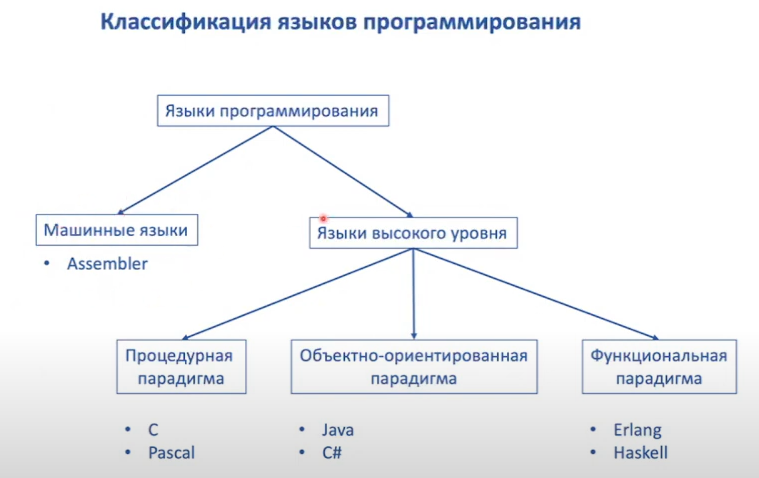
Лекция 1



Assembler

+ Напрямую обращаемся к процессору на его языке

+ Самый близкий к машинному

- Большой код и не структурировано

- Сложен для изучения

- индивидуален для каждой архитектуры

Процедурная парадигма – парадигма в которой программа выглядит в виде линейного кода. Алгоритм описывается в виде последовательности шагов, преобразования входной информации в выходную. Программа состоит из двух частей, часть в виде данных где описываются все переменные и часть где машинные инструкции языка, которые преобразовывают эти данные.

Си

+ Самый быстрый язык

+ Является лучшей версией Assembler

- Чистый язык мало что умеет без библиотек

Объектно-ориентированная парадигма — 3 основных кита: наследование, инкопсуляция и полиморфизм. Данные с кодом находятся в одном классе. Программа состоит из различных классов, а в классе функции с помощью которых мы и взаимодействуем.

Функциональная парадигма - весь код реализуется в виде функций, в отличие от других парадигм если в функцию передается параметр то он не изменен и функция всегда должна что то вернуть, так же нет циклов они реализуются в виде рекурсии то есть вызывает сама себя.

- Сложно перестроится с других парадигм и начать писать на функциональной

+ Простота чтения кода так как название функции обычно объясняет что делает данная функция

Трансляторы — программа которая переводит из любого языка программирования в машинный код.

Компилятор — это транчлятор который сразу проводит программу сразу через три этапа лексический, синтаксический и семантический анализ, на выходе получается бинарная программа которую можно в последствии запустить и посмотреть результат.

+ Если есть какие то ошибки типа лексические. Синтаксические или семантические то компилятор вам укажет на них и не бкдет создавать бинарную программу.

- Если же ошибка была логической то придется найти ошибку, переписать исходный код и снова прогнать всю программу через 5 этапов компиляции

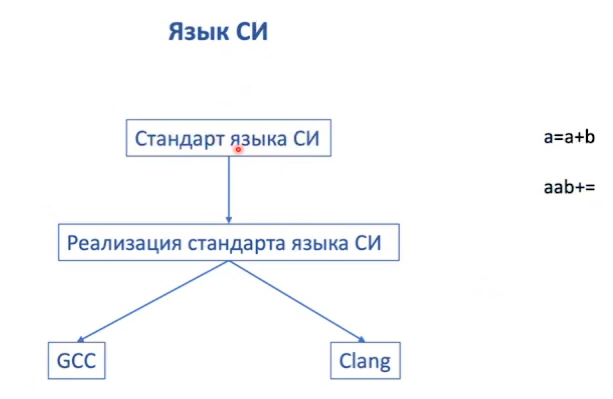
- Бинарный код можно запускать только на той системе которая бинарно совместима то есть та же архитектура, операционная система и тот же машинный язык

