**1. Классы:**

* **City:** Хранит информацию о городе (название, регион, население).
  + Конструктор упрощает создание объектов City.
* **CityNode:** Узел двусвязного списка, хранит указатели на соседние узлы (prev, next) и данные (City\*).
  + Конструктор для удобства создания узла с данными.
* **CityList:** Управляет списком городов.
  + head и tail: Указатели на начало и конец списка.
  + Конструктор, деструктор: Инициализация и очистка списка.
  + Методы:
    - addCity: Добавление города в конец списка.
    - printList: Вывод списка на экран.
    - printRegionsByPopulation: Вывод регионов по убыванию населения.
      * Использует std::map для подсчёта населения по регионам.
      * Сортировка регионов с помощью std::sort и лямбда-функции.
    - deleteCitiesByRegion: Удаление городов по региону.
      * Обработка случаев удаления head и tail.
    - clear: Очистка списка и освобождение памяти.
    - loadFromCSV: Загрузка данных из CSV файла.
      * Чтение построчно (std::getline).
      * Разделение строки на части с помощью std::string::find и std::string::substr.
      * Конвертация строки в число (std::stoi).

**2. Работа со строками:**

* std::getline: Чтение строки с пробелами.
* std::string::find, std::string::substr: Разбиение строки на подстроки.
* std::stoi: Преобразование строки в число.

**3. Контейнеры STL:**

* std::map: Хранение пар "регион - население" для удобного подсчета.
* std::vector: Используется для сортировки пар "регион - население"

**4. Алгоритмы:**

* std::sort: Сортировка регионов по убыванию населения.
* Лямбда-функция: Используется для определения критерия сортировки в std::sort.

**5. Обработка ошибок:**

* Проверка открытия файла (file.is\_open()).

**6. Основная программа (main)**:

* Организация меню с выбором действий.
* Вызов соответствующих методов класса CityList.
* Чтение данных с клавиатуры.
* **Вопрос:** Зачем вы использовали классы в своей программе? Почему не реализовали все функции в main?
  + **Ответ:** Использование классов делает код более структурированным, понятным и лёгким в поддержке. Данные о городе и методах работы с ними логически объединены в класс CityList. Это улучшает читаемость и позволяет легче модифицировать программу в дальнейшем.
* **Вопрос:** Что делает деструктор в классе CityList?
  + **Ответ:** Деструктор ~CityList() вызывается автоматически при уничтожении объекта CityList (например, в конце работы программы). Он отвечает за корректное освобождение памяти, занимаемой списком, чтобы избежать утечек.
* **Вопрос:** Вы используете std::map в функции printRegionsByPopulation. Расскажите, для чего нужна эта структура данных и как она работает?
  + **Ответ:** std::map — это ассоциативный контейнер, который хранит пары "ключ - значение". В моём случае ключом является название региона (std::string), а значением — суммарное население городов этого региона (int). Использование std::map в моём случае позволяет удобно просуммировать по названиями регионов население.
* **Вопрос:** Как работает сортировка регионов по населению?
  + **Ответ:** Для сортировки я использую std::sort. Алгоритм std::sort работает очень быстро и эффективно. Для указания критерия сортировки я передаю в std::sort лямбда-функцию. Эта функция сравнивает два элемента std::map (пары "регион - население") по полю second (население) и возвращает true, если население первого региона больше, что обеспечивает сортировку по убыванию.
* **Вопрос:** Опишите, как происходит загрузка данных из CSV файла.
  + **Ответ:** Файл открывается с помощью std::ifstream. Далее, построчно считывается каждая строка файла с помощью std::getline. Каждая строка содержит информацию о городе, разделённую точкой с запятой. Используя функции std::string::find и std::string::substr, я разбиваю строку на части и извлекаю название города, региона и население. Затем население конвертируется из строки в число с помощью std::stoi и создаётся новый город в списке.
* **Вопрос:** При удалении элемента из списка вы обрабатываете несколько случаев. Зачем это нужно?
  + **Ответ:** Удаление элемента из середины списка отличается от удаления первого или последнего элемента. Если удаляется первый элемент (head), то нужно обновить указатель head на следующий элемент. Аналогично, при удалении последнего элемента (tail) нужно обновить tail. Поэтому в коде есть проверки на current->prev и current->next, чтобы корректно обработать все случаи.
* **Вопрос:** Что происходит с памятью, выделенной под узлы списка, при удалении города или очистке всего списка?
  + **Ответ:** Для предотвращения утечек памяти я использую оператор delete. В деструкторе ~CityList() происходит удаление всех узлов списка. Также при удалении города из списка с помощью функции deleteCitiesByRegion память, занимаемая удалённым узлом и данными о городе, освобождается.