одного парижского поставщика

```
SELECT номер_поставщика
FROM S
WHERE город = (SELECT город FROM S WHERE номер_поставщика = 'S1')
```

15. Запросы с параметром. Достоинства и недостатки. ЕСТЬ В БИЛЕТАХ

Пример использования параметра в запросе

Допустим, нужно выбрать все бронирования для самого дорогого на данный момент жилого помещения. Можно использовать подзапрос для получения id самого дорогого номера, а затем использовать его в качестве параметра

```
SELECT *
FROM Reservations
WHERE room_id = (
  SELECT id
  FROM Rooms
  ORDER BY price DESC
  LIMIT 1
)
```

Достоинства запросов с параметром:

Гибкость: параметры позволяют адаптировать запрос под конкретные условия выборки данных

Безопасность: использование параметров вместо конкатенации строк предотвращает SQL-инъекции

Производительность: параметры кэшируются базой данных, что ускоряет последующие выполнения запроса

Недостатки запросов с параметром:

Ограничения на использование: не рекомендуется использовать подзапросы и соединения в параметрах, так как это может замедлить работу запроса

Необходимость предварительной обработки: значения параметров нужно предварительно подготовить, например, экранировать специальные символы

Сложность отладки: при ошибках в запросе с параметрами сложнее понять, какое значение параметра вызвало ошибку

16. Целостность и сохранность баз данных. Виды целостности. ЕСТЬ В БИЛЕТАХ

(ответы от Perplexity)

Целостность данных в базе данных означает точность, полноту и надежность данных, а также поддержание связей между записями в связанных таблицах для защиты от случайного удаления или изменения данных

Виды целостности данных:

- Целостность полей: гарантирует правильность и точность данных в полях таблиц
- Целостность сущностей: обеспечивает уникальность и правильность идентификации сущностей в базе данных
- Целостность ссылочная: поддерживает связи между таблицами, чтобы не было ссылок на несуществующие записи
- Целостность доменов: определяет правила для значений, которые могут быть введены в определенные поля
- Целостность пользовательская: позволяет создавать пользовательские правила для обеспечения целостности данных

Практика БД

1. Разработайте базу данных для предприятия связи, которая содержит следующие таблицы (не менее 3-х записей в таблицах):

-Сотрудники (табельный номер, ФИО, отдел, код должности, пол, дата рождения, стаж, семейное положение (Б-брак, Х- холост, Р- разведен), дети).

- Штатное расписание (код должности, должность, оклад).

Установите связь между таблицами.

Составьте запрос «Зарплата», в котором зарплата сотрудников вычисляется по формуле «оклад+премия». Премия зависит от стажа, если стаж <=5 лет премия равна 50 % от оклада; если стаж больше 5 лет премия равна 100 % от оклада.

Составьте запрос «Отделы»: количество сотрудников, максимальная зарплата, минимальная зарплата, общее количество детей в отделе. ЕСТЬ В БИЛЕТАХ

Андрей Ми6: Не совсем понятно, что тут конкретно требуется, поэтому так это будет выглядеть в виде SQL-запросов (запросы выполняем по очереди):

```
-- Создание таблицы "Сотрудники"
CREATE TABLE IF NOT EXISTS Сотрудники (
    табельный_номер INT PRIMARY KEY,
    ФИО VARCHAR(50),
```

```
        отдел VARCHAR(50),
        код_должности INT,
        пол VARCHAR(1),
        дата_рождения DATE,
        стаж INT,
        семейное_положение VARCHAR(1),
        дети INT
    );

    -- Вставка данных в таблицу "Сотрудники"
    INSERT INTO Сотрудники VALUES
    (1, 'Иванов Иван Иванович', 'Отдел разработки', 1, 'М',
'1990-01-01', 7, 'Б', 2),
    (2, 'Петров Петр Петрович', 'Отдел продаж', 2, 'М', '1985-05-10',
3, 'Х', 0),
    (3, 'Сидорова Елена Петровна', 'Отдел маркетинга', 3, 'Ж',
'1988-12-15', 6, 'Р', 1);

    -- Создание таблицы "Штатное расписание"
    CREATE TABLE IF NOT EXISTS Штатное_расписание (
        код_должности INT PRIMARY KEY,
        должность VARCHAR(50),
        оклад DECIMAL(10, 2)
    );
    -- Вставка данных в таблицу "Штатное расписание"
    INSERT INTO Штатное_расписание VALUES
    (1, 'Разработчик', 50000.00),
    (2, 'Менеджер по продажам', 45000.00),
    (3, 'Маркетолог', 48000.00);

    -- Установление связи между таблицами
    ALTER TABLE Сотрудники ADD FOREIGN KEY (код_должности) REFERENCES
Штатное_расписание (код_должности);

    -- Запрос "Зарплата"
    SELECT s.табельный_номер, s.ФИО, s.отдел, s.стаж,
        CASE
            WHEN s.стаж <= 5 THEN Штатное_расписание.оклад + (0.5 *
Штатное_расписание.оклад)
            ELSE Штатное_расписание.оклад + Штатное_расписание.оклад
        END AS Зарплата
    FROM Сотрудники s
    JOIN Штатное_расписание ON s.код_должности =
Штатное_расписание.код_должности;

    -- Запрос "Отделы"
    SELECT отдел, COUNT(табельный_номер) AS Количество_сотрудников,
        MAX(Штатное_расписание.оклад + CASE
                                           WHEN стаж <= 5 THEN 0.5 *
Штатное_расписание.оклад
                                           ELSE
Штатное_расписание.оклад
                                       END) AS Максимальная_зарплата,
        MIN(Штатное_расписание.оклад + CASE
                                           WHEN стаж <= 5 THEN 0.5 *
Штатное_расписание.оклад
                                           ELSE
Штатное_расписание.оклад
                                       END) AS Минимальная_зарплата,
        SUM(дети) AS Общее_количество_детей
```

```
    FROM Сотрудники
    JOIN Штатное_расписание ON Сотрудники.код_должности =
Штатное_расписание.код_должности
    GROUP BY отдел;
```

2. ЕСТЬ В БИЛЕТАХ

Создайте и заполните базу данных сотрудников предприятия связи: - Сотрудники (табельный номер, ФИО, отдел, код должности, пол, дата рождения, стаж, семейное положение (Б-брак, Х- холост, Р- разведен), дети).

- Штатное расписание (код должности, должность, оклад),

Установите связь между таблицами.

Составьте запросы для выборки информации:

- о сотрудниках: подразделение, ФИО, зарплата в рублях, зарплата в долларах (вычислить), отсортировав по фамилии (зарплата сотрудников вычисляется по формуле «оклад + премия», где премия равна 80 % от оклада.);

- по каждому подразделению вычислить: среднюю заработную плату, количество женщин и детей.

Андрей Ми6: таблицы те же, запросы другие:

```
    -- Создание таблиц
    CREATE TABLE IF NOT EXISTS Сотрудники (
        табельный_номер INT PRIMARY KEY,
        ФИО VARCHAR(255),
        отдел VARCHAR(255),
        код_должности INT,
        пол CHAR(1),
        дата_рождения DATE,
        стаж INT,
         семейное_положение CHAR(1) CHECK (семейное_положение IN ('Б',
'X', 'P')),
        дети INT
    );

    CREATE TABLE IF NOT EXISTS Штатное_расписание (
        код_должности INT PRIMARY KEY,
        должность VARCHAR(255),
        оклад DECIMAL(10, 2)
    );
    --Установление связи
    ALTER TABLE Сотрудники ADD FOREIGN KEY (код_должности) REFERENCES
Штатное_расписание(код_должности);
    --Заполнение таблиц
    -- Вставка данных в таблицу Штатное_расписание
    INSERT INTO Штатное_расписание (код_должности, должность, оклад)
VALUES
    (1, 'Инженер', 50000),
    (2, 'Менеджер', 60000),
    (3, 'Администратор', 40000);

    -- Вставка данных в таблицу Сотрудники
    INSERT INTO Сотрудники (табельный_номер, ФИО, отдел, код_должности,
пол, дата_рождения, стаж, семейное_положение, дети) VALUES
    (1, 'Иванов Иван Иванович', 'Технический отдел', 1, 'М',
'1980-01-01', 7, 'Б', 2),
    (2, 'Петрова Мария Петровна', 'Финансовый отдел', 2, 'Ж',
'1990-02-02', 6, 'Х', 1),
    (3, 'Сидоров Сергей Сидорович', 'Административный отдел', 3, 'М',
'1975-03-03', 9, 'Р', 0);
```

```
--Запросы для выборки информации:
--Запрос о сотрудниках
SELECT
    отдел,
    ФИО,
    оклад + оклад * 0.8 AS Зарплата_в_рублях,
      оклад + оклад * 0.8 * 70 / 60 AS Зарплата_в_долларах --
Предполагается, что курс составляет 70 рублей за доллар
FROM
    Сотрудники
JOIN
              Штатное_расписание   ON   Сотрудники.код_должности   =
Штатное_расписание.код_должности
ORDER BY
    ФИО;
--Запрос по каждому подразделению
SELECT
    отдел,
    AVG(оклад + оклад * 0.8) AS Средняя_заработная_плата,
    COUNT(CASE WHEN пол = 'Ж' THEN 1 END) AS Количество_женщин,
    SUM(детей) AS Общее_количество_детей
FROM
    Сотрудники
JOIN
              Штатное_расписание   ON   Сотрудники.код_должности   =
Штатное_расписание.код_должности
GROUP BY
    отдел;
```

Вопросы ЗИ (Защита информации)

1. **Понятие компьютерного вируса, виды, вредоносные функции, пути распространения, проявление действия.** **ЕСТЬ В БИЛЕТАХ**

Компьютерный вирус - это вредоносная программа, целью которой является уничтожение, изменение или кража данных, блокировка работы компьютера или сети, перехват информации, нарушение работы аппаратной части и т. д.

Виды вируса по механизму заражения:

1) Резидентный - встраиваются в оперативную память и могут оставаться активными даже после завершения программы
2) Нерезидентный - не встраиваются в оперативную память и завершают свою работу после завершения программы

Виды вирусов по степени опасности

1) Безопасные - расходуют в основном только свободное место в ПК
2) Безвредные - действия данных вирусов влияет в основном только на работу звуковых или графических эффектах
3) Опасные - вызывают сбои работы ПК
4) Очень опасные - могут уничтожаться файлы, повреждаться ОС, а также выход из строя аппаратной части ПК

Виды вирусов по назначению

1) Рекламные - предназначен для показа рекламы на компьютере
2) Шпионские - предназначены для сбора информации пользователе без его согласия (пароли, карты, личная инфа)
3) Вымогательские - вирусы, которые блокируют доступ к файлам и требуют выкупа для разблокировки
4) Троянские вирусы - маскируются под обычные программы, и могут разрешать злоумышленникам управлять ПК, собирать личную информацию, а также устанавливать доп. вредоносное ПО.
5) Черви - основная задача это распространение вируса на другие ПК. Благодаря способности быстро распространяться, черви часто используются для выполнения фрагментов кода, созданного для повреждения системы, например, они могут удалять файлы в системе, шифровать данные для атаки программы-вымогателя, красть информацию и создавать ботнеты.

Вредоносные функции:

1) распространение
2) кража личной информации
3) удаление файлов и блокировка
4) вымогательство
5) нарушение работы аппаратной части ПК

Пути распространения

1) Электронная почта
2) физические носители (флешки)
3) всплывающие окна
4) заражение путём скрытой загрузки

Проявления действия

1) Медленная работа, сбои и зависание компьютера.
2) Печально известный «синий экран смерти».
3) Автоматическое открытие и закрытие или самостоятельное изменение программ.
4) Отсутствие места для хранения.

5) Увеличение количества всплывающих окон, панелей инструментов и нежелательных программ.

6) Отправка электронных писем и сообщений без вашего ведома.

2. Основные понятия безопасности информации: конфиденциальность, целостность, доступность ЕСТЬ В БИЛЕТАХ

конфиденциальность информации - обязательное для выполнения лицом, получившим доступ к определенной информации, требование не передавать такую информацию третьим лицам без согласия ее обладателя;

целостность - принцип, направленный на обеспечение неприкосновенности данных. Он гарантирует, что информация остается неизменной и не подвергается несанкционированным изменениям.

доступность - принцип, который гарантирует непрерывный доступ к информационным ресурсам для легитимных пользователей. Легитимные пользователи - пользователи которые имеют доступ к информации и соблюдают все правила использования.

3. Виды мер обеспечения информационной безопасности: законодательные, морально-этические, организационные, технические, программно-математические. ЕСТЬ В БИЛЕТАХ

Законодательные меры обеспечения информационной безопасности включают законы, положения и правила, которые призваны регулировать использование информации и ее защиту. Эти меры включают ограничения на сбор, хранение, использование и распространение информации, а также наказания за нарушение правил безопасности.

Морально-этические меры обеспечения информационной безопасности включают принципы этики и поведения, которые призваны обеспечить безопасность информации и защиту частной жизни всех пользователей. Эти меры включают соблюдение конфиденциальности, защиту частной информации, уважение к чужой интеллектуальной собственности и защиту от мошенничества.

Организационные меры обеспечения информационной безопасности включают управленческие мероприятия, направленные на создание безопасной среды для использования и обработки информации. Эти меры включают разработку стратегии безопасности, создание политик и процедур безопасности, обучение сотрудников и проверку контрольных механизмов.

Технические меры обеспечения информационной безопасности включают использование специальных технологий и систем чтобы предотвратить несанкционированный доступ к информации. Эти меры включают различные методы шифрования данных, средства аутентификации, системы контроля доступа и мониторинга сетевой активности.

Программно-математические меры обеспечения информационной безопасности включают разработку программного обеспечения и алгоритмов, которые обеспечивают безопасность информации и защиту от различных видов атак. Эти меры включают системы обнаружения вторжений, антивирусные программы и программы защиты данных.

4. Основные защитные механизмы построения систем защиты информации: идентификация и аутентификация. Разграничение доступа. Контроль целостности. ЕСТЬ В БИЛЕТАХ

Идентификация и аутентификация - это два основных защитных механизма, которые используются для обеспечения безопасности информации. Идентификация - это процесс установления личности пользователя или устройства, путем проверки представленных ими данных, таких как логин или ID-код. Аутентификация в свою очередь - это процесс подтверждения личности пользователя или устройства, путем проверки правильности введенного пароля, биометрических данных или использования других мер обеспечения доступа.

Разграничение доступа - это механизм, позволяющий определить, кто и в каких условиях может получить доступ к определенным ресурсам в информационной системе.

Этот механизм позволяет ограничивать доступ к конфиденциальной информации и предотвращать несанкционированные действия со стороны пользователей системы.

Контроль целостности - это механизм, который обеспечивает целостность данных в информационной системе. Он позволяет проверять, не были ли данные изменены или повреждены в процессе пересылки. Для этого используются различные методы, такие как хэширование данных, электронная подпись и цифровые сертификаты. Этот механизм позволяет предотвращать несанкционированные изменения данных и обеспечивать их целостность.

5. Криптографические механизмы конфиденциальности, целостности и аутентичности информации. Электронная цифровая подпись. ЕСТЬ В БИЛЕТАХ

Криптографические механизмы используются для защиты информации от несанкционированного доступа, изменения и подделки. Они включают в себя различные методы шифрования и хэширования данных.

Механизм конфиденциальности защищает информацию от несанкционированного доступа путем шифрования данных. Существует множество алгоритмов шифрования, включая симметричные и асимметричные методы. Симметричные алгоритмы используют один ключ для шифрования и дешифрования данных, тогда как асимметричные алгоритмы используют пару ключей: публичный и приватный.

Механизм целостности обеспечивает целостность данных в информационной системе путем использования методов хэширования. Хэширование - это процесс создания уникального числового значения для определенных данных. Если данные изменены, то хэш-значение также изменится, что позволяет обнаружить изменения.

Механизм аутентичности обеспечивает подлинность информации путем использования электронных цифровых подписей. Цифровая подпись - это электронный аналог обычной подписи на бумаге. Она подтверждает авторство и целостность электронного документа или сообщения. Цифровая подпись создается путем применения криптографических алгоритмов и защищает данные от подделки или изменения.

Таким образом, криптографические механизмы обеспечивают конфиденциальность, целостность и аутентичность информации в информационной системе, что является очень важным для обеспечения безопасности данных.

Электронная цифровая подпись (ЭЦП) - это аналог рукописной подписи в электронном виде, которая используется для подтверждения авторства и целостности электронных документов. ЭЦП обеспечивает юридическую значимость электронных документов, фиксируя информацию, которая была в документе на момент подписания.

Принцип работы ЭЦП основан на использовании пары ключей - открытого и закрытого:

1. Открытый ключ (сертификат) выдается удостоверяющим центром и используется для проверки подлинности подписи.
2. Закрытый ключ является криптографической частью ЭЦП и используется для создания подписи.

Таким образом, ЭЦП позволяет:

- Подтвердить авторство электронного документа
- Гарантировать неизменность документа с момента подписания
- Обеспечить конфиденциальность информации
- Исключить возможность отказа от авторства документа

Для работы с ЭЦП необходимо настроить рабочее место, установить специальное программное обеспечение и получить сертификат ключа проверки электронной подписи в аккредитованном удостоверяющем центре.

6. Классификация антивирусных программ. Программы-детекторы, программы-доктора, программы-ревизоры, программы-фильтры. Профилактика заражения вирусом. ЕСТЬ В БИЛЕТАХ

Антивирусные программы - это программное обеспечение, которое предназначено для защиты компьютеров или других устройств от вирусов, троянов, червей и другого вредоносного кода. Антивирусные программы можно разделить на несколько классов в зависимости от функций и задач:

1. Программы-детекторы - это программы, которые способны обнаруживать вирусы, используя базы данных сигнатур вирусов. Они сканируют файлы на наличие вредоносного кода и могут блокировать его исполнение.

2. Программы-доктора - это программы, которые могут удалять вирусы, обнаруженные программами-детекторами. Они также могут восстанавливать системные файлы, поврежденные вирусами, и лечить зараженные файлы.

3. Программы-ревизоры - это программы, которые могут проверять систему на наличие вирусов и других угроз безопасности, таких как трояны, черви, шпионское ПО. Они также могут проводить проверку целостности системных файлов, системных настроек и реестра.

4. Программы-фильтры - это программы, которые могут блокировать доступ к вредоносным сайтам, сообщениям электронной почты, файлам и другим источникам вредоносного кода. Они также могут использоваться для фильтрации веб-трафика и контроля доступа к сети.

Для профилактики заражения вирусом необходимо соблюдать ряд мер предосторожности:

1. Регулярно обновлять антивирусные программы и базы данных сигнатур вирусов.

2. Использовать антивирусное программное обеспечение от надежных производителей и устанавливать все обновления операционной системы.

3. Устанавливать только необходимые программы и приложения из проверенных источников.

4. Не открывать подозрительные письма электронной почты, файлы из непроверенных источников и ссылки на недоверенные сайты.

5. Регулярно резервировать данные на внешние устройства хранения и не использовать неизвестные USB-накопители.

6. Использовать пароли для доступа к устройствам и отдельным файлам, а также не делиться ими с другими пользователями.

Вопросы СП (Сетевое программирование)

1. Система контроля версий. Виды систем контроля версий. Примеры систем контроля версий НЕТ В БИЛЕТАХ

Система контроля версий (СКВ) — это инструмент, который позволяет отслеживать изменения в исходном коде программного продукта, хранить и управлять ими.

Виды СКВ:

1. Локальные СКВ. Эта система контроля версий используется для работы в рамках локальной машины. Примеры локальных СКВ: RCS и SCCS.

2. Централизованные СКВ. В этом случае все изменения отправляются на сервер, который находится в центре всей системы. Примеры централизованных СКВ: CVS и Subversion (SVN).

3. Распределенные СКВ. Распределенные СКВ используются для повышения производительности и защиты от сбоев. Примеры распределенных СКВ: Git и Mercurial.

Примеры систем контроля версий:

1. Git. Эта распределенная система контроля версий позволяет управлять кодом на локальной машине и на удаленных серверах.

2. Subversion (SVN). Эта централизованная система контроля версий наиболее популярна в мире открытого исходного кода.

3. Mercurial. Эта система контроля версий также распределенная и используется на компьютерах Mac, Linux и Windows.

4. Perforce. Эта централизованная система контроля версий используется для управления сложными проектами.

5. Team Foundation Server (TFS). Эта централизованная система контроля версий используется вместе с Visual Studio для управления проектами и командной работой.

2. HTTP- и HTTPS-протоколы. Методы HTTP-запроса. HTTP заголовки. Группы кодов состояния при выполнении запросов. НЕТ В БИЛЕТАХ

HTTP и HTTPS протоколы

HTTP (HyperText Transfer Protocol) и HTTPS (HyperText Transfer Protocol Secure) являются протоколами, используемыми для передачи данных в сети Интернет.

HTTP - это протокол, который используется для передачи данных в виде текста или графического изображения. Он использует порт 80 и работает без шифрования.

HTTPS - это защищенный протокол передачи данных, который использует порт 443 и работает с использованием шифрования данных с помощью SSL (Secure Sockets Layer) или TLS (Transport Layer Security).

Методы HTTP-запроса

HTTP определяет несколько методов запросов, которые позволяют клиенту и серверу взаимодействовать друг с другом:

1. GET - используется для запроса конкретного ресурса, который определяется URL-адресом.

2. POST - используется для отправки данных на сервер и обработки этих данных приложением на сервере.

3. PUT - используется для обновления ресурса на сервере.

4. DELETE - используется для удаления ресурса на сервере.

5. HEAD - используется для запроса заголовков сообщения.

6. CONNECT - используется для установления соединения с сервером через прокси-сервер.

7. OPTIONS - используется для получения списка методов, поддерживаемых сервером.

HTTP заголовки

HTTP заголовки - это часть сообщения HTTP, которая содержит информацию о запросе и ответе. Они могут быть использованы для передачи различной информации, включая куки, параметры запроса и тело сообщения.

Некоторые заголовки, которые могут быть включены в сообщение HTTP, включают:

1. User-Agent - содержит информацию о браузере клиента и операционной системе.
2. Content-Type - указывает тип содержимого, передаваемого в сообщении.
3. Content-Length - указывает длину тела сообщения в байтах.
4. Cookie - передает куки на сервер.
5. Cache-Control — указывает, какие данные можно кэшировать и на какой период времени.
6. Date - указывает дату и время отправки или получения сообщения.

Группы кодов состояния при выполнении запросов

HTTP протокол возвращает коды состояния после выполнения каждого запроса. Коды состояния делятся на несколько групп:

1xx - информационные сообщения.
2xx - успешное выполнение запроса.
3xx - перенаправление.
4xx - ошибка на стороне клиента.
5xx - ошибка на стороне сервера.

Примеры кодов состояния включают:

200 - OK; запрос выполнен успешно.
301 - Moved Permanently; ресурс перенесен на другой URL.
404 - Not Found; запрашиваемый ресурс не найден на сервере.
500 - Internal Server Error; внутренняя ошибка сервера.
503 - Service Unavailable; сервер временно недоступен.

