# Слайд 1 – Название работы

**Здравствуйте, уважаемые слушатели! Меня зовут** Переведенцев Артем. **Название моей работы:** “Разработка собственного языка программирования”.

# Слайд 2 – Цель и задачи работы

Целью моей работы была **разработка** **системы** **трансляции** **и исполнения** **программ** **на** **C**-**подобном** **языке**.

В нее входило несколько задач:

**Компилятор, Ассмеблер, Процессор**.

Каждая из этих задач имела свои подзадачи.

В **компилятор** входила разработка **лексического** **анализатора**, **синтаксического** **анализатора**, **семантического** **анализатора** и **т.д**.

В **ассемблер** входила разработка системы трансляции команд в машинный код.

В **процессор** – исполнение машинного кода.

Обо всем этом подробнее в следующих слайдах.

# Слайд 3 – Стадии трансляции

**Давайте сначала разберемся, как же заветные строчки кода превращаются в команды для процессора**.

Как и в любом языке программирования пользователь сначала пишет программу. Затем модуль **“Compiler”** начинает работать с кодом.

Первый этап это **лексический анализатор.** На выходе он выдает поток лексем.

Далее идет **синтаксический анализатор.** Его конечным результатом является дерево с программой.

И последний этап это **кодогенератор**. На входе он получает дерево с программой, на выходе выдает файл с командами для ассемблера.

Затем мы переходим к следующему модулю - **“Assembler”.**

Этот модуль работает только с ассемблерным кодом и его результатом является файл с командами для процессора (байткодом).

# Слайд 4 – Лексический анализатор

**Начнем с самого первого этапа**. Это лексический анализатор.

Его задача это разбиение текста программы на лексемы или, иначе говоря, парсинг.

Лексема может быть нескольких типов: **число**, **ключевое** **слово**, **команда** **для** **препроцессора**, **оператор**, или **имя**.

# Слайд 5 – Синтаксический анализатор

Далее идет синтаксический анализатор. **Это самая важная часть компилятора**. На данной стадии происходит построение дерева (AST) по потоку лексем, с помощью особых

грамматических правил методом рекурсивного спуска. **Некоторая часть правил представлена на слайде**.

# Слайд 6 – Кодогенерация

После построения дерева мы переходим к кодогенерации. Операции в ассемблере делятся на несколько типов: **арифметические**, **работа** **с** **памятью**, **переходы** и **другие**.

Результатом работы этого анализатора является обход дерева и запись команд в файл с ассемблерным кодом.

# Слайд 7 – Исполнение программы

**После того, как программа собрана, мы должны ее на чем-то запустить**. Для этого была разработана виртуальная машина, исполняющая команды.

Как и в настоящем процессоре в ней имеются: **оперативная память (RAM), набор регистров, два стека и флаги**. Затем происходит исполнение команд.

# Слайд 8 – Результаты

Перейдем к результатам. На слайде вы видите пример простой программы. В данном случае был выбран алгоритм Евклида.

Так же вы можете видеть рядом результат построения дерева из программы. **Для большей наглядности вершины были раскрашены в разные цвета.**

**В ходе работы было написано** приблизительно 5000 строк кода в 24 файлах. На выполнение работы было затрачено около 350 часов.

# Слайд 9 – Итоги

Перейдем к итогам.

В процессе выполнения работы были реализованы следующие задачи разработка: **лексического** **анализатора**, **синтаксического** **анализатора**, **семантического**

**анализатора** и **т.д**. **А** **Также были получены неоценимые знания по работе с деревьями**.

# Слайд 10 – Вопросы

Всем спасибо за внимание! Теперь я готов ответить на ваши вопросы.