1. \*\*Последовательность обработки исходного текста транслятором\*\*:

- Лексический анализ (лексер)

- Синтаксический анализ (парсер)

- Семантический анализ

- Генерация промежуточного кода

- Оптимизация кода

- Генерация объектного кода

- Линковка и загрузка

2. \*\*Лексический анализ\*\*:

- \*\*Определение\*\*: Это процесс разбора исходного текста программы на токены, которые представляют минимальные значимые единицы программы (ключевые слова, идентификаторы, операторы, константы и т.д.).

- \*\*Назначение\*\*: Преобразовать исходный текст программы в последовательность токенов для передачи синтаксическому анализатору.

- \*\*Применение\*\*: На этапе компиляции программ для подготовки текста к синтаксическому анализу.

3. \*\*Лексический анализатор\*\*:

- \*\*Место в структуре транслятора\*\*: Лексический анализатор — первый модуль компилятора. Он передает токены синтаксическому анализатору.

- \*\*Входная информация\*\*: Исходный код программы.

- \*\*Выходная информация\*\*: Поток токенов.

4. \*\*Назначение и принципы работы лексических анализаторов\*\*:

- \*\*Назначение\*\*: Преобразование исходного текста в поток токенов и устранение ошибок, связанных с неправильными символами или последовательностями.

- \*\*Принципы работы\*\*: Лексер читает исходный текст посимвольно, объединяет символы в лексемы и классифицирует их как токены.

5. \*\*Функции лексического анализатора\*\*:

- Разбиение исходного текста на токены.

- Удаление пробелов и комментариев.

- Распознавание ключевых слов и операторов.

- Определение констант и идентификаторов.

- Передача токенов синтаксическому анализатору.

6. \*\*Результат успешного лексического разбора исходного текста\*\*:

- Поток токенов, каждый из которых представляет отдельную лексическую единицу программы, передаваемый на этап синтаксического анализа.

7. \*\*Способы взаимодействия лексического анализатора с синтаксическим анализатором\*\*:

- Лексер передает токены синтаксическому анализатору по мере их распознавания.

- Взаимодействие может происходить через API или буфер токенов, в который лексер помещает токены для использования парсером.

8. \*\*Таблица идентификаторов и область видимости конкретного идентификатора\*\*:

- Таблица идентификаторов хранит информацию об идентификаторах программы (имена, типы данных, адреса).

- Область видимости определяет, где в программе идентификатор доступен (локальная, глобальная, блоковая область).

9. \*\*Ошибки, определяемые лексическим анализатором\*\*:

- Неправильные символы или последовательности символов.

- Незакрытые строки.

- Превышение допустимой длины идентификаторов.

10. \*\*Грамматики, применяемые для описания лексики языка программирования\*\*:

- Регулярные грамматики и регулярные выражения используются для описания синтаксиса токенов в лексическом анализе.

11. \*\*Простейший распознаватель регулярного множества\*\*:

- Это конечный автомат, который принимает или отклоняет строки, описанные регулярным выражением.

12. \*\*Алгоритм разбора цепочки символов\*\*:

- Конечный автомат считывает символы строки и на каждом шаге переходит в новое состояние в зависимости от входного символа. Цепочка принимается, если автомат приходит в конечное состояние.

13. \*\*Определение конечного автомата\*\*:

- Конечный автомат — это абстрактная машина, которая может находиться в одном из конечного множества состояний и переходит из одного состояния в другое под действием входного символа.

14. \*\*Определение графа переходов конечного автомата\*\*:

- Граф переходов — это ориентированный граф, где вершины представляют состояния, а ребра обозначают переходы между состояниями при чтении символов.

15. \*\*Пример построения графа конечного автомата по регулярному выражению (d|a)+(ab)\*c\*\*:

- Начнем с объединения состояний для символов `d` и `a`, так как оператор `|` указывает на альтернативу.

- Добавим цикл для `ab`, так как оператор `\*` означает повторение.

- Завершим состояние символом `c`, который должен быть один раз в конце.

- Граф будет содержать ветвление для `d` или `a`, цикл для `ab`, и одно конечное состояние для `c`.