3. Клиент-серверная архитектура: назначение блоков, описание технических устройств клиентской и серверной части, описание связи базы данных с интерфейсом. НЕТ В БИЛЕТАХ

Клиент-серверная архитектура является распространенной моделью взаимодействия между клиентскими приложениями и серверным программным обеспечением. Она представляет собой дробление одного большого приложения на две составляющие: клиентскую и серверную части. Эта модель позволяет упростить процесс разработки и расширения сложных приложений.

Клиентская часть — это программа, которая запускается на компьютере или устройстве пользователя и обеспечивает интерфейс для взаимодействия с приложением. Клиентское приложение может быть установлено на устройстве пользователя или запущено в веб-браузере с использованием технологий, таких как HTML, CSS и JavaScript.

Серверная часть — это программа, которая запущена на сервере и обрабатывает запросы клиента, выполняет операции с базой данных и возвращает результаты работы в клиентское приложение. Обычно серверные приложения работают на серверах, имеющих большую производительность и мощность, чем компьютеры или устройства пользователей.

Связь клиентской и серверной частей обеспечивается с использованием протоколов передачи данных, таких как HTTP, HTTPS, FTP и других. Когда пользователь запрашивает данные или услуги, клиентское приложение отправляет запрос на сервер, который получает его, обрабатывает и отправляет обратно в клиентское приложение.

Связь базы данных с интерфейсом обеспечивается с помощью технологий, таких как SQL, которые используются для работы с данными в разных базах данных. Интерфейс приложения использует запросы к базе данных, чтобы получить доступ к необходимым данным. База данных, в свою очередь, выдает данные клиенту через приложение сервера. Обычно база данных управляется серверным программным обеспечением, которое обрабатывает запросы клиента и обеспечивает доступ к необходимым данным.

Таким образом, клиент-серверная архитектура позволяет создавать сложные приложения, которые могут быть управляемыми удаленно и использоваться на разных платформах, что делает ее популярной и востребованной в различных областях.

4. Модель TCP/IP: назначение уровней, протоколы. Маршрутизация назначение, классификация, функции. НЕТ В БИЛЕТАХ

Модель TCP/IP – это набор стандартов, описывающих протоколы, используемые для передачи данных в компьютерных сетях. Она состоит из четырех уровней:

1. Уровень прикладных протоколов (Application layer) – обеспечивает обмен данными между приложениями различных компьютеров. Примеры протоколов: HTTP, FTP, SMTP, POP3.
2. Транспортный уровень (Transport layer) – обеспечивает передачу данных между приложениями на разных компьютерах. Примеры протоколов: TCP, UDP.
3. Сетевой уровень (Network layer) – обеспечивает передачу данных между компьютерами через сеть. Примеры протоколов: IP, ICMP.
4. Уровень сетевых интерфейсов (Link layer) – обеспечивает передачу данных между компьютерами в пределах локальной сети. Примеры протоколов: Ethernet, Wi-Fi.

Маршрутизация – это процесс выбора оптимального пути для передачи данных от отправителя к получателю в компьютерной сети. Маршрутизаторы – это устройства, которые выполняют эту функцию. Они классифицируются по типу сети, которую они обслуживают:

1. Маршрутизаторы для локальных сетей (LAN routers) – обеспечивают маршрутизацию внутри одной локальной сети.
2. Маршрутизаторы для глобальных сетей (WAN routers) – обеспечивают маршрутизацию между различными локальными сетями, которые связаны между собой через глобальную сеть.
3. Маршрутизаторы для мобильных сетей (Mobile routers) – обеспечивают маршрутизацию в мобильных сетях, которые зачастую меняют свою топологию.

Основные функции маршрутизаторов:

1. Обнаружение других устройств в сети и обмен информацией с ними.
2. Выбор оптимального пути для передачи данных.
3. Пересылка данных между различными сегментами сети.
4. Отслеживание состояния сети и оповещение администратора о возможных проблемах.

5. Назначение API НЕТ В БИЛЕТАХ

API (Application Programming Interface) – это интерфейс программирования приложений, который позволяет различным программам взаимодействовать друг с другом

Основное назначение API:

1. Интеграция приложений

API служит средством интеграции различных программ, позволяя им обмениваться данными и функциональностью. Это достигается за счет стандартизированного набора правил и спецификаций, которым должны следовать взаимодействующие приложения

2. Инкапсуляция функциональности

API предоставляет разработчикам набор функций и методов для работы с программой, скрывая при этом внутреннюю реализацию. Это позволяет использовать функциональность программы, не вникая в детали ее работы

3. Синхронизация систем

API обеспечивает синхронизацию работы различных информационных систем, позволяя им обмениваться данными в реальном времени. Это особенно важно для распределенных систем, компоненты которых могут находиться на разных платформах.

4. Автоматизация процессов

API дает возможность автоматизировать многие процессы, связанные с обменом данными между приложениями. Разработчики могут создавать программы, которые будут автоматически перемещать данные между системами по мере выполнения определенных действий и событий

5. Расширение функциональности

API позволяет расширять функциональность приложений за счет использования возможностей других программ. Разработчики могут подключать нужные сервисы и API к своим приложениям, не занимаясь реализацией этого функционала самостоятельно.

До появления API взаимодействие между программами осуществлялось напрямую, без использования стандартизированных интерфейсов. Это было неэффективно и затрудняло интеграцию различных систем:

- Программы были жестко связаны друг с другом, любые изменения в одной из них требовали модификации другой
- Отсутствовала инкапсуляция функциональности, разработчики должны были знать внутреннюю реализацию программ
- Не было возможности автоматизировать обмен данными между приложениями
- Расширение функциональности требовало внесения изменений в исходный код

6. Формы представления данных: JSON, XML. НЕТ В БИЛЕТАХ

JSON и XML являются двумя популярными форматами представления данных, используемыми для обмена данными между различными системами и приложениями. Они оба предлагают различные подходы к структуре и представлению данных, что делает их подходящими для различных сценариев использования.

JSON означает JavaScript Object Notation. Это открытый стандарт, основанный на синтаксисе JavaScript, который используется для предетавления структурированных данных. JSON легко читается как людьми, так и машинами.

XML означает Extensible Markup Language. Это язык разметки, который используется для описания данных в виде текстовых документов. XML позволяет определять собственные теги и структуры данных.

XML eXtensible Markup Language Расширяемый язык разметки	JSON Java Script Object Notation Обозначение объектов Java Script
Текстовые форматы, которые : • удобные для чтения, • имеют иерархическую структуру, • могут быть использованы многими языками программирования, • могут быть получены с помощью XMLHttpRequest.	
Язык	Текстовый формат
Представляет элементы данных	Используется для репрезентации объектов
Нет прямой поддержки массивов	Поддерживает текст и числовые типы данных, массивы и объекты
Использует открывающий и закрывающий теги	Не использует закрывающие теги, но использует для этого скобки (фигурные и квадратные)
Поддерживает пространство имен (namespace)	Не поддерживает пространство имен
Более защищен	Менее защищен
Поддерживает комментарии	Не поддерживает комментарии
Независимый формат данных, который поддерживает разные кодировки	Независимый от языка формат обмена данными, который поддерживает только UTF-8 кодирование

JSON пример

```
«employees»:[
    {«firstName»:»Lev», «lastName»:»Tolstoy»},
    {«firstName»:»Anna», «lastName»:»Karenina»},
    {«firstName»:»Aleksey», «lastName»:»Vronsky»},
]
```

XML пример:

```
<employees>
    <employee>
<firstName>Lev</firstName>
<lastName>Tolstoy</lastName>
    </employee>
    <employee>
<firstName>Anna</firstName>
<lastName>Karenina</lastName>
    </employee>
    <employee>
<firstName>Aleksey</firstName>
<lastName>Vronsky</lastName>
    </employee>
</employees>
```

7. Аутентификация и авторизация пользователей в клиент-серверных приложениях НЕТ В БИЛЕТАХ

Аутентификация и авторизация пользователей являются важными процессами в клиент-серверных приложениях. Аутентификация - это процесс проверки подлинности пользователя, который пытается получить доступ к приложению или ресурсам, а авторизация - это процесс проверки прав пользователя на доступ к определенным ресурсам в приложении.

Как правило, в клиент-серверных приложениях аутентификация проводится на стороне сервера. Для аутентификации обычно используются такие методы, как проверка имени пользователя и пароля, использование сертификатов или биометрических данных.

После успешной аутентификации сервер проводит авторизацию и определяет, какие ресурсы и функции доступны пользователю. Для этого могут использоваться различные методы авторизации, например, определение ролей и прав пользователя, настройка политик безопасности и т.д.

Чтобы обеспечить безопасность приложения, необходимо использовать надежные методы аутентификации и авторизации. Это позволит защитить приложение от несанкционированного доступа и обеспечить конфиденциальность данных пользователей.

8. Масштабирование клиент-серверных приложений НЕТ В БИЛЕТАХ

Масштабирование клиент-серверных приложений — это процесс расширения функциональности и возможностей приложения путем добавления дополнительных серверов или ресурсов.

Для масштабирования клиент-серверных приложений используются различные методики:

1. Вертикальное масштабирование — добавление ресурсов на одном сервере, например, добавление процессора или оперативной памяти.

2. Горизонтальное масштабирование - добавление новых серверов в сеть, чтобы выполнять задачи параллельно.

3. Кластеризация — объединение множества серверов в единый кластер для более эффективного распределения нагрузки между ними.

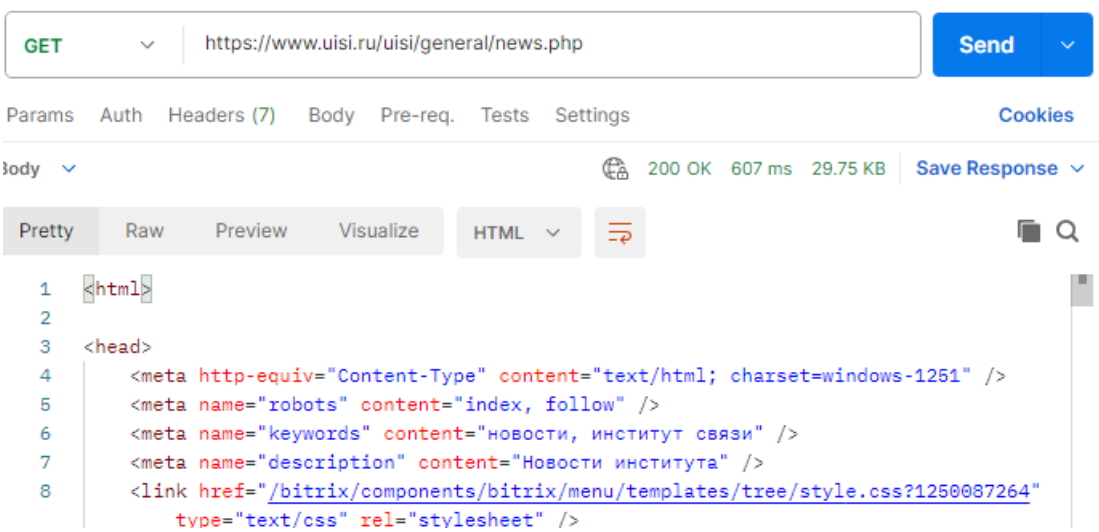
4. Разделение функционала - разделение сложных приложений на более мелкие, независимые компоненты, что позволяет легче масштабировать каждую из них.

5. Использование облачных технологий - перенос приложения на облачные сервера, которые обеспечивают гибкость и масштабируемость в зависимости от запросов пользователей.

Масштабирование клиент-серверных приложений позволяет поддерживать высокую производительность и доступность при росте количества пользователей или объема данных.

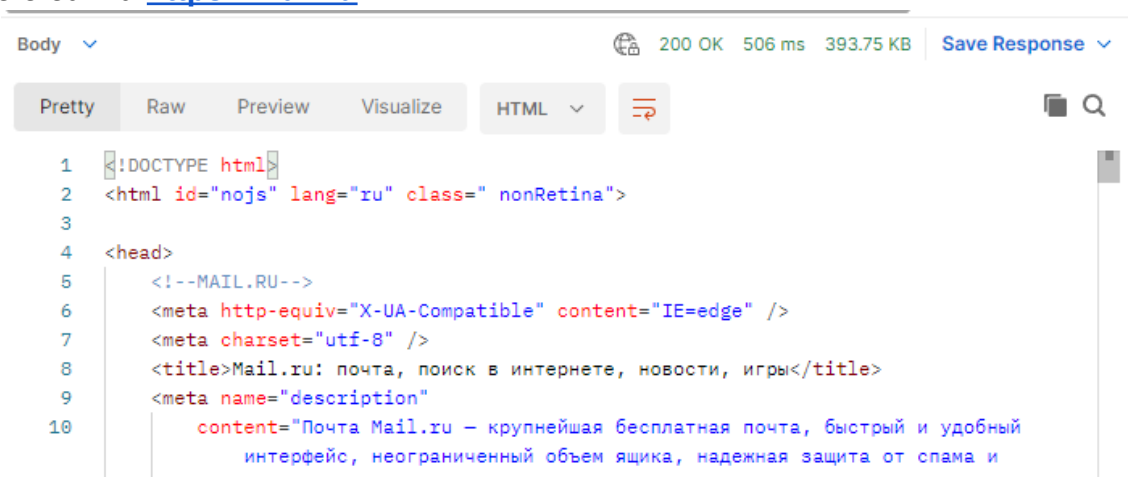
~~Практика СП: В БИЛЕТАХ НЕТ~~

~~1. При помощи программного обеспечения Postman проанализируйте GET запрос с сайта https://www.uisi.ru/uisi/general/news.php~~



~~URL: https://mail.ru.~~
~~Протокол: https~~
~~Домен: www.uisi.ru~~
~~Поддомен: www~~
~~Путь: /uisi/general/news.php~~
~~Файл: news.php~~
~~Метод: В данном случае, это GET.~~
~~Заголовки: пусты~~
~~Параметры запроса: пусты~~
~~Тело запроса: отсутствует, потому что у GET запроса его нет.~~
~~Статус код: 200.~~
~~Инициализация сокета 1,45 мс~~
~~Поиск по DNS 92,66 мс~~
~~Подтверждение связи по протоколу TCP 71,18 мс~~
~~Подтверждение связи по протоколу SSL 134,8 мс~~
~~Начало передачи 274. 21 мс~~
~~Скачивание 31,74 мс~~
~~Процесс 1,53 мс~~
~~Время ответа: 607 ms.~~
~~Ответ: ответ выдан в виде html файла.~~
~~Размер ответа: 29.75 кб.~~

~~2. При помощи программного обеспечения Postman проанализируйте GET запрос с сайта https://mail.ru~~



~~URL: https://mail.ru.~~
~~Протокол: https~~

Домен: mail.ru
Метод: В данном случае, это GET.
Заголовки: пусты
Параметры запроса: пусты
Тело запроса: отсутствует, потому что у GET запроса его нет.
Статус код: 200.
Инициализация сокета 1,47 мс
Поиск по DNS 34,49 мс
Подтверждение по протоколу TCP 26 мс
Подтверждение по протоколу SSL 59,32 мс
Начало передачи 345,02 мс
Скачать 28,93 мс
Процесс 1,26 мс
Время ответа: 506ms.
Ответ: ответ выдан в виде html файла.
Размер ответа: 393.75 кб.

3 Создать страницу, где следует отобразить следующую информацию:
Студент Фамилия ИО
Мое любимое стихотворение:
##указать любое стихотворение, в котором должны быть различные стили написания текста (курсив, жирный, перечеркнутый и т.п.). Подключение стилей должно осуществляться через файл style.css.
style.css

```
.favorite-poem p{
    margin: 0;
    padding: 0;
    font-size: 16px;
}

.favorite-poem p:first-of-type{
    font-family: sans-serif;
    font-weight: 900;
}

.favorite-poem p:nth-child(4n - 2){
    font-family: Helvetica;
    font-style: italic;
    font-weight: 300;
    color: aqua;
}

.favorite-poem p:nth-child(4n - 1){
    font-family: Arial;
    font-weight: 400;
    text-decoration: underline;
    font-size: 20px;
    color: brown;
}

.favorite-poem p:nth-child(4n){
    font-family: initial;
    font-weight: 100;
    font-size: 13px;
    color: aquamarine;
}
```

index.html

```
<!DOCTYPE html>
<html lang="ru">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width,
initial-scale=1.0">
    <title>Студент и стихотворение</title>
    <link rel="stylesheet" href="./styles.css">
</head>
<body>
    <div class="student">
        <h1 class="student__fio">
            Капралов А.С.
        </h1>
    </div>
    <h2>Любимое стихотворение:</h2>
    <div class="favorite-poem">
        <p>Мама — это первое ведь слово,</p>
        <p>Мама — важный в жизни человек!</p>
        <p>Мамочка, ты будь всегда здорова</p>
        <p>И живи, пожалуйста, сто лет!</p>
    </div>
</body>
</html>
```

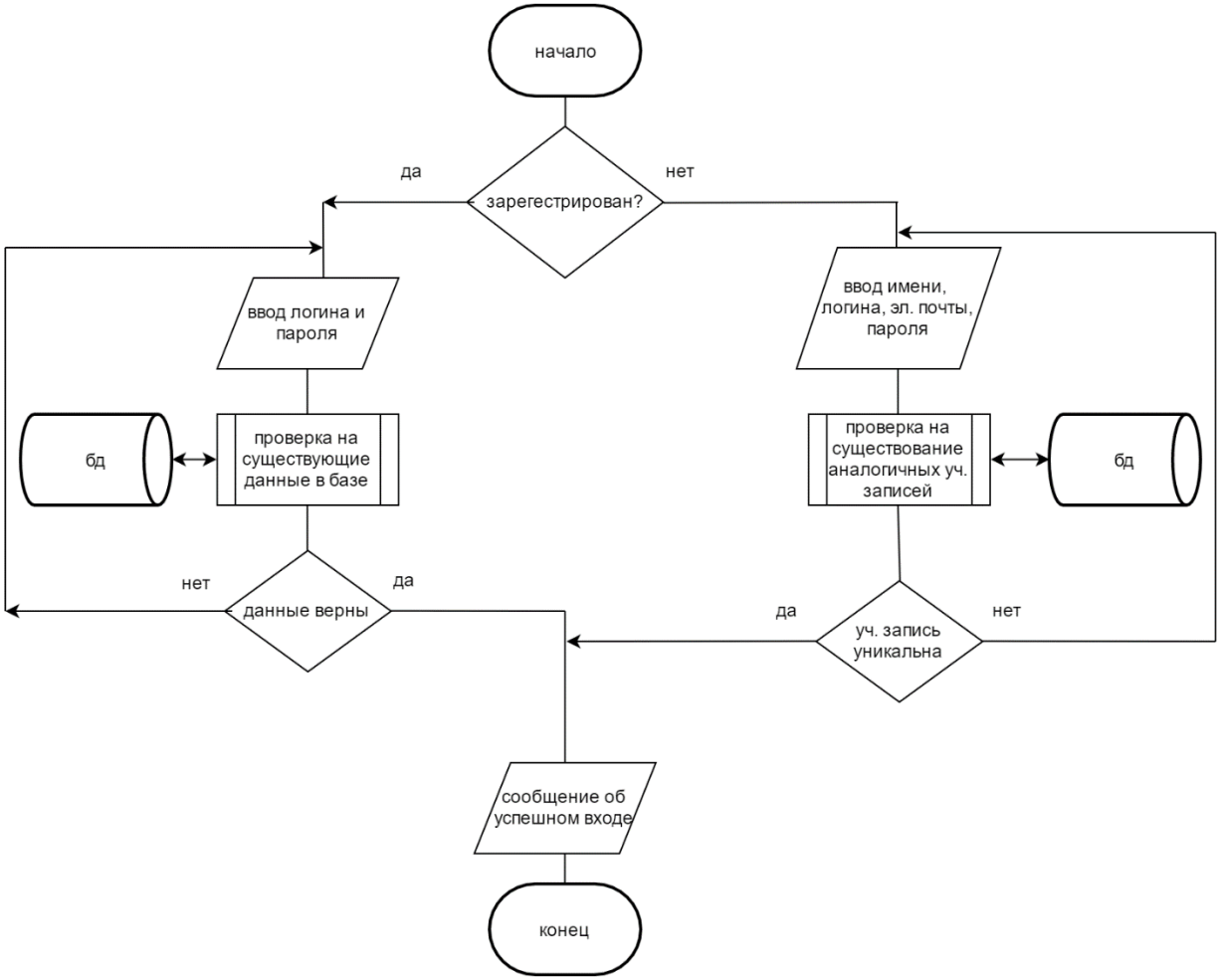
4. Разработать веб-приложение, которое будет решать математическую задачу по расчету Гипотенузу и площадь прямоугольного треугольника с катетами a, b с вводом и выводом результата на странице.

```
<!DOCTYPE html>
<html>
<head>
 <meta charset="utf-8">
        <meta        name="viewport"        content="width=device-width,
initial-scale=1">
    <title>Расчет гипотенузы и площади прямоугольного
треугольника</title>
                                <link                                rel="stylesheet"
href="https://stackpath.bootstrapcdn.com/bootstrap/4.1.1/css/bootstrap.m
in.css">
                                                                    <script
src="https://code.jquery.com/jquery-3.3.1.slim.min.js"></script>
                                                                    <script
src="https://cdnjs.cloudflare.com/ajax/libs/popper.js/1.14.3/umd/popper.
min.js"></script>
                                                                    <script
src="https://stackpath.bootstrapcdn.com/bootstrap/4.1.1/js/bootstrap.min
.js"></script>
</head>
<body>
 <div class="container">
    <h1>Расчет гипотенузы и площади прямоугольного треугольника</h1>
    <form id="calc-form">
        <div class="form-group">
            <label for="input-a">Катет a:</label>
                <input  type="number"  class="form-control"  id="input-a"
required>
        </div>
```

```
        <div class="form-group">
            <label for="input_b">Катет b:</label>
            <input type="number" class="form-control" id="input_b" required>
        </div>
        <button type="submit" class="btn btn-primary">Рассчитать</button>
    </form>
    <div id="result" class="mt-3" style="display: none;"></div>
</div>
<script>
    $(document).ready(function() {
        $('#calc-form').submit(function(event) {
            event.preventDefault(); // предотвращаем отправку формы
            var a = parseFloat($('#input_a').val());
            var b = parseFloat($('#input_b').val());
            var c = Math.sqrt(a*a + b*b); // расчет гипотенузы
            var s = a*b/2; // расчет площади
            $('#result').html('<p>Гипотенуза: '+c.toFixed(2)+'</p><p>Площадь: '+s.toFixed(2)+'</p>').show();
        });
    });
</script>
</body>
</html>
```

5. Разработать блок-схему процесса авторизации согласно ГОСТ 19.701.

https://darminaopel.ru/full_img/65cb8ca642659df3b3e230cb/6



Вопросы Программирование

1. **Построить 3D график по следующим формулам: ЕСТЬ В БИЛЕТАХ**

x=u cos(u) (cos(v)+1)

y=u sin(u) (cos(v)+1)

z=u sin(v)

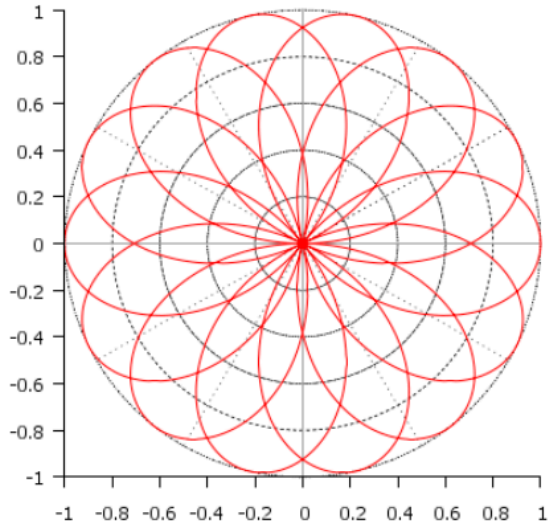
u Î[0;3π], v Î[-π ; π]

Графиков должно быть два: каркасный и сплошной – разных цветов.

```
import numpy as np
import plotly.graph_objects as go

# Генерация значений u и v
u = np.linspace(0, 3*np.pi, 10)
v = np.linspace(-np.pi, np.pi, 10)

# Создание сетки значений u и v
# U - это массив u, повторенный 10 раз
# V - это массив v, повторенный 10 раз
U, V = np.meshgrid(u, v)

# Вычисление значений x, y и z
X = U * np.cos(U) * (np.cos(V) + 1)
Y = U * np.sin(U) * (np.cos(V) + 1)
Z = U * np.sin(V)

fig = go.Figure()

# Добавление сплошного графика
fig.add_trace(go.Surface(z=Z, x=X, y=Y))

# Добавление каркасного графика
fig.add_trace(
    go.Scatter3d(
        x=X.flatten(), y=Y.flatten(), z=Z.flatten(),
        mode='lines', marker=dict(color='green')
    )
)

fig.show()
```

2. **Используя возможности библиотеки Matplotlib, постройте график полярной розы в полярной системе координат. Формула розы и ее вид представлен ниже ЕСТЬ В БИЛЕТАХ**

$$r(t) = \sin\left(\frac{7}{4}t\right), \quad t \in [0; 8\pi]$$




```
import numpy as np
import matplotlib.pyplot as plt

# Генерация значений t от 0 до 8*pi
t_values = np.linspace(0, 8*np.pi, 1000)
# Вычисление радиуса
radii = np.sin(7/4 * t_values)

plt.polar(t_values, radii)
plt.show()
```

~~3. Постройте график спирали по формуле: x=t sin(t), y=t cos(t), t Î[0; 5 π]. Оптимальный вариант – использовать возможности библиотеки Matplotlib.~~ ~~НЕТ В БИЛЕТАХ~~

```
import numpy as np
import matplotlib.pyplot as plt

# Определение диапазона значений t
t = np.linspace(0, 5*np.pi, 100)

# Вычисление значений x и y
x = t * np.sin(t)
y = t * np.cos(t)

plt.plot(x, y)
plt.show()
```

4. Разработайте форму, которая обеспечивает ввод информации о студентах (ФИО, № группы, 3 оценки за сессию) в базу данных. Одна из функций формы – вывод содержимого базы в текстовое окно. ЕСТЬ В БИЛЕТАХ]

Код на Python с Tkinter более легкий (Колташев ЯА)

```
import tkinter as tk
import sqlite3

# Создание базы данных и таблицы для студентов
conn = sqlite3.connect('students.db')
c = conn.cursor()
c.execute('''CREATE TABLE IF NOT EXISTS students
             (id INTEGER PRIMARY KEY, name TEXT, group_number TEXT,
grade1 INTEGER, grade2 INTEGER, grade3 INTEGER)''')
conn.commit()

# Функция для добавления студента в базу данных
def add_student():
    name = entry_name.get()
    group_number = entry_group.get()
    grade1 = int(entry_grade1.get())
    grade2 = int(entry_grade2.get())
    grade3 = int(entry_grade3.get())

     c.execute("INSERT INTO students (name, group_number, grade1,
grade2, grade3) VALUES (?, ?, ?, ?, ?)", (name, group_number,
grade1, grade2, grade3))
    conn.commit()

# Функция для вывода содержимого базы данных в текстовое окно
def show_students():
    c.execute("SELECT * FROM students")
    students = c.fetchall()
    text.delete(1.0, tk.END)
    for student in students:
        text.insert(tk.END, f"{student}\n")

# Создание графического интерфейса
root = tk.Tk()
root.title("Форма для ввода информации о студентах")

label_name = tk.Label(root, text="ФИО:")
label_name.pack()
entry_name = tk.Entry(root)
entry_name.pack()

label_group = tk.Label(root, text="№ группы:")
label_group.pack()
entry_group = tk.Entry(root)
entry_group.pack()

label_grades = tk.Label(root, text="Оценки за сессию (через
пробел):")
label_grades.pack()
entry_grade1 = tk.Entry(root)
entry_grade1.pack()
entry_grade2 = tk.Entry(root)
entry_grade2.pack()
entry_grade3 = tk.Entry(root)
entry_grade3.pack()

button_add = tk.Button(root, text="Добавить студента",
command=add_student)
```

```
button_add.pack()

button_show = tk.Button(root, text="Показать студентов",
command=show_students)
button_show.pack()

text = tk.Text(root)
text.pack()

root.mainloop()
```

Капралов А.С. - я хз нужен ли в этом задании сервак, поэтому сделал на всякий случай. (код на сером фоне к коду в табличках не относиться)

Сервак на питоне

```
from flask import Flask, request, jsonify, render_template
import sqlite3

app = Flask(__name__, static_url_path='/static')

# Создание базы данных и таблицы, если они еще не существуют
def create_db():
    conn = sqlite3.connect('users.db')
    cursor = conn.cursor()
    cursor.execute('''CREATE TABLE IF NOT EXISTS users
                      (surname TEXT, name TEXT, second_name TEXT,
group_name TEXT, mark1 INTEGER, mark2 INTEGER, mark3 INTEGER)''')
    conn.commit()
    conn.close()

@app.route('/')
def home():
    return render_template('index.html')

@app.route('/add_user', methods=['POST'])
def add_user():
    data = request.get_json()  # Получаем данные пользователя в
формате JSON
    if not data:
        return jsonify({"error": "No data received"}), 400

    surname = data.get('surname')
    name = data.get('name')
    second_name = data.get('second_name')
    group_name = data.get('group_name')
    mark1 = data.get('mark1')
    mark2 = data.get('mark2')
    mark3 = data.get('mark3')

    if not all([surname, name, second_name, group_name,
str(mark1), str(mark2), str(mark3)]):  # Проверяем, что все поля
заполнены
        return jsonify({"error": "Missing required fields"}), 400

    try:
        conn = sqlite3.connect('users.db')
        cursor = conn.cursor()
```

```
        cursor.execute("INSERT INTO users VALUES (?,?,?,?,?,?,?)",
(surname, name, second_name, group_name, mark1, mark2, mark3))
        conn.commit()
        conn.close()

        return jsonify({"status": 200})
    except:

        return jsonify({"status": 500})

@app.route('/users', methods=['GET'])
def get_users():
    try :
        conn = sqlite3.connect('users.db')
        cursor = conn.cursor()
        cursor.execute("SELECT * FROM users")
        users = cursor.fetchall()
        conn.close()

        # Преобразование списка кортежей в список словарей для
удобства
        users_list = [{"surname": user[0], "name": user[1],
"second_name": user[2], "group_name": user[3], "mark1": user[4],
"mark2": user[5], "mark3": user[6]} for user in users]

        return jsonify({"users": users_list, "status": 200})

    except:
        return jsonify({"status": 500})


if __name__ == '__main__':
    create_db()  # Создание базы данных и таблицы при запуске
приложения
    app.run(debug=True)
```

html файл

```
<!DOCTYPE html>
<html lang="ru">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width,
initial-scale=1.0">
    <title>Студенты и оценки</title>
    <link rel="stylesheet" href="../static/styles.css">
</head>
<body>
    <div class="wrapper">
        <form class="form" method="post">
            <input type="text" name="surname" required
placeholder="Фамилия">
            <input type="text" name="name" required
placeholder="Имя">
            <input type="text" name="second_name" required
placeholder="Отчество">
```

```
            <input type="text" name="group_name" required
placeholder="Номер группы">
            <div class="marks">
                <input type="number" min="1" max="5" name="mark1"
required placeholder="Оценка 1">
                <input type="number" min="1" max="5" name="mark2"
required placeholder="Оценка 2">
                <input type="number" min="1" max="5" name="mark3"
required placeholder="Оценка 3">
            </div>
            <input value="Записать студента" type="submit">
        </form>
        <table class="students">
            <thead class="students__head">
                <tr>
                    <th class="students__th">Группа</th>
                    <th class="students__th">ФИО</th>
                    <th class="students__th">Оценка 1</th>
                    <th class="students__th">Оценка 2</th>
                    <th class="students__th">Оценка 3</th>
                </tr>
            </thead>
            <tbody class="students__list">

            </tbody>
        </table>
    </div>
    <template id="student">
        <tr class="students__item">
            <td class="students__item-group"></td>
            <td class="students__item-fio"></td>
            <td class="students__item-mark1"></td>
            <td class="students__item-mark2"></td>
            <td class="students__item-mark3"></td>
        </tr>
    </template>
</body>
<script src="../static/script.js"></script>
</html>
```

css файл, но можно и без него

```
.wrapper {
    display: flex;
    align-items: flex-start;
    gap: 10px;
}
.form {
    display: flex;
    flex-direction: column;
    gap: 10px;
    flex: 0 1 400px;
    position: sticky;
    top: 0px;
}
```

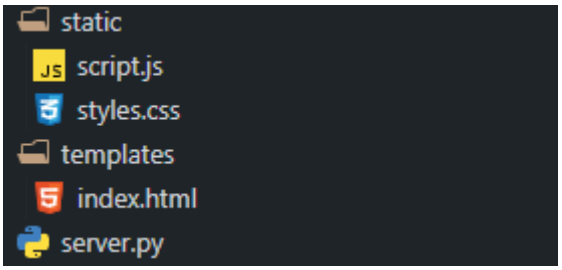
```
.marks {
    display: flex;
    gap: 10px;
    flex: 100%;
}

.marks input {
    flex: 1 1 30%;
    width: 0;
}

.students__head {
    position: sticky;
    top: 0px;
    background-color: white;
}
.students__th {
    white-space: nowrap;
    padding: 10px;
}

.students__item td {
    padding: 10px;
    text-align: center;
}

.students__item .students__item-fio {
    white-space: nowrap;
    text-align: left;
}
```

js файл - обязателен

```
const form = document.querySelector('.form');
const students = document.querySelector('.students__list');
const studentContent = document.querySelector('#student').content

async function fetchUsers() {
    const response = await fetch('/users');
    const result = await response.json();

    if (result.status == 200) {
        result.users.forEach(user => {
            addStudent(user);
        })
    }
}

function addStudent(studentData) {
    const { surname, name, second_name, group_name, mark1, mark2,
mark3 } = studentData;
    const student = studentContent.cloneNode(true);

    student.querySelector('.students__item-fio').textContent =
`${surname} ${name} ${second_name}`;
    student.querySelector('.students__item-group').textContent =
```

```
group_name;

    [mark1, mark2, mark3].forEach((mark, ind) => {
        student.querySelector('.students__item-mark' + (ind +
1)).textContent = mark;
    });

    students.append(student);
}

form.addEventListener('submit', async function (e) {
    e.preventDefault();

    const user = Object.fromEntries(new FormData(this));
    const response = await fetch('/add_user', {
        method: 'POST',
        headers: {
            'Content-Type': 'application/json;charset=utf-8'
        },
        body: JSON.stringify(user)
    });

    const result = await response.json();

    if (result.status == 200) {
        addStudent(user);
    }
});

fetchUsers();
```

ВАЖНО вот структура, а иначе не запуститься:



Чей-то код, не моё, не трогаю:

```
<!DOCTYPE html>
<html>
 <head>
   <meta charset="utf-8">
   <title>Форма информации о студенте</title>
 </head>
 <body>
   <h1>Форма информации о студенте</h1>

   <form id="student-form">
     <label for="name">ФИО:</label>
     <input type="text" id="name" required>
     <br><br>

     <label for="group-number">Группа:</label>
```

```
        <input type="text" id="group-number" required>
        <br><br>

        <label for="mark1">Сессия #1 Оценка:</label>
        <input type="number" id="mark1" min="0" max="5" required>
        <br><br>

        <label for="mark2">Сессия #2 Оценка:</label>
        <input type="number" id="mark2" min="0" max="5" required>
        <br><br>

        <label for="mark3">Сессия #3 Оценка:</label>
        <input type="number" id="mark3" min="0" max="5" required>
        <br><br>

        <button type="submit">Добавить студента</button>
      </form>

      <br>

      <h2>Список студентов:</h2>
      <ul id="students-list"></ul>

      <script>
        const form = document.getElementById('student-form');
        const studentsList = document.getElementById('students-list');
          let students = []; // array to store information about the
students

        form.addEventListener('submit', (e) => {
          e.preventDefault();

          // get values from form fields
          const name = form.querySelector('#name').value;
                                          const     groupNumber     =
form.querySelector('#group-number').value;
          const mark1 = parseInt(form.querySelector('#mark1').value);
          const mark2 = parseInt(form.querySelector('#mark2').value);
          const mark3 = parseInt(form.querySelector('#mark3').value);

          // create an object with student information
          const student = {
            name,
            groupNumber,
            marks: [mark1, mark2, mark3]
          };

          // add student to the array
          students.push(student);

          // update the list of students
          updateStudentsList();

          // reset the form fields
          form.reset();
        });

        function updateStudentsList() {
          let html = '';
```

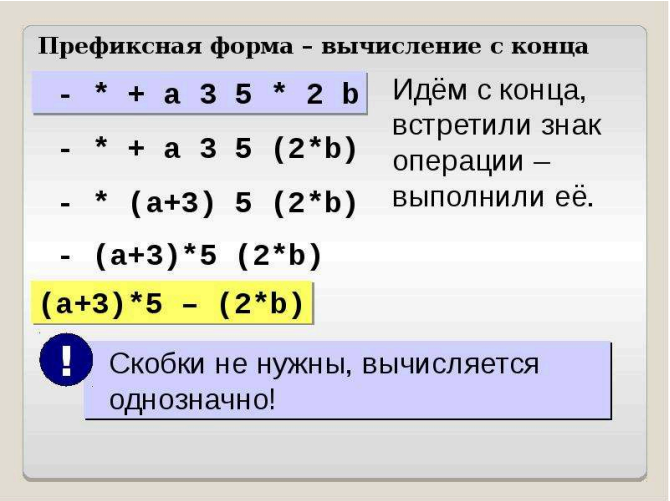
```
            // loop through the array and generate list items for each
student
            students.forEach((student) => {
              html += `<li>
                                                    ${student.name},  Группа
#${student.groupNumber}, Оценки: ${student.marks.join(', ')}
                        </li>`;
            });

            // update the list with the new items
            studentsList.innerHTML = html;
          }
        </script>
      </body>
    </html>
    >
```

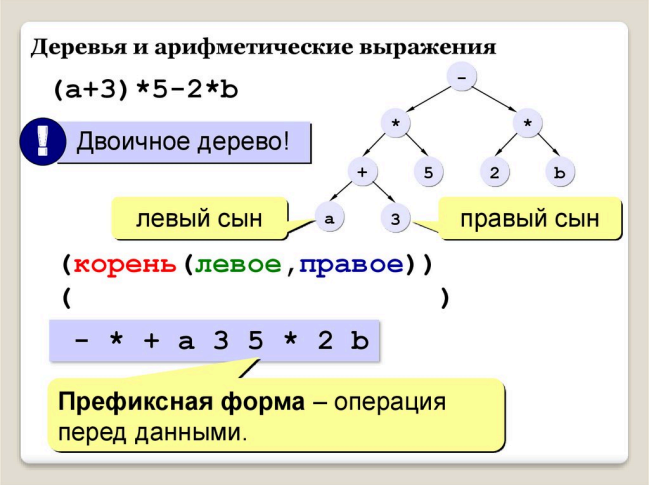
5. Для построения графика функции $y=2x^2+4$ требуются массивы x на отрезке от -5 до 5 с шагом 0.1 и массив y, определяемый по формуле. Составьте скрипт, которые такие массивы создаст (можно, например, с использованием библиотеки NumPy). Определите минимальное значение y в соответствующем массиве. ЕСТЬ В БИЛЕТАХ

```
    import numpy as np

    x = np.arange(-5.0, 5.0, 0.1)
    y = ((2*x)**2) + 4
    min_y = np.min(y)

    print("Массив x:", x)
    print("Массив y:", y)
    print("Минимальное значение y:", min_y)
```

Вопросы СиАОД (Структуры и алгоритмы обработки данных)

1. **Нарисуйте дерево, соответствующее префиксному выражению *a+b*c+de.**
ЕСТЬ В БИЛЕТАХ





https://www.calcont.in/Conversion/prefix_to_infix

Решение

* a + b * c + d e
1. * a + b * c (d+e)
2. * a + b c * (d+e)
3. * a b + c * (d+e)
4. a * (b + c * (d+e))

Дерево

```
        *
      /   \
     a     +
         /   \
        b     *
            /   \
           c     +
               /   \
              d     e
```

2. **Преобразуйте выражение ((a+b)+c*(d+e)+f)*(g+h) в префиксную форму**
ЕСТЬ В БИЛЕТАХ

```
((a+b)+c*(d+e)+f)*(g+h)
* (a+b)+c*(d+e)+f g+h
* + (a+b)+c*(d+e) f g+h
* + + (a+b) c*(d+e) f g+h
* + + (a+b) * c (d+e) f g+h
* + + + a b * c (d+e) f (g+h)
* + + + a b * c + d e f + g h
```

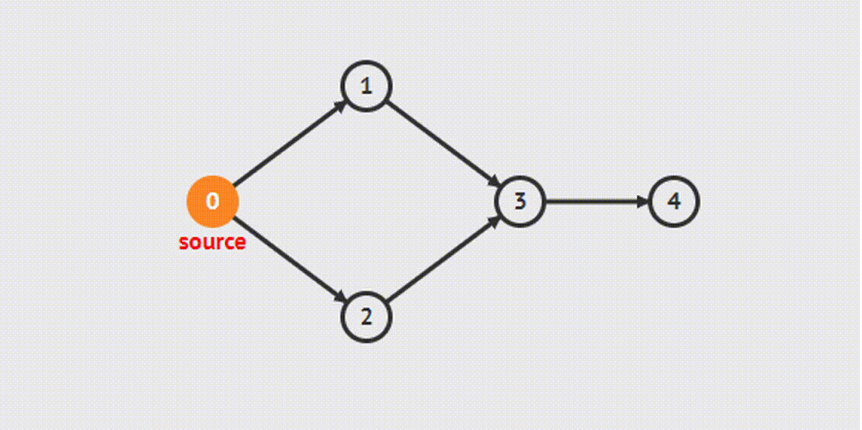
Префиксная форма (также известная как польская нотация) - это способ записи арифметических или логических выражений, в котором операторы располагаются перед своими операндами. В отличие от обычной инфиксной записи, где операторы располагаются между операндами, в префиксной форме операторы всегда идут первыми.

3. Графы. Обходы графов. ЕСТЬ В БИЛЕТАХ

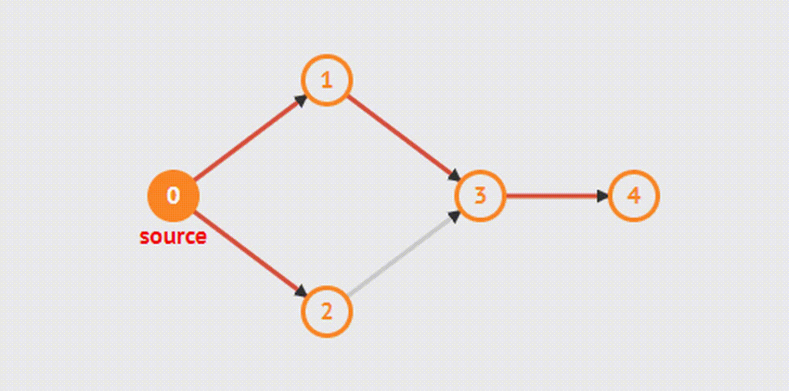
Граф - это абстрактная математическая модель, представляющая собой набор вершин и ребер, соединяющих эти вершины. Графы широко используются в математике, информатике, экономике и других науках для моделирования различных ситуаций.

Обход графа - это процесс посещения каждой вершины графа по некоторому правилу. Существует несколько методов обхода графов:

1. Обход в глубину (DFS) - начиная с одной вершины, процесс идет вглубь каждой ветки, пока не достигнет тупика, затем возвращается назад и продолжает идти по другой ветке.

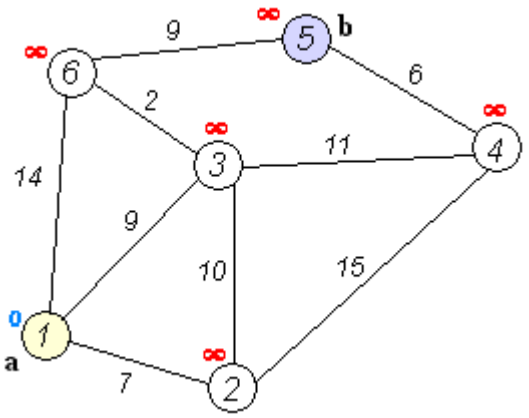


2. Обход в ширину (BFS) - начиная с одной вершины, посещает сначала все вершины, расположенные на одном уровне, затем переходит к следующему уровню.



3. Топологическая сортировка - используется для ориентированных графов, при которой вершины графа упорядочиваются в линейном порядке так, чтобы ребра указывали только на более ранние вершины.

4. Алгоритм Дейкстры - используется для определения кратчайшего пути между двумя вершинами взвешенного графа.



Графы и обходы графов являются важным математическим инструментом для решения многих задач в различных областях науки и техники.

4. Графы. Способы реализации. ЕСТЬ В БИЛЕТАХ

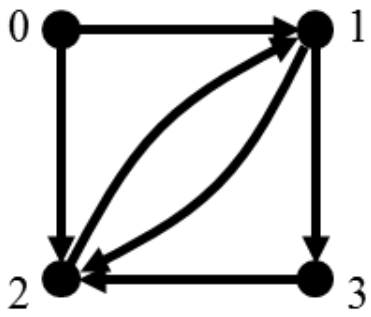
Граф - это абстрактная математическая модель, представляющая собой набор вершин и ребер, соединяющих эти вершины. Графы широко используются в математике, информатике, экономике и других науках для моделирования различных ситуаций.

Графы можно реализовать разными способами, в зависимости от типа графа и типа задач, которые необходимо решить. Некоторые из наиболее распространенных способов реализации графов:

1. Матрица смежности. Это двумерный массив, где каждый элемент [i,j] равен 1, если есть ребро (i,j), и 0, если ребра нет. Для неориентированного графа матрица смежности симметрична относительно главной диагонали.
2. Список смежности. Это массив, где каждый элемент i представляет список соседних вершин для вершины i. В списке могут содержаться номера вершин или ссылки на сами вершины.
3. Матрица инцидентности. Это двумерный массив, где каждый элемент [i,j] равен 1, если ребро j инцидентно вершине i, и -1, если ребро j исходит из вершины i. В этом случае количество столбцов равно количеству ребер.
4. Список ребер. Это массив, где каждый элемент i представляет ребро (вершины начала и конца). Этот способ реализации удобен для обхода всех ребер графа.

Каждый из этих способов реализации имеет свои преимущества и недостатки, и выбор конкретного способа зависит от типа задачи и требований к производительности.

Граф
N = 4, M = 6



Список рёбер

(0; 1)
(0; 2)
(1; 2)
(1; 3)
(2; 1)
(3; 2)

Матрица смежности

	0	1	2	3
0	0	1	1	0
1	0	0	1	1
2	0	1	0	0
3	0	0	1	0

Списки смежности

0	1	2
1	2	3
2	1	
3	2	

5. Графы. Построение минимального остовного дерева. ЕСТЬ В БИЛЕТАХ

Граф - это абстрактная математическая модель, представляющая собой набор вершин и ребер, соединяющих эти вершины. Графы широко используются в математике, информатике, экономике и других науках для моделирования различных ситуаций.

Минимальное остовное дерево (minimum spanning tree, MST) для связного взвешенного графа - это подграф графа, являющийся деревом, т.е. связным графом без циклов, и содержащий все вершины графа, которые имеет минимальную сумму весов ребер. Как построить минимальное остовное дерево?

Один из наиболее эффективных алгоритмов построения минимального остовного дерева - это алгоритм Прима:

1. Выберите случайную вершину графа и добавьте ее в MST.
2. Выберите ребро минимального веса, идущее из вершины, принадлежащей MST, в вершину, не принадлежащую MST, и добавьте эту вершину в MST. Повторяйте этот шаг, пока все вершины графа не будут принадлежать MST.
3. Повторяйте шаг 2 до тех пор, пока MST не станет деревом.

Алгоритм Крускала - еще один известный алгоритм построения MST:

1. Сортируйте все ребра графа в порядке возрастания их веса.
2. Повторяйте следующее: для каждого ребра проверьте, создает ли оно цикл в текущем подграфе графа. Если ребро не создает цикла, добавьте его в MST. Если создает - пропустите его.

3. Повторяйте шаг 2, пока MST не станет деревом.

Алгоритм Прима использует список смежности, а алгоритм Крускала использует систему непересекающихся множеств (disjoint-set data structure), которая позволяет быстро проверять наличие циклов. Оба алгоритма имеют временную сложность O(m * log n), где m - количество ребер, n - количество вершин графа.

6. Графы. Кратчайшие расстояния ЕСТЬ В БИЛЕТАХ

Граф - это абстрактная математическая модель, представляющая собой набор вершин и ребер, соединяющих эти вершины. Графы широко используются в математике, информатике, экономике и других науках для моделирования различных ситуаций.

Кратчайшие расстояния (shortest path) - это кратчайшие пути между двумя вершинами в графе, учитывая веса ребер. Применяются для решения множества задач, таких как построение маршрута на карте, оптимизация транспортных маршрутов и т.д.

Существует несколько алгоритмов, которые позволяют вычислить кратчайшие расстояния в графе. Некоторые из них:

1. Алгоритм Дейкстры - находит кратчайшие расстояния от одной начальной вершины до всех остальных вершин в взвешенном графе. Этот алгоритм работает только с неотрицательными весами ребер.
2. Алгоритм Флойда-Уоршелла - находит кратчайшие расстояния между всеми парами вершин в графе. Этот алгоритм может работать с отрицательными весами ребер.
3. Алгоритм Беллмана-Форда - находит кратчайшие расстояния от одной начальной вершины до всех остальных вершин в графе, причем веса ребер могут быть отрицательными, но граф не должен содержать циклов отрицательного веса.
4. Алгоритм Джонсона - находит кратчайшие расстояния между всеми парами вершин в графе, включая графы с отрицательными весами ребер, использует модификацию алгоритма Беллмана-Форда.

Каждый из этих алгоритмов имеет свои особенности и ограничения, поэтому выбор конкретного алгоритма зависит от типа задачи и свойств графа.

7. Поиск в линейных структурах. ЕСТЬ В БИЛЕТАХ

Поиск в линейных структурах - это процесс нахождения элемента в упорядоченной или неупорядоченной последовательности, такой как массив, связный список, стек или очередь. Существуют разные алгоритмы поиска в линейных структурах в зависимости от типа структуры данных и результата, который необходимо получить.

1. Линейный поиск (Linear search) - это самый простой алгоритм поиска, который используется для поиска элемента в неупорядоченной последовательности. Поиск начинается с первого элемента, и каждый элемент последовательно сравнивается с искомым элементом до его обнаружения или до конца последовательности.
2. Бинарный поиск (Binary search) - это алгоритм поиска в упорядоченной последовательности. Последовательность делится пополам, и искомый элемент сравнивается с элементом в середине. Если элемент в середине больше искомого, то поиск продолжается в левой половине, иначе в правой половине. Этот процесс повторяется, пока искомый элемент не будет найден.
3. Интерполяционный поиск (Interpolation search) - это алгоритм поиска в упорядоченной последовательности, который использует интерполяцию для более быстрого нахождения элемента. Этот алгоритм предполагает, что элементы равномерно распределены в последовательности, и находит положение искомого элемента, основываясь на значениях первого и последнего элементов, а также на свойствах искомого элемента.
4. Поиск по хэш-таблице (Hash table lookup) - это алгоритм поиска, используемый для поиска элемента по ключу в хэш-таблице. Этот алгоритм использует функцию хэширования для преобразования ключа элемента в индекс массива, где элемент может быть найден.
5. Поиск в связанном списке (Linked list search) - это алгоритм поиска элемента в связанном списке, который последовательно проходит по узлам связанного списка до обнаружения искомого элемента.

Каждый из этих алгоритмов имеет свои особенности и ограничения, и выбор конкретного алгоритма зависит от типа линейной структуры и условий задачи.

8. АТД - стек. Реализация с помощью указателей. ЕСТЬ В БИЛЕТАХ

АТД "Стек" - это абстрактный тип данных, представляющий собой упорядоченную коллекцию элементов, в которой добавление и удаление элементов всегда происходит с вершины стека. Стек работает по принципу LIFO (Last-In-First-Out) - последний элемент, помещенный в стек, будет извлечен первым.
Основные операции со стеком:

Push - добавление элемента на вершину стека

Pop - удаление элемента с вершины стека

Peek - получение элемента с вершины стека без его удаления

Стек часто реализуется на основе массива фиксированного размера или связного списка. При реализации на массиве размер стека должен быть известен заранее.

```
class Node:
    def __init__(self, data):
        self.data = data
        self.next = None

class Stack:
    def __init__(self):
        self.top = None

    # Добавить в стек
    def push(self, data):
        new_node = Node(data)
        new_node.next = self.top
        self.top = new_node

    # Извлечь верхний элемент
    def pop(self):
        if self.top is None:
            return None
        data = self.top.data
        self.top = self.top.next
        return data

    # Посмотреть содержание верхнего элемента
    def peek(self):
        if self.top is None:
            return None
        return self.top.data

    # Проверка есть ли элементы в стеке
    def is_empty(self):
        return self.top is None

# Пример использования стека
stack = Stack()
stack.push(1)
stack.push(2)
stack.push(3)

print(stack.pop())  # Вывод: 3
print(stack.peek())  # Вывод: 2
print(stack.is_empty())  # Вывод: False
```

9. АТД - список. Реализация с помощью указателей. ЕСТЬ В БИЛЕТАХ

АТД "Список" - это абстрактный тип данных, который представляет собой структуру данных, позволяющую хранить и управлять коллекцией элементов. В отличие от массива, список позволяет добавлять и удалять элементы динамически, без необходимости определения размера заранее.

Основные операции с АТД "Список":

Добавление элемента в список

Удаление элемента из списка

Получение элемента по индексу

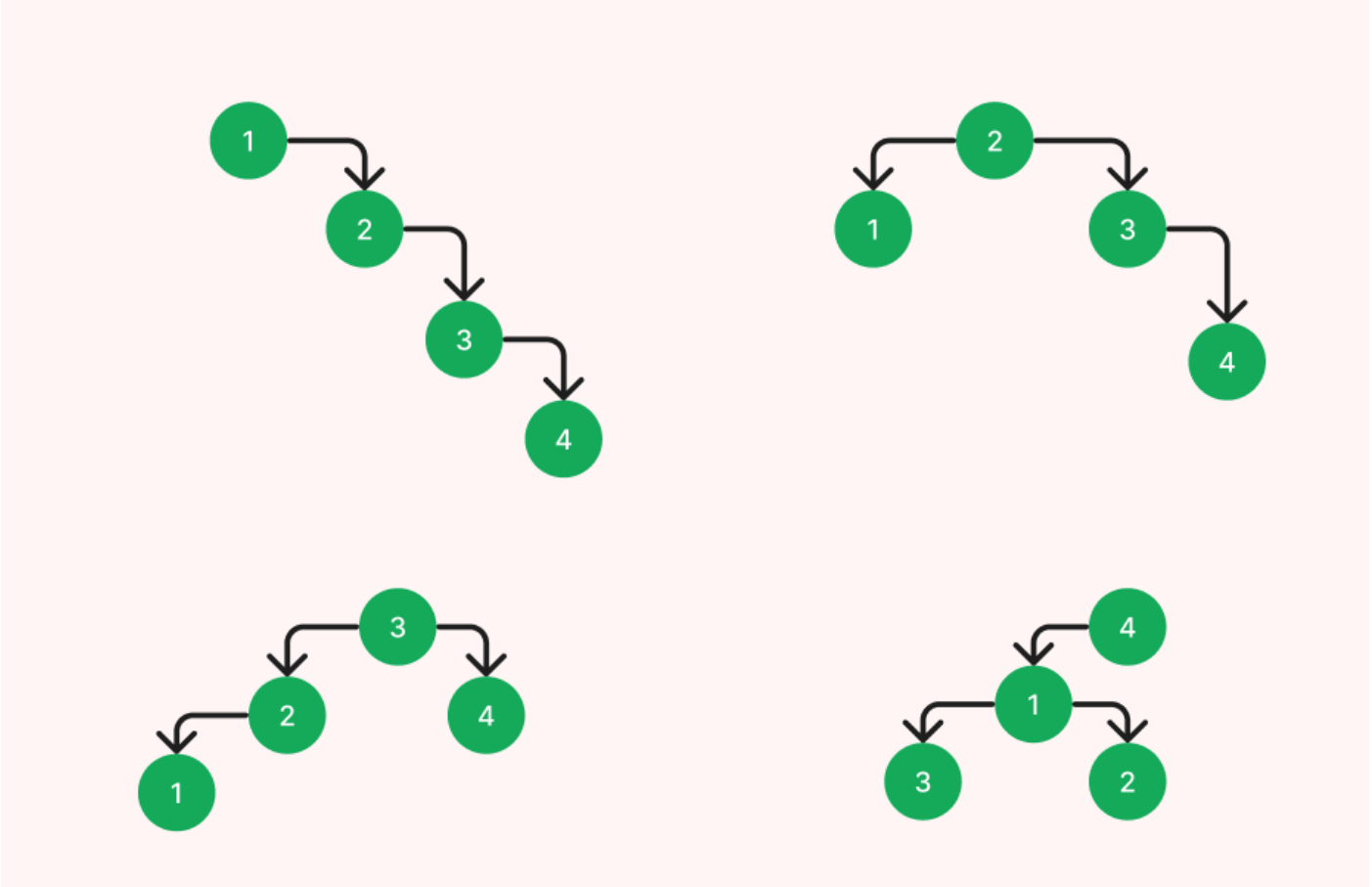
Поиск элемента в списке

Список может быть реализован на основе различных структур данных, таких как массивы или связные списки. При использовании связных списков, особенно двусвязных, обеспечивается более гибкая работа с данными и динамическое управление элементами.

```
class Node:
    def __init__(self, data):
        self.data = data
        self.prev = None
        self.next = None

    def __name__(self):
        prev = self.prev.data if self.prev != None else 'None'

        next = self.next.data if self.next != None else 'None'

        s = f"""
            | Предыдущий | Текущий | Следующий|
            |----------------------------------|
            | {prev} | {self.data} | {next} |"""

        return s



class DoublyLinkedList:
    def __init__(self):
        self.head = None

    def insert_at_end(self, data):
        new_node = Node(data)
        if self.head is None:
            self.head = new_node
        else:
            current = self.head
            while current.next:
                current = current.next
            current.next = new_node
            new_node.prev = current

    def insert_at_start(self, data):
        new_node = Node(data)
        if self.head is None:
            self.head = new_node
        else:
            # Начальный узел
            current = self.head
            current.prev = new_node
            new_node.next = current
```

```
            self.head = new_node

    def display(self):
        current = self.head
        while current:
            print(current.__name__())
            print(f"""
                ||
                \/""")
            current = current.next

# Пример использования двусвязного списка
dll = DoublyLinkedList()
dll.insert_at_end(1)
dll.insert_at_end(2)
dll.insert_at_end(3)

dll.insert_at_start(4)

dll.display()
```

10. Нарисуйте все возможные деревья двоичного поиска для элементов 1, 2, 3, 4. **ЕСТЬ В БИЛЕТАХ**



Далее вытекающие из правила остальные деревья. (правый элемент узла больше, левый меньше)

11. АТД - очередь. Реализация с помощью указателей. **ЕСТЬ В БИЛЕТАХ**

АТД - очередь - это абстрактный тип данных, представляющий собой упорядоченную коллекцию элементов, в которой элементы добавляются в конец очереди, а извлекаются из

начала. Очередь подчиняется принципу FIFO (First-In-First-Out) - первым пришел, первым вышел.
Основные операции над очередью:
Enqueue - добавление элемента в конец очереди
Dequeue - извлечение и удаление элемента из начала очереди
IsEmpty - проверка, пуста ли очередь

Очередь может быть реализована с помощью различных структур данных, например, массива или связного списка. Реализация на Python с использованием связного списка может выглядеть следующим образом:

```
class Node:
    def __init__(self, data):
        self.data = data
        self.next = None

class Queue:
    def __init__(self):
        self.front = self.rear = None

    def isEmpty(self):
        return self.front == None

    def enqueue(self, data):
        newNode = Node(data)
        if self.rear == None:
            self.front = self.rear = newNode
        else:
            self.rear.next = newNode
            self.rear = newNode

    def dequeue(self):
        if self.isEmpty():
            return
        else:
            dequeueNode = self.front
            self.front = self.front.next
            if self.front == None:
                self.rear = None
            return dequeueNode.data

    def display(self):
        temp = self.front
        while temp:
            print(temp.data)
            temp = temp.next

q = Queue()
q.enqueue(10)
q.enqueue(20)
q.enqueue(30)
q.display()  # Output: 10 20 30
print(q.dequeue())  # Output: 10
q.display()  # Output: 20 30
```

12. Преобразуйте выражение ((a+b)+c*(d+e)+f)*(g+h) в постфиксную форму
ЕСТЬ В БИЛЕТАХ

```
((a+b)+c*(d+e)+f)*(g+h)
(a+b)+c*(d+e)+f (g+h) *
(a+b)+c*(d+e) f + (g+h) *
(a+b) c*(d+e) + f + (g+h) *
(a+b) c (d+e) * + f + (g+h) *
a b + c d e + * + f + g h + *
```

https://timeweb.cloud/blog/poryadok-vypolneniya-operacij-v-programmirovanii

Вопросы ТМО (Теория массового обслуживания)

1. Предмет теории массового обслуживания. Основные понятия теории. НЕТ В БИЛЕТАХ

Теория массового обслуживания - это математический инструмент для анализа и оптимизации систем обслуживания, которые обрабатывают потоки заявок. Она изучает процессы поступления, обработки и выхода заявок из системы, а также оценивает характеристики эффективности обслуживания, такие как время ожидания, загруженность устройств и вероятности отказа.

Основные задачи теории массового обслуживания включают анализ стационарного и непрерывного потока заявок, определение оптимального количества устройств обслуживания и оптимальных стратегий обработки заявок, а также прогнозирование производительности системы в зависимости от технических параметров.

1. Анализ стационарного и непрерывного потока заявок:

- Стационарный поток заявок означает, что его характеристики (такие как интенсивность потока или распределение межприбытий) остаются постоянными со временем. Анализ стационарного потока позволяет предсказать общую производительность системы и оценить временные характеристики, например, среднее время ожидания.

- Непрерывный поток заявок обычно предполагает изменяющиеся характеристики по времени. Анализ непрерывного потока требует применения более сложных методов стохастического моделирования, чтобы учесть временные изменения и прогнозировать долгосрочные тенденции.

2. Определение оптимального количества устройств обслуживания и оптимальных стратегий обработки заявок:

- Оптимальное количество устройств обслуживания определяется с учетом баланса между экономическими затратами на оборудование и желаемым уровнем обслуживания. Анализ позволяет определить такое количество устройств, при котором достигается оптимальное соотношение между производительностью и эффективностью.

- Оптимальные стратегии обработки заявок определяются с учетом характеристик потока и целей системы. Например, выбор правила обслуживания (например, FIFO или приоритет по типу заявки) может быть подобран исходя из конкретных требований к системе обслуживания.

3. Прогнозирование производительности системы в зависимости от технических параметров:

- Прогнозирование производительности системы позволяет предсказать ее поведение при изменении различных технических параметров, таких как интенсивность потока, время обслуживания, количество устройств обслуживания и другие. Это позволяет оценить, как система будет работать в различных условиях и принимать решения об оптимизации.

Методы теории массового обслуживания могут быть применены в различных областях, таких как телекоммуникации, транспорт, информационные технологии, здравоохранение, финансы и другие сферы, где важным является эффективное управление потоками заявок.

Теория массового обслуживания включает в себя такие основные понятия, как потоки заявок, устройства обслуживания, параметры времени ожидания и вероятности обслуживания. Она предоставляет инструменты для моделирования и анализа систем обслуживания в различных условиях и способствует разработке стратегий оптимизации для улучшения эффективности обслуживания.

1. Потоки заявок:

- Интенсивность потока (λ) - это среднее число заявок, поступающих в систему за единицу времени. Может быть постоянной или переменной.

- Распределение межприбытий (inter-arrival time distribution) - описывает интервал времени между поступлением двух последовательных заявок.

- Вид потока (дискретный/непрерывный) - поток может быть дискретным, если заявки приходят в дискретные моменты времени, или непрерывным, если заявки могут приходить в любое время.

2. Устройства обслуживания:

- Серверы - физические или программные устройства, обслуживающие поступившие заявки.

- Каналы обслуживания - количество параллельных потоков заявок, которые могут быть одновременно обслуживаемы в системе.

- Время обслуживания (service time) - время, затрачиваемое на обработку заявки устройством обслуживания.

3. Системы массового обслуживания:

- Очереди (Queues) - упорядоченный список заявок, ожидающих обслуживания.

- Правила обслуживания (Service discipline) - определяют, какая заявка будет обслужена следующей из очереди, например, по принципу FIFO (первым пришел, первым обслужен) или по приоритету.

4. Время ожидания:

- Среднее время ожидания - среднее время, которое заявка проводит в очереди в ожидании обслуживания.

- Вероятность отказа - вероятность того, что заявка не будет обслужена из-за переполненности системы.

2. Классификация систем массового обслуживания. **НЕТ В БИЛЕТАХ**

Системы массового обслуживания важны для моделирования и анализа различных процессов, таких как обслуживание клиентов в банках, аэропортах, обработка запросов в компьютерных системах и т.д. Классификация СМО может быть проведена по нескольким параметрам.

1. Тип обслуживания:

- Одноприборные СМО: в таких системах каждое поступившее требование обслуживается одним обслуживающим устройством. Например, это может быть один сервер, обрабатывающий запросы от клиентов.

- Многоприборные СМО: в этих системах каждое требование может быть обработано несколькими обслуживающими устройствами параллельно. Такое решение позволяет распределять нагрузку и повышать эффективность обслуживания.

2. Распределение времени обслуживания:

- Экспоненциальное: это одно из самых распространенных распределений времени обслуживания в системах массового обслуживания. Оно характеризуется тем, что вероятность того, что требование будет обслужено за определенный промежуток времени, убывает экспоненциально.

- Обратное гамма-распределение: в отличие от экспоненциального, обратное гамма-распределение характеризуется возможностью длительных времен обслуживания, что может быть важно для некоторых типов систем.

3. Количество обслуживающих устройств:

- Одноканальные СМО: такие системы имеют только одно обслуживающее устройство. Например, это может быть один сервер, обрабатывающий все поступающие запросы.

- Многоканальные СМО: в таких системах несколько параллельно работающих обслуживающих устройств позволяют обрабатывать несколько запросов одновременно, что увеличивает общую производительность.

4. Дисциплина обслуживания:

- FIFO (First In, First Out): это общая дисциплина, при которой требования обслуживаются в порядке их поступления.

- LIFO (Last In, First Out): построение системы обслуживания на принципе обслуживания последних поступивших требований первыми. Это может быть применимо в некоторых специфических случаях.

- Приоритетная дисциплина: такая дисциплина позволяет устанавливать приоритеты для обслуживания требований в зависимости от их важности или характеристик.

5. Тип потока заявок:

- Однородный: все поступающие требования имеют одинаковый тип и статистические характеристики. Например, это может быть случайный поток запросов клиентов на обслуживание.

- Неоднородный: поступающие требования различаются по типу и статистическим характеристикам. Это могут быть различные типы запросов с разной продолжительностью обслуживания или разной важностью.

3. Задача минимизации штрафа за задержку обслуживания. НЕТ В БИЛЕТАХ

4. Задача «директора» (задача одного станка). НЕТ В БИЛЕТАХ

5. Задача двух станков. Алгоритм Джонсона. НЕТ В БИЛЕТАХ

6. Потоки событий. Простейший поток событий. НЕТ В БИЛЕТАХ

1. Потоки событий: Потоки событий представляют собой последовательность событий, которые происходят в определенном порядке, и могут быть обработаны программой или системой. Каждое событие может содержать информацию о времени его возникновения, типе события и других атрибутах.

Пример: Веб-приложение, которое отслеживает действия пользователей, может генерировать поток событий для каждого клика, наведения курсора, отправки формы и т.д.

2. Простейший поток событий: Простейший поток событий представляет собой базовый способ отслеживания событий в программе. Этот тип потока событий может быть реализован с помощью цикла, который ожидает и обрабатывает поступающие события.

Пример: Небольшая программа обработки текста может иметь простейший поток событий, который ожидает ввода пользователя, затем обрабатывает его и выводит результат на экран.

7. Случайные процессы. Марковский процесс. НЕТ В БИЛЕТАХ

1. Случайные процессы - это математические модели, которые описывают эволюцию случайных величин во времени или пространстве. Это может быть как случайное изменение величины во времени, так и случайное изменение величины в пространстве. Примером случайного процесса может быть модель изменения цены акции на фондовом рынке в течение дня, где цена изменяется случайным образом в каждый момент времени.

2. Марковский процесс - это случайный процесс, для которого выполняется свойство Маркова, то есть будущее состояние зависит только от текущего состояния и не зависит от прошлого. Это свойство называется отсутствием памяти. Примером Марковского процесса может быть модель погоды, где завтрашняя погода зависит только от текущей погоды, и не зависит от погоды в предыдущие дни.

8. Уравнения Колмогорова. Предельные вероятности. НЕТ В БИЛЕТАХ

Уравнения Колмогорова — это система дифференциальных уравнений, которые описывают динамику вероятностных процессов, таких как случайные блуждания, марковские процессы и другие случайные системы.

Уравнения Колмогорова имеют важное значение для анализа и прогнозирования случайных процессов. Они позволяют моделировать и изучать вероятностные системы, описывая их эволюцию во времени и прогнозируя их будущее состояние на основе начальных условий.

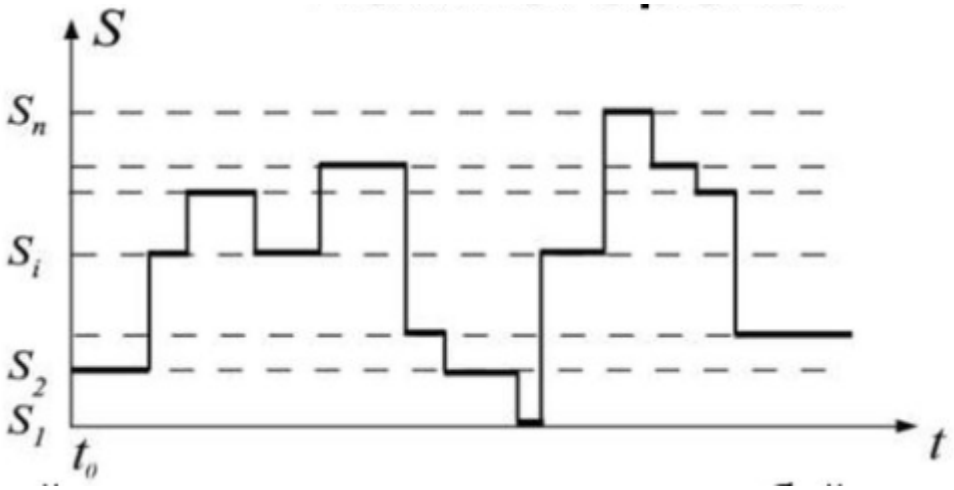


Схема случайного процесса представляет собой ступенчатую кривую, на рис. изображен один из возможных вариантов реализаций процесса.

Для любого момента времени t вероятность состояний есть P1(t), P2(t), …Pi(t),… Pn(t), при этом соблюдается условие нормировки.

$$\sum_{i=1}^{n} P_i(t) = 1$$

Чтобы найти все вероятности состояний , необходимо решить систему дифференциальных уравнений Колмогорова.

$$\frac{dP_i(t)}{dt} = \sum_{j=1}^{n} \lambda_{ij} P_j(t) - P_j(t) \sum_{j=1}^{n} \lambda_{ij}$$

Предельные вероятности - это вероятности событий, которые стремятся к определенным значениям при увеличении числа испытаний. Например, если мы бросаем монету много раз, предельная вероятность выпадения орла будет стремиться к 0.5, а выпадения решки – к 0.5.

Пример: если мы бросаем кубик много раз, то предельная вероятность выпадения каждой из шести граней будет стремиться к 1/6, так как у нас есть равные шансы на выпадение каждой грани при большом числе бросков.

9. Процесс гибели и размножения. Формулы для предельных вероятностей. НЕТ В БИЛЕТАХ

В ТМО широко распространен специальный класс случайных процессов – так называемые процессы гибели и размножения. Название это связано с рядом биологических задач, где этот процесс служит математической моделью изменения численности биологических популяций.

Особенность такой системы заключается в том, что переходы могут осуществляться из любого состояния только в соседние состояния, т.е. из состояния Sk возможен переход только в сотояния Sk-1 или Sk+1.

Составим и решим СЛАУ для предельных вероятностей состояний (а их существование вытекает из возможности перехода из каждого состояния в каждое другое за конечное число шагов).

Для S_0: $\lambda_{01} P_0 = \lambda_{10} P_1$. (1)

Для S_1: $(\lambda_{10} + \lambda_{12}) P_1 = \lambda_{21} P_2 + \lambda_{01} P_0$. (2)

С учетом этого: $\lambda_{12} P_1 = \lambda_{21} P_2$. (3)

Аналогично, записывая уравнения для предельных вероятностей других состояний, можно получить следующую систему уравнений:

$$\begin{cases} \lambda_{01}P_0 = \lambda_{10}P_1, \\ \lambda_{12}P_1 = \lambda_{21}P_2, \\ \dots \\ \lambda_{k-1,k}P_{k-1} = \lambda_{k,k-1}P_k \\ \dots \\ \lambda_{n-1,n}P_{n-1} = \lambda_{n,n-1}P_n, \end{cases}$$

к которой добавляется нормировочное уравнение: P0+P1+...+Pn = 1

Решим эту систему:

$$P_1 = \frac{\lambda_{01}P_0}{\lambda_{10}},$$

$$P_2 = \frac{\lambda_{12}P_1}{\lambda_{21}} = \frac{\lambda_{12}\lambda_{01}}{\lambda_{21}\lambda_{10}}P_0,$$

$$P_3 = \frac{\lambda_{23}\lambda_{12}\lambda_{01}}{\lambda_{32}\lambda_{21}\lambda_{10}}P_0,$$

$$\dots$$

$$P_k = \frac{\lambda_{k-1,k}\dots\lambda_{12}\lambda_{01}}{\lambda_{k,k-1}\dots\lambda_{21}\lambda_{10}}P_0$$

Обратим внимание на формулы для Pi: числители представляют собой произведения всех интенсивностей, стоящих у стрелок, ведущих слева направо (от начала до данного состояния Si); знаменатели – произведения всех интенсивностей, стоящих у стрелок, ведущих справа налево (из состояния Si до начала).

Таким образом, все вероятности состояний Pi выражены через одну из них (через вероятноять P0). Подставим эти выражения в нормировочное условие и получим:

$$P_0 = \left(1 + \frac{\lambda_{01}}{\lambda_{10}} + \frac{\lambda_{12}\lambda_{01}}{\lambda_{21}\lambda_{10}} + \dots + \frac{\lambda_{n-1,n}\dots\lambda_{12}\lambda_{01}}{\lambda_{n,n-1}\dots\lambda_{21}\lambda_{10}}\right)^{-1}$$

.

10. Одноканальная СМО с отказами. Расчет показателей эффективности. НЕТ В БИЛЕТАХ

Одноканальная (СМО) с отказами – это модель, которая описывает процесс обслуживания заявок (или заявителей) в одной очереди, где может происходить отказ в обслуживании. Заявки поступают на обслуживание в систему и могут быть обслужены или отклонены в зависимости от различных факторов.

Представим, что у нас есть одно устройство (сервер) для обслуживания заявок. Заявки поступают в систему, и если сервер свободен, заявка будет обслужена. Однако, если сервер занят обслуживанием другой заявки, возникает отказ в обслуживании, и заявка должна быть либо вынуждена ждать, либо покидает систему.

№	Термин	Обозначение
1	Интенсивность входящего потока заявок	λ
2	Интенсивность выходящего потока обслуженных заявок	μ
3	Приведенная интенсивность потока заявок	ρ
4	Среднее время обслуживания заявки	$\bar{t}_{serv}$
5	Относительная пропускная способность СМО	q
6	Абсолютная пропускная способность СМО	A
7	Вероятность того, что заявка будет обслужена	P_{serv}
8	Вероятность того, что заявка получит отказ	P_{otk}

Постановка задачи

Параметры λ и μ известны.

Требуется найти $\bar{t}_{serv}$, ρ, q, A, P_{serv}, P_{otk}.

Формулы для расчетов

В теории массового обслуживания доказывается, что показатели эффективности одноканальной СМО с отказами вычисляются по следующим формулам:

$$\bar{t}_{serv} = \frac{1}{\mu}, \quad (4.1)$$

$$\rho = \frac{\lambda}{\mu}, \quad (4.2)$$

$$q = \frac{1}{\rho+1}, \quad (4.3)$$

$$A = \lambda q, \quad (4.4)$$

$$P_{serv} = q, \quad (4.5)$$

$$P_{otk} = 1 - P_{serv}. \quad (4.6)$$

~~11. Многоканальная СМО с отказами. Расчет показателей эффективности.~~ ~~НЕТ В БИЛЕТАХ~~

~~Многоканальная система массового обслуживания (СМО) с отказами представляет собой модель, в которой есть несколько серверов (каналов) для обслуживания заявок, и при этом возможны отказы в обслуживании. Это контекст, в котором множество заявок поступает на обслуживание в систему, где они могут быть обслужены одним из доступных серверов или отклонены в случае, если все серверы заняты.~~

~~Рассмотрим классическую задачу Эрланга. Имеется n каналов, на которые поступает поток заявок с интенсивностью λ. Поток обслуживаний имеет интенсивность μ. Найти предельные вероятности состояний системы и показатели ее эффективности. Система S (СМО) имеет следующие состояния (нумеруем их по числу заявок, находящихся в системе): S0, S1, S2, …, Sk, …, Sn, где Sk — состояние системы, когда в ней находится k заявок, т.е. занято k каналов. Граф состояний СМО соответствует процессу гибели и размножения и показан на рис. 1.~~

~~Рисунок – 1.~~

~~Поток заявок последовательно переводит систему из любого левого состояния в соседнее правое с одной и той же интенсивностью λ. Интенсивность же потока обслуживаний, переводящих систему из любого правого состояния в соседнее левое состояние, постоянно меняется в зависимости от состояния. Действительно, если СМО находится в состоянии S2 (два канала заняты), то она может перейти в состояние S1 (один канал занят), когда закончит обслуживание либо первый, либо второй канал, т.е. суммарная интенсивность их потоков обслуживании будет 2μ. Аналогично суммарный поток обслуживаний, переводящий СМО из состояния S3 (три канала заняты) в S2, будет иметь интенсивность 3μ, т.е. может освободиться любой из трех каналов и т.д.~~

№	Термин	Обозначение
1	Число каналов обслуживания	$n \ (n>1)$
2	Интенсивность входящего потока заявок	λ
3	Интенсивность потока обслуженных заявок, выходящего *из одного канала*	μ
4	Приведенная интенсивность потока заявок	ρ
5	Вероятность того, что занято 0, 1, …, n каналов, соответственно	$p_0, p_1, \ldots, p_n$
6	Относительная пропускная способность СМО	q
7	Абсолютная пропускная способность СМО	A
8	Вероятность того, что заявка будет обслужена	P_{serv}
9	Вероятность того, что заявка получит отказ	P_{otk}
10	Среднее число занятых каналов	$\bar{k}$

~~Рассчет~~

Параметры n, λ и μ известны.

Требуется найти ρ, $p_0, p_1, \ldots, p_n$, P_{otk}, q, A, P_{serv}, $\bar{k}$.

Формулы для расчетов

Приведенная интенсивность потока заявок вычисляется по формуле

$$\rho = \frac{\lambda}{\mu}. \qquad (5.1)$$

Вероятности $p_0, p_1, \ldots, p_n$ вычисляются по *формулам Эрланга*:

$$\begin{cases} p_0 = \left(\sum\limits_{k=0}^{k=n} \frac{\rho^k}{k!} \right)^{-1}, \\ p_k = \frac{\rho^k}{k!} \cdot p_0, \ k = 1,2,\ldots,n. \end{cases} \qquad (5.2)$$

Вероятности $p_0, p_1, \ldots, p_n$ вычисляются по *формулам Эрланга*:

$$\begin{cases} p_0 = \left(\sum\limits_{k=0}^{k=n} \frac{\rho^k}{k!} \right)^{-1}, \\ p_k = \frac{\rho^k}{k!} \cdot p_0, \ k = 1,2,\ldots,n. \end{cases} \qquad (5.2)$$

Поскольку заявка получает отказ, если все каналы обслуживания заняты, то

$$P_{\text{otk}} = p_n = \frac{\rho^n}{n!} \cdot p_0. \qquad (5.3)$$

Кроме того,

$$q = 1 - P_{\text{otk}} = 1 - \frac{\rho^n}{n!} \cdot p_0, \qquad (5.4)$$

$$A = \lambda q = \lambda \left(1 - \frac{\rho^n}{n!} \cdot p_0 \right), \qquad (5.5)$$

$$P_{\text{serv}} = 1 - P_{\text{otk}} = q, \qquad (5.6)$$

$$\bar{k} = \frac{A}{\mu} = \rho \cdot \left(1 - \frac{\rho^n}{n!} \cdot p_0\right).$$

~~12. Одноканальная СМО с неограниченной очередью. Формулы Литтла. НЕТ В БИЛЕТАХ~~

~~13. Многоканальная СМО с неограниченной очередью. НЕТ В БИЛЕТАХ~~

~~14. Одноканальная СМО с ограниченной очередью. НЕТ В БИЛЕТАХ~~

~~15. Многоканальная СМО с ограниченной очередью. НЕТ В БИЛЕТАХ~~

~~Практические задания:~~ ~~НЕТ В БИЛЕТАХ~~

~~1. Написать программу для решения задачи минимизации суммарного штрафа.~~

~~2. Написать программу для решения задачи «директора» (одного станка).~~

~~3. Написать программу для решения задачи двух станков~~

Объект смерти (ЕСТЬ В БИЛЕТАХ)

Вопросы СИСА (Сетевое и системное администрирование)

Полезные ссылки: https://www.youtube.com/@merionacademy

1. Объясните основные принципы работы протокола TCP/IP. НЕТ В БИЛЕТАХ
(ответ от Perplexity)

Протокол TCP/IP (Transmission Control Protocol/Internet Protocol) является основой сетевой связи в интернете и обеспечивает передачу данных между устройствами в сети. Основные принципы работы протокола TCP/IP включают:

Стек протоколов: TCP/IP состоит из четырех уровней - сетевого, транспортного, сеансового и прикладного. Каждый уровень отвечает за определенные функции передачи данных и обеспечивает их надежность и целостность

IP-адресация: Протокол IP (Internet Protocol) назначает уникальные IP-адреса устройствам в сети для их идентификации и маршрутизации данных. IP-адресация позволяет точно доставить пакеты данных от отправителя к получателю

Маршрутизация: TCP/IP определяет правила маршрутизации данных между различными сетями и устройствами. Маршрутизация осуществляется путем обращения к определенным IP-адресам с использованием маски

Транспортный уровень: На транспортном уровне протоколы TCP и UDP обеспечивают передачу данных между приложениями. TCP гарантирует доставку данных в правильном порядке и проверяет их целостность, в то время как UDP подходит для отправки небольших сообщений, где не критичны потери данных

Надежность передачи данных: TCP обеспечивает надежную передачу данных, проверяя доставку всех пакетов адресату и повторно запрашивая данные в случае ошибок. Это позволяет гарантировать целостность и правильный порядок данных при передаче

2. Какие основные задачи выполняет DHCP сервер в сетях? НЕТ В БИЛЕТАХ
(ответ от Perplexity)

DHCP-сервер выполняет следующие основные задачи в сетях

- Выдача IP-адресов клиентам. DHCP-сервер динамически распределяет IP-адреса клиентам в сети, обеспечивая им инструмент для подключения к сети
- Настройка сетевых параметров. Помимо IP-адреса, DHCP-сервер может выдавать клиентам другие параметры сетевой конфигурации, такие как маска подсети, шлюз по умолчанию, адреса DNS-серверов и т.д.
- Автоматизация процесса назначения IP-адресов. DHCP позволяет автоматизировать процесс назначения IP-адресов, избавляя администраторов от необходимости настраивать адреса вручную на каждом клиентском устройстве
- Управление пулом IP-адресов. DHCP-сервер управляет пулом доступных IP-адресов, выдавая их клиентам по запросу и возвращая адреса обратно в пул по истечении срока аренды
- Обеспечение уникальности IP-адресов. DHCP гарантирует, что каждому клиенту будет назначен уникальный IP-адрес, предотвращая конфликты адресов в сети

3. Что такое DNS и какова его роль в сетевом администрировании? НЕТ В БИЛЕТАХ
(ответ от Perplexity)

DNS (Domain Name System) - это система доменных имен, которая преобразует человеко-читаемые доменные имена в IP-адреса, необходимые для идентификации устройств в сети

Основная роль DNS в сетевом администрировании включает:

Идентификация ресурсов: DNS позволяет привязывать уникальные доменные имена к соответствующим IP-адресам, обеспечивая удобный доступ к ресурсам в сети

Упрощение навигации: Благодаря DNS пользователи могут использовать понятные и запоминающиеся доменные имена вместо сложных IP-адресов для доступа к веб-сайтам и другим ресурсам в интернете

Распределение запросов: DNS распределяет запросы на доменные имена между различными серверами, обеспечивая балансировку нагрузки и эффективное функционирование сети

Обеспечение безопасности: DNS играет важную роль в обеспечении безопасности сети, предотвращая атаки, такие как DNS-спуфинг и DNS-отравление, и обеспечивая защиту от угроз безопасности

Управление доменами: DNS позволяет администраторам управлять доменными именами, добавлять новые записи, изменять настройки и обеспечивать правильную работу сети

4. Какие преимущества и недостатки имеют статические и динамические IP-адреса? НЕТ В БИЛЕТАХ

Преимущества статического IP-адреса:

Постоянный адрес, который не меняется, что удобно для доступа к серверам и удаленного управления

Не требует дополнительных настроек для доступа, так как адрес известен заранее

Позволяет настроить надежную связь между устройствами

Недостатки статического IP-адреса:

Более дорогой, так как выделяется постоянно, в отличие от динамического

Сложнее в настройке, требует ручной конфигурации на устройстве и маршрутизаторе

Если адрес станет известен злоумышленникам, это может создать угрозу безопасности

Преимущества динамического IP-адреса:

Дешевле, так как выделяется только на время сессии из пула адресов

Автоматически назначается DHCP-сервером, не требует ручной настройки

Если адрес станет известен злоумышленникам, его можно сменить

Недостатки динамического IP-адреса:

Адрес может меняться при каждом подключении, что неудобно для доступа к серверам

Требует дополнительных настроек для доступа, так как адрес неизвестен заранее

Если адрес часто меняется, это может создавать проблемы для некоторых приложений

5. Что такое VLAN, и как оно используется для управления сетевым трафиком? НЕТ В БИЛЕТАХ

VLAN (Virtual Local Area Network) - это технология, позволяющая разделить физичес

Основные цели использования VLAN:

- Повышение безопасности и управляемости сети. Хосты в разных VLAN не могут напрямую обмениваться трафиком, что ограничивает распространение вирусов и несанкционированный доступ.Сокращение широковещательного трафика
- Широковещательные пакеты ограничиваются рамками VLAN, что улучшает производительность сети.
- Гибкость в группировке устройств. Устройства в одном VLAN могут находиться в разных физических местах, группировка определяется логически – по отделам, проектам и т.д.

Принцип работы VLAN:

- Коммутатор помечает кадры VLAN-тегом (802.1Q), содержащим ID VLAN
- Коммутатор передает кадр только портам того же VLAN, что и источник

- Для связи между VLAN требуется маршрутизатор

Существует два основных типа VLAN:

- Port-based VLAN - порты коммутатора статически назначаются VLAN
- Tagged VLAN - коммутатор использует 802.1Q теги для динамического определения VLAN

6. Расскажите о принципах работы протоколов TCP и UDP и в каких случаях их следует применять. НЕТ В БИЛЕТАХ

Протоколы TCP (Transmission Control Protocol) и UDP (User Datagram Protocol) - это два основных протокола транспортного уровня модели TCP/IP, которые обеспечивают передачу данных в сети

TCP

TCP - это протокол, который обеспечивает надежную передачу данных, гарантируя доставку пакетов в том порядке, в котором они были отправлены. Он использует механизм подтверждения получения пакетов, чтобы убедиться, что данные были доставлены целыми и без ошибок. TCP также обеспечивает упорядоченную доставку данных, что означает, что пакеты будут доставлены в том порядке, в котором они были отправлены

UDP

UDP - это протокол, который обеспечивает быструю передачу данных, но не гарантирует доставку пакетов. Он не использует механизм подтверждения получения пакетов, что делает его более быстрым, но менее надежным, чем TCP. UDP не обеспечивает упорядоченную доставку данных, что означает, что пакеты могут быть доставлены в любом порядке

Основными отличиями между TCP и UDP являются:

- Надежность: TCP обеспечивает надежную передачу данных, в то время как UDP не гарантирует доставку пакетов.
- Скорость: UDP является более быстрым, чем TCP, из-за меньшей сложности его алгоритмов.
- Упорядоченность: TCP обеспечивает упорядоченную доставку данных, в то время как UDP не обеспечивает упорядоченную доставку.

TCP следует применять в случаях, когда важна надежность передачи данных, например:

- Файловый обмен
- Электронная почта
- Веб-серфинг

UDP следует применять в случаях, когда важна скорость передачи данных, например:

- Онлайн-игры
- Видеотрансляции
- VoIP

7. Каковы основные принципы работы межсетевого экрана (firewall) и его роли в сетевой безопасности? НЕТ В БИЛЕТАХ

Межсетевой экран (Firewall) - это программно-аппаратный комплекс, предназначенный для контроля и фильтрации сетевого трафика на основе заданных правил безопасности

Принципы работы межсетевого экрана:

- Фильтрация трафика: межсетевой экран контролирует и фильтрует трафик, проходящий через него, основываясь на заданных правилах и политиках безопасности
- Блокирование нежелательного трафика: он блокирует доступ к нежелательным или потенциально опасным ресурсам, предотвращая угрозы безопасности

- Мониторинг сетевой активности: межсетевой экран анализирует сетевую активность, обнаруживает аномалии и предотвращает атаки на сеть
- Управление доступом: он управляет доступом к сетевым ресурсам, определяя, какие устройства и приложения имеют право на доступ к определенным ресурсам

Роль межсетевого экрана в сетевой безопасности:

- Защита периметра сети: межсетевой экран защищает периметр сети, контролируя входящий и исходящий трафик, предотвращая несанкционированный доступ к сети
- Предотвращение вторжений: он обнаруживает и блокирует попытки вторжения и атаки на сеть, обеспечивая безопасность сетевых ресурсов
- Фильтрация трафика: межсетевой экран фильтрует трафик на основе заданных правил, блокируя вредоносные пакеты и защищая сеть от угроз
- Контроль доступа: он управляет доступом к сетевым ресурсам, обеспечивая конфиденциальность и целостность данных

8. Какие существуют методы обеспечения безопасности в беспроводных сетях? НЕТ В БИЛЕТАХ

Шифрование трафика

Использование протоколов шифрования, таких как WPA2 или WPA3, позволяет защитить передаваемые данные от перехвата злоумышленниками. Шифрование гарантирует, что даже если сигнал будет пойман, он будет нечитаемым.

Фильтрация по MAC-адресам

Межсетевой экран может фильтровать трафик, пропуская только устройства с разрешенными MAC-адресами. Это ограничивает доступ к сети посторонних устройств.

Использование виртуальных локальных сетей (VLAN)

Разделение беспроводной сети на несколько логических сегментов (VLAN) позволяет ограничить распространение широковещательного трафика и повысить безопасность. Устройства в разных VLAN не могут напрямую обмениваться трафиком.

Скрытие SSID

Отключение трансляции SSID (названия беспроводной сети) делает сеть менее заметной для посторонних. Однако это не является надежной защитой, так как SSID можно легко узнать другими способами.

Использование межсетевых экранов

Межсетевые экраны (firewall) контролируют и фильтруют трафик, проходящий через них, блокируя подозрительную активность и предотвращая несанкционированный доступ. Они позволяют гибко настраивать правила безопасности.

Регулярное обновление ПО

Своевременное обновление программного обеспечения точек доступа и клиентских устройств важно для устранения уязвимостей и повышения защищенности беспроводной сети

9. Объясните, как работает протокол SSL/TLS в контексте безопасности сетевого взаимодействия. НЕТ В БИЛЕТАХ

Протокол SSL/TLS (Secure Sockets Layer/Transport Layer Security) обеспечивает безопасность сетевого взаимодействия путем шифрования данных и аутентификации участников соединения

1. Установка соединения

Клиент (например, браузер) инициирует соединение с сервером по защищенному протоколу HTTPS (порт 443). Начинается процесс SSL/TLS-рукопожатия для согласования параметров шифрования.

2. SSL/TLS-рукопожатие

- Клиент и сервер обмениваются случайными данными для генерации ключей шифрования
- Сервер отправляет свой SSL-сертификат, содержащий открытый ключ

- Клиент проверяет подлинность сертификата и отправляет зашифрованным открытым ключом сервера сеансовый ключ

3. Шифрование трафика
   - Клиент и сервер используют сеансовый ключ для симметричного шифрования всего последующего трафика
   - Это обеспечивает конфиденциальность передаваемых данных, таких как данные кредитных карт

4. Аутентификация
   - SSL/TLS гарантирует, что клиент общается именно с тем сервером, с которым намеревался
   - Сертификат сервера подтверждает его подлинность и принадлежность владельцу

5. Целостность данных
   - Используются коды аутентичности сообщений (MAC), чтобы обнаруживать любые изменения в переданных данных
   - Это предотвращает атаки "человек посередине" и подделку сообщений

10. Что такое сетевые протоколы ICMP и IGMP, и для чего они используются? НЕТ В БИЛЕТАХ

ICMP (Internet Control Message Protocol) и IGMP (Internet Group Management Protocol) - это два важных сетевых протокола, используемых в стеке TCP/IP

ICMP (Internet Control Message Protocol)

ICMP используется для передачи диагностической информации об ошибках и состоянии IP-соединений.

Основные функции ICMP:

- Доставка сообщений об ошибках, например, недоступность узла или порта
- Диагностика сети с помощью утилит ping и traceroute
- Передача отчетов о недоступности устройств в сети

ICMP работает на сетевом уровне, инкапсулируясь в IP-пакеты, но не передает данные пользователя

IGMP (Internet Group Management Protocol)

IGMP используется для организации сетевых устройств в группы с помощью маршрутизатора. Он позволяет:

- Управлять членством хостов в IP-мультикаст группах
- Передавать данные от сервера к множеству клиентов, принимающих видеотрансляцию

IGMP работает над IP, обеспечивая сообщение IP-хостами соседним маршрутизаторам о своем членстве в группах.

11. Какие протоколы используются для маршрутизации в сетях? Объясните принцип их работы. НЕТ В БИЛЕТАХ

Дистанционно-векторные протоколы

- RIP (Routing Information Protocol) - один из старейших протоколов маршрутизации, использующий алгоритм Беллмана-Форда
- IGRP (Interior Gateway Routing Protocol) - проприетарный протокол Cisco, похожий на RIP

Протоколы состояния каналов связи

- OSPF (Open Shortest Path First) - внутренний протокол маршрутизации, использующий алгоритм Дейкстры для поиска кратчайшего пути

- IS-IS (Intermediate System to Intermediate System) - протокол маршрутизации, похожий на OSPF

Протоколы для междоменной маршрутизации

- EGP (Exterior Gateway Protocol) - один из первых протоколов междоменной маршрутизации, в настоящее время практически не используется
- BGP (Border Gateway Protocol) - основной протокол междоменной маршрутизации в Интернет, использует атрибуты маршрутов для выбора оптимального пути

Протоколы для беспроводных сенсорных сетей

- AODV (Ad hoc On-Demand Distance Vector) - протокол маршрутизации для самоорганизующихся беспроводных сетей, использует алгоритм DVA
- DSR (Dynamic Source Routing) — протокол реактивной маршрутизации для ad-hoc сетей

12. Какие протоколы используются для обеспечения безопасности на уровне сетевого соединения (VPN)? НЕТ В БИЛЕТАХ

IPSec (Internet Protocol Security)

IPSec - это набор протоколов, обеспечивающих шифрование и аутентификацию на сетевом уровне. Он создает защищенный туннель между устройствами, гарантируя конфиденциальность передаваемых данных.

L2TP (Layer 2 Tunneling Protocol)

L2TP используется для туннелирования данных на канальном уровне. Он не обеспечивает шифрование, поэтому часто применяется в связке с IPSec для повышения безопасности.

PPTP (Point-to-Point Tunneling Protocol)

PPTP - один из старейших протоколов для создания VPN-соединений. Он обеспечивает туннелирование на канальном уровне, но использует устаревшие алгоритмы шифрования, поэтому считается небезопасным.

OpenVPN

OpenVPN - это протокол с открытым исходным кодом, использующий SSL/TLS для создания защищенных туннелей/ Он поддерживает различные алгоритмы шифрования и аутентификации, обеспечивая высокий уровень безопасности.

IKEv2 (Internet Key Exchange version 2)

IKEv2 - это протокол, используемый в связке с IPSec для создания безопасных VPN-соединений. Он отличается высокой скоростью и стабильностью работы, особенно на мобильных устройствах.

13. Как происходит обеспечение отказоустойчивости в сетевых системах? НЕТ В БИЛЕТАХ

- Избыточность компонентов: Один из ключевых аспектов отказоустойчивой системы — наличие резервных компонентов для любой части системы. Это позволяет системе продолжать работу даже при отказе одного из элементов.
- Резервирование и репликация данных: Резервирование и репликация дисков позволяют избежать сбоев системы в случае отказа части внутренних дисков. Также важно резервирование внешних сетевых соединений и внутренних соединений сети хранения данных.
- Устойчивость к катастрофам: Для обеспечения отказоустойчивости системы важно иметь резервные компоненты, способные принять всю нагрузку в случае отказа основных элементов. Это позволяет избежать полного отключения системы в случае катастрофических событий.

- Архитектурный подход: Проектирование системы с учетом архитектурного подхода к обеспечению устойчивости позволяет смягчить воздействие неблагоприятных факторов и оставаться надежной для конечного пользователя. Этот подход включает в себя определение местоположения каждой роли и сохранение состояния для обработки данных.

14. Какие основные шаги необходимо предпринять при планировании и внедрении нового сервера в корпоративной сети? НЕТ В БИЛЕТАХ

- Определение требований и целей: Первым шагом является определение требований к новому серверу и целей его внедрения. Необходимо понять, какие функции сервер должен выполнять и какие задачи должен решать.
- Выбор аппаратного обеспечения: На основе выявленных требований следует выбрать подходящее аппаратное обеспечение для сервера, учитывая производительность, объем хранилища, память и другие характеристики.
- Выбор программного обеспечения: После выбора аппаратного обеспечения необходимо определить необходимое программное обеспечение, такое как операционная система, прикладное ПО, антивирусные программы и т.д.
- Планирование сетевой интеграции: Следующим шагом является планирование интеграции нового сервера в корпоративную сеть. Это включает настройку сетевых параметров, безопасности, доступа и т.д.
- Установка и настройка сервера: После получения оборудования и программного обеспечения необходимо установить и настроить сервер в соответствии с требованиями и целями проекта.
- Тестирование и отладка: После установки и настройки сервера следует провести тестирование его работоспособности, а также отладку для выявления и устранения возможных проблем.
- Документирование и обучение персонала: Не менее важным шагом является документирование процесса внедрения нового сервера и обучение персонала по его использованию и обслуживанию.
- Мониторинг и поддержка: После внедрения сервера необходимо обеспечить его мониторинг, регулярное обновление и поддержку для обеспечения его надежной и эффективной работы в корпоративной сети.

15. Какие сетевые протоколы применяются для мониторинга и управления сетевыми устройствами (SNMP, NetFlow и т.д.)? НЕТ В БИЛЕТАХ

- SNMP (Simple Network Management Protocol): SNMP - стандартный протокол управления сетью, используемый для мониторинга и управления сетевыми устройствами. Он позволяет сетевым администраторам удаленно управлять и контролировать сетевые устройства, собирать данные для устранения неполадок, отслеживания трафика и мониторинга производительности.
- NetFlow: NetFlow – технология, позволяющая собирать информацию о трафике в сети и анализировать его для оптимизации сетевой производительности/ Она предоставляет данные о трафике, источниках и назначениях пакетов, а также об объеме переданных данных.
- WMI (Windows Management Instrumentation): WMI - технология управления Windows, предоставляющая единый интерфейс для запроса данных и выполнения действий над управляемыми объектами. Она позволяет приложениям и сценариям запрашивать данные и выполнять действия над объектами.

Практика СИСА: НЕТ В БИЛЕТАХ

1. Настройка маршрутизатора для связи между двумя локальными сетями. (По предложенной топологии)

~~2. Создание и настройка виртуального интерфейса VLAN на коммутаторе. (По предложенной топологии)~~

~~3. Установка и настройка инструментов удаленного управления (SSH) на сетевом устройстве.~~

~~4. Создание DHCP-пула и настройка параметров выдачи IP-адресов на маршрутизаторе. (По предложенной топологии)~~

~~5. Включите протокол маршрутизации OSPF на маршрутизаторе и коммутаторе и обеспечьте связность. По предложенной топологии)~~

~~6. Включите протокол маршрутизации BGP на маршрутизаторе и коммутаторе и обеспечьте связность. По предложенной топологии)~~

Вопросы ОС (Операционные системы)

Всё это есть в билетах

1. Организация параллельной работы устройств ввода-вывода и процессора. https://studfile.net/preview/2949233/page:3/

Эволюция ввода – вывода

1. Процессор непосредственно управляет периферийным устройством.

2. Устройство управляется контроллером. Процессор использует программируемый ввод - вывод без прерываний (переход к абстракции интерфейса ввода - вывода) – режим обмена с опросом готовности.

3. Использование контроллера прерываний. Ввод-вывод, управляемый прерываниями – режим обмена с прерываниями.

4. Использование модуля (канала) прямого доступа к памяти. Перемещение данных в память (из нее) без использования процессора.

5. Использование отдельного специализированного процессора ввода-вывода, управляемого центральным процессором.

6. Использование отдельного компьютера для управления устройствами ввода-вывода при минимальном вмешательстве центрального процессора.

1. Программируемый ввод-вывод без прерываний

Процессор посылает необходимые команды контроллеру устройства ввода-вывода и переводит процесс в состояние ожидания завершения операции ввода-вывода. Контроллер транслирует принятые команды в сигналы управления устройством ввода вывода. После выполнения команды устройством его контроллер выдает *сигнал готовности*. Процессор (через драйвер устройства) периодически проверяет состояние устройства ввода вывода с целью проверки завершения операций ввода вывода и готовности к принятию новой команды

Недостаток: большие потери процессорного времени

2. Ввод-вывод, управляемый прерываниями.

Процессор посылает необходимые команды контроллеру ввода-вывода и продолжает выполнять процесс, если нет необходимости в ожидании выполнения операции. В противном случае процесс приостанавливается до получения сигнала прерывания о завершении ввода/вывода, а процессор переключается на выполнение другого процесса. В конце каждого цикла выполненных команд процессор проверяет наличие прерываний.

Режим обмена с прерываниями по своей сути является режимом асинхронного управления. Для того чтобы не потерять связь с устройством, может быть запущен отсчет времени, в течение которого устройство обязательно должно выполнить команду и выдать-таки сигнал запроса на прерывание.

Если это время истекло после выдачи устройству очередной команды, а устройство так и не ответило, то делается вывод о том, что связь с устройством потеряна и управлять им больше нет возможности. Пользователь и/или задача получают соответствующее диагностическое сообщение.

Драйверы, работающие в режиме прерываний, представляют собой сложный комплекс программных модулей и могут иметь несколько секций: *секцию запуска*, одну или несколько секций *продолжения секцию завершения*.

Секция запуска инициирует операцию ввода-вывода. Эта секция запускается для включения устройства ввода-вывода или просто для инициализации очередной операции ввода-вывода.

Секция продолжения(их может быть несколько, если алгоритм управления обменом данными сложный, и требуется несколько прерываний для выполнения одной логической операции) осуществляет основную работу по передаче данных.

Секция завершения обычно выключает устройство ввода-вывода или просто завершает операцию.

3. Прямой доступ к памяти (dma).

Модуль прямого доступа к памяти управляет обменом данных между основной памятью и контроллером ввода-вывода. Процессор посылает запрос на передачу блока данных модулю прямого доступа к памяти, а прерывание происходит только после передачи всего блока данных.

DMA-контроллер имеет доступ к системной шине независимо от центрального процессора. Контроллер содержит несколько регистров, доступных центральному процессору для чтения и записи (регистр адреса памяти, счетчик байтов, управляющие регистры).

Перед выполнением операции обмена ЦП программирует DMA-контроллер, устанавливая его регистры (шаг 1).

DMA-контроллер начинает перенос данных, посылая дисковому контроллеру по шине запрос чтения (шаг 2). Адрес памяти уже находится на адресной шине, так что контроллер знает, куда пересылать следующее слово из своего буфера.

Запись в память является еще одним стандартным циклом шины (шаг 3). Когда запись закончена, контроллер диска посылает сигнал подтверждения контролеру DMA(шаг 4).

Затем контролер DMA увеличивает используемый адрес памяти и уменьшает значение счетчика байтов.

После этого шаги 2, 3 и 4 повторяются, пока значение счетчика не станет равным нулю. По завершении цикла копирования контроллер DMA инициирует прерывание процессора, сообщая ему о завершении операции ввода-вывода.

Одним из способов организации параллельной работы устройств ввода-вывода и процессора является применение асинхронной передачи данных. При этом данные передаются без привязки к тактовым импульсам процессора, что позволяет выполнять другие операции во время передачи данных. Для этого устройства ввода-вывода обеспечиваются специальными буферами, которые хранят данные до тех пор, пока они не будут обработаны процессором.

Еще одним способом параллельной работы является использование DMA (Direct Memory Access) – прямого доступа к памяти. В этом случае устройство ввода-вывода имеет свой собственный контроллер DMA, который может напрямую обращаться к памяти и осуществлять передачу данных без участия процессора. По завершении передачи, контроллер DMA посылает процессору соответствующий сигнал, и процессор уже обрабатывает данные.

Также можно использовать прерывания – специальные сигналы, посылаемые устройствами ввода-вывода процессору для активации определенной программы-обработчика. При получении прерывания процессор сохраняет текущее состояние, прерывает текущую операцию и выполняет программу-обработчик. После завершения обработки процессор возвращает сохраненное состояние и продолжает выполнение прерванной операции.

Таким образом, существует несколько способов организации параллельной работы устройств ввода-вывода и процессора, и выбор конкретного способа зависит от требований к системе и особенностей используемых устройств.

2. Структура ОС UNIX. Особенности функционирования.

ОС UNIX имеет следующую структуру:

1. Ядро (kernel) - основной компонент ОС, управляющий работой всех других компонентов, включая управление памятью, процессами, драйверами устройств и т.д.

2. Системные программы (system utilities) - набор инструментов для управления ОС, таких как командная оболочка, утилиты для работы с файлами и каталогами, сетевыми настройками и т.д.

3. Библиотеки (libraries) - набор программных модулей, используемых другими приложениями для выполнения различных задач, таких как работа с графическими интерфейсами или кодированием данных.

4. Прикладное ПО (applications) - приложения, которые запускаются пользователем для выполнения различных задач, таких как обработка текстовых документов, проигрывание медиафайлов, работа с базами данных и т.д.

Одной из особенностей функционирования ОС UNIX является использование текстовых конфигурационных файлов для настройки системы. Кроме того, ОС UNIX представляет собой многозадачную систему, то есть на ней могут работать множество приложений одновременно и одновременно выполнять множество задач. Также ОС UNIX имеет открытый исходный код, что позволяет разработчикам создавать собственные приложения и утилиты для работы с системой.

3. Назначение и функции операционных систем (ОС).

Операционные системы (ОС) - это программное обеспечение, управляющее аппаратными ресурсами компьютера и обеспечивающее запуск и работу других приложений.

Назначение ОС:

1. Управление ресурсами компьютера. ОС управляет доступом к ресурсам компьютера, включая процессор, RAM, жесткий диск, периферийные устройства.

2. Обеспечение безопасности. ОС обеспечивает безопасность данных, приложений и системы в целом. Она управляет доступом пользователей к файлам и другим ресурсам и защищает от вирусов и других вредоносных программ.

3. Обеспечение интерфейса пользователя. ОС обеспечивает пользовательский интерфейс для управления компьютером. Это может быть графический интерфейс или интерфейс командной строки.

4. Запуск и управление приложениями. ОС позволяет запускать и управлять работой приложений.

5. Управление сетевыми ресурсами. ОС управляет доступом к сетевым ресурсам и обеспечивает сетевую связь между компьютерами.

Функции ОС:

1. Управление процессами: ОС управляет процессами, запущенными на компьютере, распределяя ресурсы между ними.

2. Управление памятью: ОС контролирует доступ к памяти и распределяет ее между приложениями.

3. Управление файлами: ОС обеспечивает доступ и управление файлами на жестком диске и других устройствах.

4. Управление устройствами: ОС управляет работой устройств компьютера, таких как принтеры, сканеры, клавиатуры и т.д.

5. Управление безопасностью: ОС обеспечивает защиту компьютера от вирусов и других вредоносных программ и контролирует доступ к ресурсам компьютера.

6. Управление сетью: ОС позволяет подключаться к сети и использовать сетевые ресурсы, такие как файлы и принтеры.

4. Архитектурные особенности операционных систем.

Операционные системы имеют следующие архитектурные особенности:

1. Многозадачность. Операционные системы позволяют запускать несколько приложений одновременно и переключаться между ними.

2. Многопоточность. Операционные системы поддерживают выполнение нескольких потоков в рамках одного приложения.

3. Виртуальная память. Операционные системы могут создавать виртуальное адресное пространство для приложений и управлять доступом к физической памяти компьютера.

4. Разделение ресурсов. Операционные системы контролируют доступ к аппаратным ресурсам компьютера, разделяя их между приложениями.

5. Иерархическая структура файловой системы. Операционные системы используют иерархическую структуру файловой системы для организации файлов и директорий.

6. Конкурентность. Операционные системы обеспечивают конкурентное выполнение процессов или потоков, используя механизмы синхронизации и взаимодействия.

7. Модульность. Операционные системы состоят из различных модулей, каждый из которых выполняет определенные функции.

8. Абстракции. Операционные системы скрывают сложность аппаратных ресурсов от пользователей и приложений, предоставляя им простой интерфейс взаимодействия.

9. Поддержка сетевых протоколов. Операционные системы обеспечивают поддержку сетевых протоколов и позволяют компьютерам взаимодействовать друг с другом.

10. Обнаружение и исправление ошибок. Операционные системы обнаруживают и исправляют ошибки, возникающие в работе приложений и системы в целом, чтобы обеспечить стабильную и безопасную работу компьютера.

5. Понятие базовой ЭВМ. Ее характеристика и принцип построения. Принцип обработки команд на примере базовой ЭВМ.

Базовая ЭВМ – это компьютер, способный обрабатывать данные и выполнять программы. Ее характеристики включают в себя: центральный процессор (CPU), оперативную память (RAM), жесткий диск (HDD), блок питания и системную плату.

Принцип построения базовой ЭВМ заключается в том, что информация хранится в оперативной памяти и используется центральным процессором для обработки команд. ЦП обрабатывает данные, заключенные в оперативной памяти, и отправляет результаты в оперативную память.

Принцип обработки команд на примере базовой ЭВМ можно проиллюстрировать следующим образом. В начале процесса ЦП считывает команду из оперативной памяти, которая указывает, какую операцию нужно выполнить. Затем ЦП определяет адреса байтов операндов и считывает их из оперативной памяти. Затем ЦП выполняет операцию над операндами и сохраняет результат в оперативной памяти.

Таким образом, базовая ЭВМ использует принцип обработки команд, который позволяет выполнять задачи, используя информацию в оперативной памяти.

6. Цели и задачи файловой системы. Примеры ФС и их особенности.

Цели файловой системы:

1. Обеспечение хранения и организации файлов на устройстве хранения.

2. Управление доступом к файлам и папкам, ограничение доступа пользователей и программ.

3. Обеспечение безопасности и сохранности файлов, в том числе защита от потери данных.

4. Ускорение работы с файлами и устройствами хранения.

Задачи файловой системы:

1. Разбиение устройства хранения на блоки и организация связи между ними.

2. Предоставление интерфейса для работы с файлами и папками, включая возможность создания, удаления, копирования и перемещения файлов.

3. Реализация механизмов управления доступом к файлам и папкам.

4. Обеспечение безопасности файлов, включая защиту от ошибок работы программ и от злоумышленных действий.

Примеры файловых систем и их особенности:

1. FAT (File Allocation Table) – простая файловая система, используемая в ОС DOS и Windows. Основной недостаток – низкая эффективность использования дискового пространства.

2. NTFS (New Technology File System) – файловая система, используемая в Windows. Позволяет управлять доступом и защитить файлы с помощью атрибутов безопасности файлов.

3. EXT (Extended File System) – файловая система, используемая в ОС Linux. Обеспечивает высокую эффективность использования дискового пространства.

4. APFS (Apple File System) – файловая система, используемая в ОС macOS, iOS, iPadOS и watchOS. Обеспечивает высокую производительность и защиту данных, поддерживает шифрование файлов и снимки системы.

5. ZFS (Zettabyte File System) – мощная файловая система, используемая в ОС FreeBSD, OpenSolaris и Linux. Позволяет выполнять снимки файловой системы и восстановление данных, обеспечивает высокую надежность хранения данных.

7. Понятие системы команд в ЭВМ. Классификация команд и их формат. Понятие адресации команд и данных в ЭВМ. Виды адресаций команд.

https://studme.org/93890/informatika/adresatsiya_komand_dannyh
https://studfile.net/preview/7497320/page:6/

Система команд - это совокупность инструкций, которые могут быть выполнены ЭВМ, и способ, которым эти инструкции кодируются и интерпретируются.

Команды в ЭВМ классифицируются по их функциональному назначению и типу операций, которые они выполняют. Основные типы команд включают в себя:

1) Арифметические команды: выполняют математические операции, такие как сложение, вычитание, умножение и деление.

2) Логические (битовые) команды: выполняют логические операции, такие как сравнение, AND, OR и NOT.

3) Команды управления: управляют потоком выполнения программы, такие как переходы, вызовы подпрограмм и возвраты.

4) Команды ввода-вывода: управляют передачей данных между ЭВМ и внешними устройствами, такими как клавиатура, монитор и накопители.

5) Команды управления памятью: управляют доступом к памяти, такие как загрузка, сохранения.

Формат команд в ЭВМ обычно включает в себя следующие поля:

Опкод (операционный код): код операции, которая должна быть выполнена.

Адреса операндов: адреса или значения операндов, участвующих в операции.

Модификатор: указывает на тип адресации или другие модификации операции.

Флаги: указывают на условия выполнения операции или ее результат.

Пример формата команды: OPCODE ADDR1 ADDR2 MODIFIER FLAGS

Адресация команд и данных в ЭВМ - это способ, которым компьютер определяет местоположение данных и данных в памяти для их выполнения или обработки.

Адресация команд. При выполнении линейной части программы, когда команды расположены в памяти одна за другой, для их выборки из памяти используется программный счетчик (указатель команд), т.е. адресация полностью возлагается на аппаратные средства и адрес следующей команды в текущей команде не указывается. Необходимость указания адреса следующей команды в текущей возникает при передаче управления подпрограмме (процедуре), расположенной в другом месте памяти.

Адресация данных. В описании каждой команды должны быть указаны адреса мест расположения исходных операндов (источников), задействованных при выполнении операции, и адрес места расположения операнда результата операции (приемника).

Виды адресаций команд делятся на прямые и непрямые:

Прямые -

1) Неявная адресация. В таких командах явного адресного поля нет (нуль адресная команда). Операнд задается кодом операции.

2) Непосредственная адресация. В адресном поле фактически указывается не адресный код, а сам операнд. Этот способ не требует дополнительных обращений к памяти за операндами, но адресное поле должно иметь длину операнда.

3) Абсолютная (прямая) адресация – характеризуется тем, что в адресном поле, задается полный адрес памяти, где хранится операнд.

Непрямые -

1) Базирование (относительная адресация). Процедура формирования исполнительного адреса: Аисп = Абаза + <смещение>. Для реализации этого способа в ЭВМ выделяются специальные ячейки, которые выполняют функции базовых регистров.

2) Косвенная адресация. При косвенной адресации адресный код в команде содержит не адрес самого операнда, а содержит адрес памяти, где хранится адрес операнда.

3) Автоинкрементная, автодекрементная (индексная) адресация. Предусматривают автоматическое изменение адреса после каждой операции, облегчая работу с памятью

4) Укороченная адресация – всевозможные способы, ориентированные на уменьшение длины команды за счет сокращения адресного кода.

5) Стековая адресация. При использовании стековой адресации, команды не имеют адресного поля (безадресные) для задания адресов операндов, текущий адрес ячейки памяти хранится в указателе стека.

8. Виды интерфейсов операционных систем ОС.

1. Графический интерфейс (GUI) - это наиболее распространенный и удобный тип интерфейса ОС, который позволяет пользователю взаимодействовать с системой с помощью графических элементов, таких как кнопки, меню, окна и иконки.

2. Консольный интерфейс (CLI) - это тип интерфейса, который позволяет пользователю взаимодействовать с системой через текстовое поле (командную строку), где пользователь вводит команды для выполнения основных операций.

3. Интерфейс командной строки (CLI) - это тип интерфейса, который позволяет пользователю взаимодействовать с системой через текстовое поле (командную строку), где пользователь вводит команды для выполнения основных операций.

4. Интерфейс управления задачами (Task Manager) - это тип интерфейса, который позволяет пользователям контролировать и управлять процессами, которые выполняются на компьютере.

5. Веб-интерфейс - это тип интерфейса, который позволяет пользователям взаимодействовать с системой через веб-браузер. Это позволяет пользователям управлять своими компьютерами удаленно и выполнять различные задачи с помощью веб-приложений.

Интерфейсы операционных систем можно классифицировать по различным признакам. Вот некоторые из основных видов интерфейсов:

1. **Командная строка (CLI - Command Line Interface): Пользователь взаимодействует с ОС через текстовые команды, которые вводятся в командной строке. Это один из самых старых типов интерфейсов, который до сих пор используется из-за своей гибкости и мощи, особенно в серверных ОС и среди профессиональных пользователей.**
2. **Графический интерфейс пользователя (GUI - Graphical User Interface): Это интерфейс, который позволяет пользователям взаимодействовать с компьютером через графические элементы, такие как окна, иконки и кнопки. GUI сделал компьютеры более доступными для**

широкой публики и является стандартным интерфейсом для большинства современных ОС.

3. **Сенсорный интерфейс (TUI - Touch User Interface): Интерфейс, оптимизированный для сенсорного управления, часто используется в мобильных устройствах, таких как смартфоны и планшеты.**
4. **Голосовой интерфейс (VUI - Voice User Interface): Позволяет пользователю взаимодействовать с системой с помощью голосовых команд. Примеры включают в себя голосовых ассистентов, таких как Siri, Alexa и Google Assistant.**
5. **Жестовый интерфейс: Интерфейс, который позволяет управлять устройством с помощью жестов, обычно с использованием специальных датчиков или камер.**
6. **Текстовый пользовательский интерфейс (TUI - Text-based User Interface): Похож на CLI, но представляет собой текстовый интерфейс с элементами меню, кнопками и формами, которые можно навигировать с помощью клавиатуры.**

9. Понятие процесса в операционных системах.

Процесс в операционных системах – это программа, которая выполняется на компьютере. Каждый процесс имеет свой уникальный идентификатор, набор ресурсов и состояний. Он может занимать определенное количество оперативной памяти, использовать процессор и другие ресурсы системы.

Процессы создаются операционной системой для выполнения задач. Они могут быть запущены как пользователем, так и системными процессами. Операционная система управляет процессами, определяет их приоритеты и распределяет ресурсы между ними.

Каждый процесс может быть в следующих состояниях:

- Готов: процесс находится в очереди на выполнение, но ещё не получил доступ к процессору.
- Выполняется: процесс получил доступ к процессору и выполняется.
- Заблокирован: процесс ожидает, пока выполнится какое-то событие (например, чтение из файла).
- Завершён: процесс завершил выполнение.

Управление процессами – это одна из ключевых функций операционной системы, поскольку она позволяет эффективно использовать ресурсы компьютера и обеспечивать стабильную работу системы.

10. Прерывания. Порядок их обработки.

Прерывания – это механизм, который позволяет компьютеру остановить выполнение текущей задачи и переключиться на обработку другой задачи, которая имеет более высокий приоритет или которая требует немедленного внимания. Различные устройства, такие как клавиатура, мышь, жесткий диск, могут генерировать прерывания для уведомления процессора о том, что они нуждаются в обработке.

Порядок обработки прерываний определяется иерархией приоритетов. Каждое прерывание имеет свой уровень приоритета, и обработчик прерывания должен выполнить действия, связанные с ним, до того, как перейти к нижестоящим уровням приоритета.

Процессор начинает обработку прерывания, проверяя вектор прерывания, который указывает на адрес начала обработчика прерывания. Обработчик прерывания выполняет необходимые операции, сохраняя при этом состояние регистров и стека, чтобы можно было вернуться к выполнению прерванной операции после завершения обработки прерывания.

После завершения обработки прерывания процессор восстанавливает регистры и стек и возобновляет выполнение прерванной операции.

11. Ядро ОС UNIX. Управление процессами.

Ядро ОС UNIX занимается управлением процессами, которые являются основными задачами, выполняемыми пользователем. Процесс - это экземпляр программы, выполняемый в операционной системе. Каждый процесс имеет свой уникальный идентификатор (PID) и состоит из кода программы и данных, которые используются программой.

Ядро ОС UNIX управляет процессами с помощью планировщика, который определяет, какие процессы получат доступ к ЦП в следующий раз. Процессы могут быть запущены в одном из нескольких различных состояний: ожидание, готовность, выполнение и завершение. Когда процесс завершается, ядро освобождает все его ресурсы.

Кроме того, ядро ОС UNIX предоставляет механизмы для синхронизации доступа к разделяемым ресурсам, таким как файлы и устройства ввода-вывода. Это обеспечивает защиту данных от несанкционированного доступа и конфликтов при их одновременном доступе.

Для управления процессами, ядро ОС UNIX предоставляет ряд команд, которые могут быть использованы для запуска, приостановки, возобновления и прерывания процессов. Некоторые из этих команд включают ps, kill, nice, renice и другие.

12. Синхронизация процессов и потоков.

Синхронизация процессов и потоков - это процесс упорядочивания доступа к общим ресурсам между несколькими процессами или потоками.

В многопоточных и многопроцессных приложениях существует необходимость в использовании общих ресурсов, таких как память, файлы, сетевые соединения и другие. В этом случае один поток или процесс может использовать ресурс одновременно с другими потоками и процессами. Различные потоки или процессы могут конкурировать за доступ к общим ресурсам, что может привести к неправильной работе приложения, ошибкам или даже к его сбою.

Синхронизация позволяет избежать этих проблем путем регулирования доступа к общим ресурсам. Для этого используются механизмы синхронизации, такие как мьютексы, семафоры, критические секции и другие.

В целом, синхронизация процессов и потоков является важной частью многопоточных и многопроцессных приложений, которая позволяет обеспечить безопасный и эффективный доступ к общим ресурсам.

Вопросы ВП (Визуальное программирование)

Все вопросы я когда то подготовил в этом файле Вопросы на экзамен.docx

Всё это есть в билетах

1. Преимущества и недостатки визуального программирования.

Преимущества:

1) Графическое представление легче для понимания, чем текстовый вариант

2) Низкий порог вхождения. Интерфейс чаще всего простой и понятный, что даёт возможность легко освоить начинающему программисту.

3) Более быстрая разработка.

Недостатки:

1) Ограниченный функционал. Платформы визуального программирования ограничены в своих функциях для более масштабных и более сложных проектов.

2) Медленная скорость работы по сравнению с классическим написанием кода.

3) Требуют больше памяти, т.к. используют графику.

2. Визуальное программирование как способ создания программ для ЭВМ путем манипулирования графическими объектами вместо написания текста. Приведите примеры, как визуальное программирование реализовано в Unity.

В Unity визуальное программирование реализовано в “Bolt Visual Scripting”. После скачивания у объекта можно добавить в инспекторе компонент “Flow machine”. После этого создаём Macro и нажимаем Edit Graph. Далее можно соединять необходимые методы и компоненты, а также управлять состояниями объекта через граф. Например: Update -> Input Get Key Down -> Transform Rotate (yAngle = 10)

3. Примеры графических и визуальных языков программирования.

1) Scratch

2) Blockly - Разрабатывается и поддерживается компанией Google с 2012 года. Свободно распространяется вместе с исходным кодом по лицензии Apache 2.0. Целевой аудиторией проекта являются программисты, разрабатывающие веб-приложения, включающие Блокли, в основном для учебных целей.

3) AgentCubes - это образовательный язык программирования для детей, позволяющий создавать 3D и 2D онлайн-игры и симуляции. Основное применение AgentCubes — это инструмент вычислительного мышления, обучающий детей вычислительному мышлению посредством разработки игр и симуляций на основе учебной программы масштабируемого игрового дизайна.

4. Unity Bolt для визуального программирования. Принцип применения, ограничения и возможности.

В Unity визуальное программирование реализовано в “Bolt Visual Scripting”. После скачивания у объекта можно добавить в инспекторе компонент “Flow machine”. После этого создаём Macro и нажимаем Edit Graph. Далее можно соединять необходимые методы и компоненты. И управлять состояниями объекта

через граф. Например: Update -> Input Get Key Down -> Transform Rotate (yAngle = 10)

Из минусов можно отметить, что чем больше разрастается граф, тем более запутаннее будет визуальное отображение. Слабая оптимизация.

Возможности: более быстрая разработка скриптов по сравнению с написанием кода в C#.

5. Фундаментальные ограничения визуального программирования.

1) Ограничения визуального интерфейса могут запутывать разработчика даже больше, чем текст.

2) С повышением сложности программ программист начинает заниматься абстракцией и снижением связности, и уровень программиста во многом определяется тем, насколько удачно это получилось. Визуальные средства редко имеют развитую поддержку данного процесса.

3) Для текстового представления в настоящее время существует множество инструментов: системы управления версиями, автодополнение, которых нет у визуального программирования.

6. Примеры визуальных сред разработки.

Scratch - это визуальная среда программирования, разработанная MIT Media Lab для обучения основам программирования детей и подростков. В Scratch программы создаются путем соединения цветных блоков, представляющих различные команды и операции.

Qt Designer - это кроссплатформенная свободная среда для разработки графических интерфейсов программ, использующих библиотеку Qt. Он входит в состав Qt framework и предоставляет инструменты для проектирования и создания графических пользовательских интерфейсов (GUI) из компонентов Qt. Qt Designer позволяет создавать и настраивать пользовательские интерфейсы, используя drag-and-drop метод для размещения виджетов на форме и устанавливая их свойства и сигналы без необходимости написания кода вручную.

HiAsm — это конструктор, который позволяет «написать» (а вернее — собрать) программу без знаний языка.

С его помощью можно создавать программы для любой платформы, например, Windows, CNET, WEB, QT и других. Также HiAsm работает с библиотекой OpenGL, что даёт возможность создавать графические объекты.

Достоинства:

- возможность установки дополнений;
- кроссплатформенность;
- интуитивно понятный интерфейс;
- высокая скорость выполнения;
- официальная версия на русском языке.

Недостатки:

- не подходит для крупных проектов;
- большой объём исполняемых файлов.

C++ Builder - это интегрированная среда разработки (IDE) для создания приложений на языке программирования C++. Основные характеристики C++ Builder:

Позволяет быстро разрабатывать приложения с графическим интерфейсом пользователя (GUI) для различных платформ, включая Windows, macOS, iOS и Android.
Использует визуальный подход к разработке, предоставляя инструменты для drag-and-drop размещения компонентов интерфейса и настройки их свойств.
Поддерживает разработку кросс-платформенных приложений с использованием единой базы кода C++.
Включает в себя редактор кода, отладчик, инструменты доступа к базам данных и другие необходимые средства для разработки.
Доступен в бесплатной версии Community Edition для индивидуальных разработчиков, студентов и некоммерческих организаций.
Таким образом, C++ Builder - это мощная IDE, ориентированная на быструю разработку приложений с графическим интерфейсом на языке C++, поддерживающая кросс-платформенную разработку и предоставляющая широкий набор инструментов для повышения производительности программистов.

ПРАКТИКА ВП:

Всё это есть в билетах
ВНИМАНИЕ !!!! ВОТ ССЫЛКА НА ПРОЕКТ ТЫК

1.Реализовать механику “тряпичная кукла” (Ragdoll) в Unity:

a. **Создание тряпичной куклы на базе модели с сайта Mixamo.com;**
b. **Создание перехода между анимацией и Ragdoll;**

Сцена Ragdoll. Папка Ragdoll

2. Реализовать взаимодействие объектов в Unity при помощи методов OnCollisionEnter, OnCollisionStay, OnCollisionExit. Привести описание работы этих методов. Привести пример реализации для каждого метода. Описать, какие свойства Properties есть у метода OnCollisionEnter и как эти свойства можно использовать. Реализовать механику стрельбы по объектам.

Сцена Collision, Папка Collision, внутри папки есть объяснение.txt

3.Продемонстрировать корректное приложение сил в Unity. AddForce(). В каком методе будет корректно реализовать приложение сил с помощью GetKey(), а в каком GetKeyDown();
a) Что произойдет, если GetKeyDown() обрабатывать в методе FixedUpdate?
b) AddForce() следует обрабатывать в Update() или в FixedUpdate()? Дайте развернутый аргументированный ответ.

Сцена GetKey, Папка GetKey, обьяс.txt

4. Реализовать воспроизведение звука при падении объекта в Unity. Громкость воспроизведения должна зависеть от силы удара объекта SoundBox.

Сцена Sound, Папка Sound

Другие вопросы

Всё это есть в билетах

ПРОГРАММИРОВАНИЕ

1. Сгенерируйте массив 5x5 случайных чисел от 1 до 20 и все числа от 3 до 8 замените на 0.

```
import random

array = [[random.randint(1, 20) for _ in range(5)] for _ in range(5)]

print('Исходная матрица:', array)

for i in range(5):
    for j in range(5):
        if array[i][j] >= 3 and array[i][j] <= 8:
            array[i][j] = 0

print('Измененная матрица:', array)
```

2. Сгенерировать массив 5x5 случайных целых чисел от 1 до 10. Посчитайте, сколько раз каждое из этих чисел повторяется в массиве.

```
import numpy as np

mass = np.random.randint(1, 10, size=(5, 5))
print('Массив: \n', mass)
freq = {n: 0 for n in range(1, 11)}
for i in range(5):
   for j in range(5):
       freq[mass[i, j]] += 1
print('Частота:', freq)
```

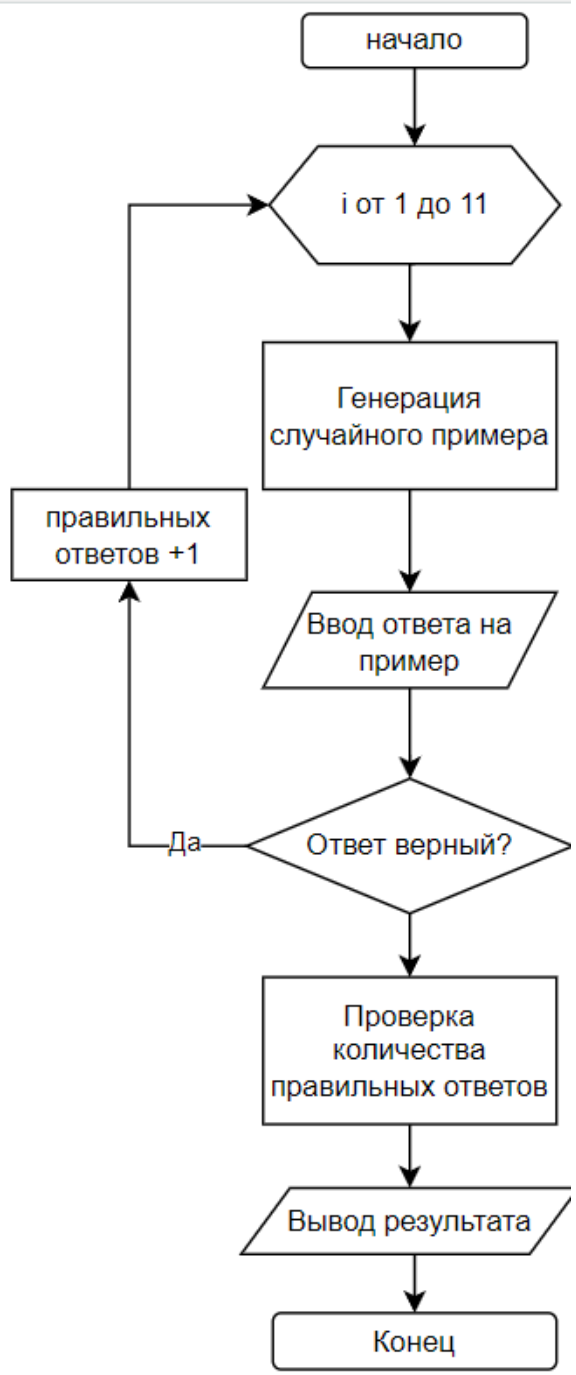
```
import random

array = [[random.randint(1, 10) for _ in range(5)] for _ in range(5)]

all_numbers = {i: 0 for i in range(1, 11)}

for i in range(5):
    for j in range(5):
        all_numbers[array[i][j]] +=1
```

```
print("Исходная матрица", array)
print("Количество элементов", all_numbers)
```

3. Поменяйте в словаре d = {1: '5', 2: '6', 3: '7', 4: '8'} ключи и значения местами. Выведите итоговый словарь на экран.

```
d = {1: '5', 2: '6', 3: '7', 4: '8'}

d_keys_last = list(d.keys())

for key in d_keys_last:
    # Меняем ключи и значения местами
    d[d[key]] = key
    # Удаляем предыдущий ключ
    d.pop(key)


print('d: ', d)
```

4. Составьте блок-схему и напишите программу тестовой проверки знаний. Программа должна вывести 10 примеров (например, вычисления значений таблицы умножения), задающихся случайным образом и выставить оценку за 10 правильных ответов – «отлично», за 9 и 8 – «хорошо», за 7 и 6 «удовлетворительно», за 6 и менее – «плохо».




```
import random
# Функция для генерации случайного примера
def generate_example():
    a = random.randint(1, 10)
    b = random.randint(1, 10)
    op = random.choice(['+', '-', '*'])
    if op == '+':
        return f"{a} + {b}", a + b
    elif op == '-':
        return f"{a} - {b}", a - b
    else:
        return f"{a} * {b}", a * b

# Список для хранения правильных ответов
correct_answers = 0

print("Решите 10 примеров. Введите ответ для каждого примера.")

# Цикл для вывода 10 примеров
for i in range(1, 11):
```

```
        example, result = generate_example()
        print(f"{i}. {example} = ", end="")
        user_answer = int(input())
        if user_answer == result:
            correct_answers += 1

    # Определение оценки
    if correct_answers == 10:
        grade = "Отлично"
    elif correct_answers >= 8:
        grade = "Хорошо"
    elif correct_answers >= 6:
        grade = "Удовлетворительно"
    else:
        grade = "Плохо"

    print(f"Количество правильных ответов: {correct_answers} из 10")
    print(f"Оценка: {grade}")
```

НАПИСАНИЕ СКРИПТА

1. Напишите скрипт, содержащий функцию dict_l(lst), которая принимает аргумент в виде списка и возвращает словарь, в котором каждый элемент списка является и ключом, и значением. Предполагается, что элементы списка будут соответствовать правилам задания ключей в словарях. А скрипт должен проверять, как работает эта функция.

```
def dict_l(lst):
    if not isinstance(lst, list):
                    raise ValueError(f""" Тип данных {type(lst)} не поддерживается!!! Необходимо предать тип "List" """)

    try:
        return {val:val for val in lst}
    except Exception as e:
        print(f"Произошла ошибка: {e}")



test_lst_1 = [1, 2, 3, 4]
test_lst_2 = [1, 1, 1, 1]
test_lst_3 = []


# Тест первый
assert {1: 1, 2: 2, 3: 3, 4: 4} == dict_l(test_lst_1), "Ошибка в создании словаря, ожидаемый словарь не равен словарю на выводе метода"

# Тест второй
assert {1: 1} == dict_l(test_lst_2), "Ошибка в создании словаря, ожидаемый словарь не равен словарю на выводе метода"

# Тест третий
assert {} == dict_l(test_lst_3), "Ошибка в создании словаря, ожидаемый словарь не равен словарю на выводе метода"
```

2. Создайте текстовый файл, содержащий фамилии студентов и их экзаменационные оценки по 5 предметам. Напишите скрипт, печатающий фамилий студентов, сдавших экзамены только на "5".

```
excellent_students = []

with open("student_grades.txt", "r", encoding="utf-8") as file:
    for line in file:
        data = line.strip().split(",")
        name = data[0]
        grades = [int(grade) for grade in data[1:]]
        if all(grade == 5 for grade in grades):
            excellent_students.append(name)

print(f'Ученики сдавшие экзамены только на 5: {excellent_students}')
```

Данные в текстовом файле

```
Иванов,5,4,5,3,5
Петров,4,5,3,5,4
Сидоров,5,5,5,5,5
Кузнецов,3,4,5,4,3
Смирнов,5,5,5,5,5
```

3. Составьте скрипт, который содержит две функции. Первая (пусть будет inp) обеспечивает ввод приведенных выше данных в бинарный файл с помощью библиотеки pickle, а вторая функция обеспечивает чтение этих данных с помощью этой же библиотеки и вывода таблицы в консоль в виде таблицы с помощью библиотеки prettytable.

Дан следующий набор данных:
nm = ['ФИО', ' Возраст', 'Должность', 'Зарплата', 'Стаж']
dt= ['Соловьев Н.С.', '65', 'Директор','150000', '12',
'Брежнев Л.И.', 'Зам. директора', ''43,'120000', '10',
'Коробова С.И.', '34', 'Главный бухгалтер', '100000', '7',
'Старков Н.В.', '35', 'Завхоз', '70000','10',
'Баранова С.А.', '33', 'Зав. складом', '90000', '6']

```
import pickle
from prettytable import PrettyTable

def inp(nm, dt):
    with open("data.bin", "wb") as file:
        pickle.dump((nm, dt), file)

def outp():
    with open("data.bin", "rb") as file:
        nm, dt = pickle.load(file)

    table = PrettyTable(nm)
```

```
        for row in dt:
            table.add_row(row)

        print(table)



    nm = ['ФИО', 'Возраст', 'Должность', 'Зарплата', 'Стаж']
    dt = [
        ['Соловьев Н.С.', '65', 'Директор', '150000', '12'],
        ['Брежнев Л.И.', '43', 'Зам. директора', '120000', '10'],
        ['Коробова С.И.', '34', 'Главный бухгалтер', '100000', '7'],
        ['Старков Н.В.', '35', 'Завхоз', '70000', '10'],
        ['Баранова С.А.', '33', 'Зав. складом', '90000', '6']
    ]

    # Вызов функций
    inp(nm, dt)
    outp()
```

СОЗДАНИЕ КЛАССА

1. Описать базовый класс «Строка». Обязательные поля класса: поле для хранения символов строки; значение типа word для хранения длины строки в байтах. Реализовать обязательные методы следующего назначения: конструктор без параметров; конструктор, принимающий в качестве параметра строковый литерал; конструктор, принимающий в качестве параметра символ; метод получения длины строки; метод очистки строки. Описать производный от «Строка» класс «Комплексное число». Строки данного класса состоят из двух полей, разделенных символом i. Первое поле задает значение действительной части числа, второе – значение мнимой. Каждое из полей может содержать только символы десятичных цифр и символы + и -, задающие знак числа. Символы + и – могут находиться только в первой позиции числа, причем символ + может отсутствовать, в этом случае число считается положительным. Для класса «Комплексное_число» определить следующие методы: проверка на равенство; сложение чисел; умножение чисел.

```
class String:
    def __init__(self, string_literal=None, character=None):
            self.characters = string_literal if string_literal else
character if character else ""
        self.length = len(self.characters.encode('utf-8'))

    def get_length(self):
        return self.length

    def clear_string(self):
        self.characters = ""
        self.length = 0

class ComplexNumber(String):
    def __init__(self, real_part, imaginary_part):
```

```
        self.real_part = real_part
        self.imaginary_part = imaginary_part
        super().__init__(str(real_part) + 'i' + str(imaginary_part))

    def __eq__(self, other):
         return self.real_part == other.real_part and self.imaginary_part
== other.imaginary_part

    def __add__(self, other):
        real_sum = self.real_part + other.real_part
        imaginary_sum = self.imaginary_part + other.imaginary_part
        return ComplexNumber(real_sum, imaginary_sum)

    def __mul__(self, other):
                real_product = self.real_part * other.real_part -
self.imaginary_part * other.imaginary_part
            imaginary_product = self.real_part * other.imaginary_part +
self.imaginary_part * other.real_part
        return ComplexNumber(real_product, imaginary_product)

# Пример использования классов
string1 = String("Hello, World!")
print(string1.get_length())  # Выводит длину строки

complex1 = ComplexNumber(3, 4)
complex2 = ComplexNumber(2, -1)

print(complex1 == complex2)  # Проверка на равенство
complex_sum = complex1 + complex2  # Сложение чисел
complex_product = complex1 * complex2  # Умножение чисел
```

2. Описать класс «самолет», содержащий следующие закрытые поля: название пункта назначения; шестизначный номер рейса; время отправления. Предусмотреть свойства для получения состояния объекта. Описать класс «аэропорт», содержащий закрытый массив самолетов. Обеспечить следующие возможности: вывод информации о самолете по номеру рейса с помощью индекса; вывод информации о самолетах, отправляющихся в течение часа после введенного с клавиатуры времени; вывод информации о самолетах, отправляющихся в заданный пункт назначения; перегруженную операцию сравнения, выполняющую сравнение времени отправления двух самолетов. Информация должна быть отсортирована по времени отправления. Написать программу, демонстрирующую все разработанные элементы классов.

```
using System;
using System.Collections.Generic;
using System.Linq;

namespace Planer
{
    class Program
    {
        static void Main(string[] args)
```

```
        {
            Airport airport = new Airport(new Plane[] {
                new Plane("Колумбия",new
int[]{12,15},"1234567891011121"),
                new Plane("Гонг конг",new
int[]{16,15},"1234567891411121"),
                new Plane("Екатеринбург",new
int[]{12,45},"1234967891011121")
            });

            Console.WriteLine("Вывод 2 элемента\n");
            Console.WriteLine(airport.Print(2));
            Console.Write("Введите время отправления (час:минуты):
");
            int[] time;
            try
            {

time=Array.ConvertAll(Console.ReadLine().Split(':'), s =>
int.Parse(s));
            }
            catch
            {
                time= null;
            }

            foreach(Plane plane in airport.Print(time))
            {
                Console.WriteLine(plane.Sostoiyanie);
            }

Console.WriteLine("---------------------------------");
            Console.Write("Введите пункт назначения: ");
            foreach(Plane plane in
airport.Print(Console.ReadLine()))
            {
                Console.WriteLine(plane.Sostoiyanie);
            }

Console.WriteLine("---------------------------------");
            Console.WriteLine(airport.Sravnenie(1, 2));
```

```
Console.WriteLine("--------------------------------");

Console.WriteLine(airport.Sravnenie("1234567891011121",
"1234567891411121"));
        }
    }

    class Plane
    {
        string name;
        string number;
        int[] time=new int[2];

        public string Name => name;
        public int[] Time => time;
        public string Number => number;

        public string Sostoiyanie =>
            "Пункт назначения: " + name + "\n" +
            "Время отправления: " + time[0]+":"+time[1] + "\n" +
            "Код рейса: " + number + "\n";
        public Plane(string name, int[] time, string number)
        {
            this.name = name;
            this.time = time;
            this.number = number;

        }
    }
    class Airport
    {
        Plane[] planes;

        public Airport(Plane[] planes)
        {
            planes=planes.OrderBy(plane =>
plane.Time[1]).ToArray();
            planes=planes.OrderBy(plane =>
plane.Time[0]).ToArray();
            this.planes = planes;
        }
        public string Print(int index)
```

```
        {
            try
            {
                return planes[index].Sostoiyanie;
            }
            catch
            {
                return "Таково индекса нет";
            }

        }

        public List<Plane> Print(int[] pl_vvod)
        {
            List<Plane> inf=new List<Plane>();
            if (pl_vvod!=null)
            foreach(Plane plane in planes)
            {
                if((plane.Time[0]==pl_vvod[0]+1 &&
plane.Time[1]<=pl_vvod[1])||(plane.Time[0] == pl_vvod[0] &&
plane.Time[1] > pl_vvod[1]))
                {
                    inf.Add(plane);
                }
            }
            return inf;
        }

        public Plane[] Print(string pl_vvod)
        {
            return planes.Where(plane => plane.Name ==
pl_vvod).ToArray();
        }

        public string Sravnenie(int index1, int index2)
        {
            return "Разница между "+index1+" и " + index2 +"
рейсами " + Math.Abs(planes[index1].Time[0] -
planes[index2].Time[0]) + ":" + Math.Abs(planes[index1].Time[1] -
planes[index2].Time[1]);
        }
        public string Sravnenie(string name1, string name2)
```

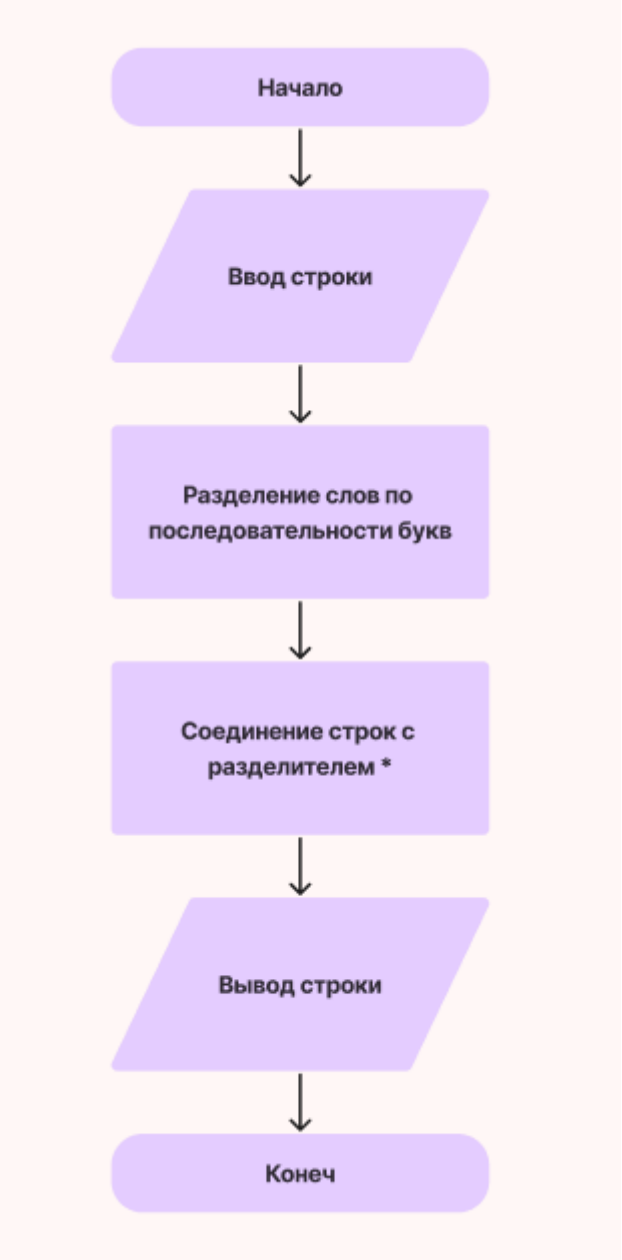
```
        {
            Plane plane1;
            Plane plane2;
            try
            {
                plane1= Array.Find(planes, plane => plane.Number
== name1);
                plane2 = Array.Find(planes, plane => plane.Number
== name2);
            }
            catch
            {
                return "ошибка ввода данных";
            }
            return "Разница между рейсами с кодом " + name1 + " и
" + name2 + " " + Math.Abs(plane1.Time[0] - plane2.Time[0]) + ":"
+ Math.Abs(plane1.Time[1] - plane2.Time[1]);
        }
    }

}
```

3. Описать класс «запись», содержащий следующие закрытые поля: фамилия, имя; номер телефона; дата рождения (массив из трех чисел). Предусмотреть свойства для получения состояния объекта. Описать класс «записная книжка», содержащий закрытый массив записей. Обеспечить следующие возможности: вывод на экран информации о человеке, номер телефона которого введен с клавиатуры; если такого нет, выдать на дисплей соответствующее сообщение; поиск людей, день рождения которых сегодня или в заданный день; поиск людей, день рождения которых будет на следующей неделе; поиск людей, номер телефона которых начинается на три заданных цифры. Написать программу, демонстрирующую все разработанные элементы классов.

БЛОК-СХЕМА

Составьте блок-схему и напишите программу, которая вводит строку символов и заменяет все разделители на символ «*», причем если между словами несколько разделителей, то должен стоять ОДИН символ «*». Например, «мама,,,, мыла…. раму???» меняется на «мама*мыла*раму*».

```
import re

# s = input('Введите строку: ')
s = "мама,,,, мыла…. раму???"

s = "*".join(re.split(r'\W+', s))

print('Преобразованная строка: ', s)
```



РАЗРАБОТКА ПРОГРАММЫ

Программа должна обеспечивать ввод данных из формы и записывать их в базу данных. По завершению программы необходимо вывести данные из базы в окно интерпретатора.

СОЗДАНИЕ БД

1. Создайте и заполните базу данных сотрудников предприятия связи, содержащей следующие таблицы:

-Сотрудники (табельный номер, фио, пол, номер_должности, отдел, дата рождения, хобби, стаж, семейное положение (Б-брак, Х- холост, Р- разведен), дети).

- Штатное расписание (номер_должности, должность, оклад).

Установите связь между таблицами.

Создайте следующие запросы

- «Подарки» - сколько подарков на 8 Марта нужно приготовить в каждом отделе.

- «Старшие» - информация о сотрудниках, имеющих возраст больше среднего возраста всех сотрудников, вывести поля: фио, должность, оклад, пол, возраст (вычислить), отсортировав по полю дата рождения.

```
            НЕДОРЕАЛИЗОВАНО!!!! (см выше, отмечено красным)
-- create
CREATE TABLE STAFF (
  extId INTEGER PRIMARY KEY,
  name TEXT NOT NULL,
  sex TEXT NOT NULL,
  jobId INTEGER NOT NULL,
  birthday DATETIME NOT NULL,
  hobby TEXT,
  experience TEXT NOT NULL,
  maritalStatus TEXT NOT NULL,
  children BOOLEAN NOT NULL DEFAULT False
);

CREATE TABLE STAFF_SCHEDULE (
  jobId INTEGER NOT NULL,
  jobName TEXT NOT NULL,
  sex TEXT NOT NULL,
  salary INTEGER NOT NULL
);


-- insert
INSERT INTO STAFF VALUES
(1, 'Иванов Иван Иванович', 'Мужской', 101, '1980-01-01 00:00:00',
'Чтение книг', '5 лет опыта работы', 'Б', False),
(2, 'Петрова Мария Петровна', 'Женский', 102, '1990-02-02 00:00:00',
'Спорт', '3 года опыта работы', 'Х', False),
(3, 'Сидоров Сергей Сидорович', 'Мужской', 103, '2000-03-03 00:00:00',
'Кулинария', '1 год опыта работы', 'Х', False),
(4, 'Алексеева Елена Алексеевна', 'Женский', 104, '2010-04-04
00:00:00', 'Фотография', 'Нет опыта работы', 'Р', True),
(5, 'Сергей Кузнецов', 'Мужской', 101, '1975-03-03 00:00:00',
'Фотография', '7 лет опыта', 'Х', False),
(6, 'Ольга Смирнова', 'Женский', 104, '2000-04-04 00:00:00',
'Кулинария', '1 год опыта', 'Б', True),
(7, 'Алексей Фёдоров', 'Мужской', 105, '1965-05-05 00:00:00', 'Гонка на
велосипеде', '10 лет опыта', 'Б', True);

INSERT INTO STAFF_SCHEDULE (jobId, jobName, sex, salary) VALUES
(101, 'Разработчик', 'Мужской', 80000),
(102, 'Дизайнер', 'Женский', 75000),
(103, 'Менеджер', 'Мужской', 90000),
(104, 'Аналитик', 'Женский', 85000),
(105, 'Тестировщик', 'Мужской', 70000);

-- fetch
SELECT COUNT(*) as 'Кол-во подарков', STAFF_SCHEDULE.jobName as
'Название отдела' FROM STAFF
INNER JOIN STAFF_SCHEDULE ON STAFF.jobId = STAFF_SCHEDULE.jobId
WHERE STAFF.sex = 'Женский' GROUP BY STAFF_SCHEDULE.jobName;
```

2. Разработайте базу данных предприятия связи, содержащей следующие таблицы:

-Сотрудники (табельный номер, фио, пол, код_должности, отдел, дата рождения, хобби, стаж, семейное положение (Б-брак, Х- холост, Р- разведен), дети).

Штатное расписание (код_должности, должность, оклад).

Установите связь между таблицами.

Сформируйте запросы для выборки информации:

- о заработной плате сотрудников (зарплата равна оклад плюс премия (премия рассчитывается так: если стаж <5, то 50 % от оклада; иначе 100 % от оклада); вывести поля: фио, должность, зарплата, стаж, отсортировав по отделам;

- о сотрудниках, имеющих стаж больше среднего стажа всех сотрудников; вывести поля: отдел, фио, должность, стаж, отсортировав по стажу.

```
-- create
CREATE TABLE STAFF (
  extId INTEGER PRIMARY KEY,
  name TEXT NOT NULL,
  sex TEXT NOT NULL,
  jobId INTEGER NOT NULL,
  department TEXT NOT NULL,
  birthday DATETIME NOT NULL,
  hobby TEXT,
  experience INTEGER NOT NULL,
  maritalStatus TEXT NOT NULL,
  children BOOLEAN NOT NULL DEFAULT False
);

CREATE TABLE STAFF_SCHEDULE (
  jobId INTEGER NOT NULL,
  jobName TEXT NOT NULL,
  sex TEXT NOT NULL,
  salary INTEGER NOT NULL
);


-- insert
INSERT INTO STAFF VALUES
(1, 'Иванов Иван Иванович', 'Мужской', 101, 'IT', '1980-01-01
00:00:00', 'Чтение книг', 5, 'Б', False),
(2, 'Петрова Мария Петровна', 'Женский', 102, 'HR', '1990-02-02
00:00:00', 'Спорт', 3, 'Х', False),
(3, 'Сидоров Сергей Сидорович', 'Мужской', 103, 'Sales', '2000-03-03
00:00:00', 'Кулинария', 1, 'Х', False),
(4, 'Алексеева Елена Алексеевна', 'Женский', 104, 'Sales', '2010-04-04
00:00:00', 'Фотография', 0, 'Р', True),
(5, 'Сергей Кузнецов', 'Мужской', 101, 'Marketing', '1975-03-03
00:00:00', 'Фотография', 7, 'Х', False),
(6, 'Ольга Смирнова', 'Женский', 104, 'HR', '2000-04-04 00:00:00',
'Кулинария', 1, 'Б', True),
(7, 'Алексей Фёдоров', 'Мужской', 105, 'HR', '1965-05-05 00:00:00',
'Гонка на велосипеде', 10, 'Б', True);

INSERT INTO STAFF_SCHEDULE (jobId, jobName, sex, salary) VALUES
(101, 'Разработчик', 'Мужской', 80000),
(102, 'Дизайнер', 'Женский', 75000),
(103, 'Менеджер', 'Мужской', 90000),
(104, 'Аналитик', 'Женский', 85000),
(105, 'Тестировщик', 'Мужской', 70000);
```

```
-- fetch
SELECT
  STAFF_SCHEDULE.salary + IF(STAFF.experience < 5,
STAFF_SCHEDULE.salary * 0.5, STAFF_SCHEDULE.salary) AS "Зарплата",
  name,
  jobName,
  salary,
  experience
FROM STAFF
INNER JOIN STAFF_SCHEDULE ON STAFF.jobId = STAFF_SCHEDULE.jobId
ORDER BY department;

SELECT department, name, jobName, experience
FROM STAFF
INNER JOIN STAFF_SCHEDULE
  ON STAFF.jobId = STAFF_SCHEDULE.jobId
WHERE experience > (SELECT AVG(experience) FROM STAFF)
ORDER BY experience DESC;
```

3. Спроектируйте базу данных для предприятия связи, содержащую следующие таблицы:

-Сотрудники (отдел, табельный номер, фио, пол, код должности, дата рождения, хобби, стаж).

- Штатное расписание (код должности, должность, оклад).

Установите связь между таблицами.

Создайте формы(у) для ввода и редактирования данных в таблицах.

Сформируйте следующие запросы:

- «Зарплата»: информация о зарплате каждого сотрудника, вычисленная по формуле зарплата = оклад + премия, где премия зависит от стажа работы и если стаж меньше или равен 5 лет, то премия равна 50 % от оклада; если стаж больше 5 лет, премия равна 100 % от оклада;

-«сотрудницы»: (пол- женский), имеющих зарплату выше средней по отделу, вывести поля: отдел, фио, пол, должность, зарплата.

4. Создайте и заполните базу данных сотрудников предприятия связи, содержащую следующие таблицы:

-Сотрудники (табельный номер, ФИО, пол, номер должности, отдел, дата рождения, стаж, семейное положение (Б-брак, Х- холост, Р- разведен), дети).

- Штатное расписание (номер должности, должность, оклад).

Составьте и выполните запросы на выборку отдела, фио, должности, зарплаты о тех сотрудниках, у которых зарплата больше средней, отсортировав по отделам, а внутри отдела по фамилии.

КОНЕЦ ДОКУМЕНТА