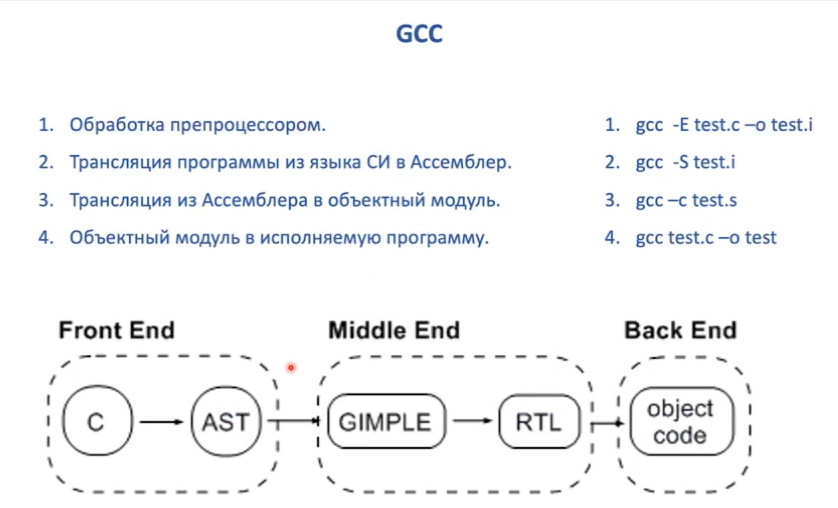
Интерпретатор — Выполняет код по строчно то есть сразу выдает результат по строке.

+ Выдает ошибку сразу как её нашел

+ Архитектурно независимый

- Медленно работает в отличии от компилятора

GCC – ((GNU C Compile) GNU Compiler Collection).

Clang – более современный и быстрый предназначен для x86 архитектуры.

Этапы компиляции

1. Обработка препроцессора — это вспомогательные программы для упрощения программирования так что сначала приводим к чистому си.

+ Посмотреть какие именно версии библиотек использовались

2. Трансляция из языка программирования в Assembler (мнимоники).

+ можно посмотреть код на Assembler и посмотреть что компилятор мог удалил из программы

3. Из Assembler в машинный код. Выполнение не возможно так как данную программу нужно разместить в оперативной памяти по определенному адресу и записать данный адрес в определенный регистр процессора.

+ Для раздельной компиляции

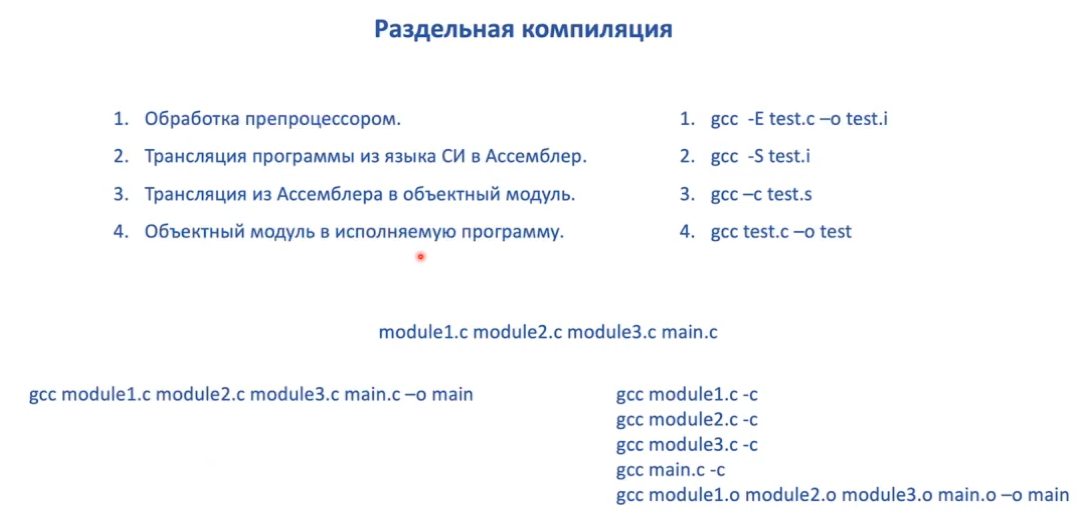
4. Размещается в нужном месте и дописываются некоторые инструкции для того что бы программа могла выполнятся в операционной системе.

Front End – часть в которой программу переводим во внутреннее представление.

Внутреннее представление — (AST) Абстрактное синтаксическое дерево предназначено для перевода кода в более удобный вариант для машины.

Middle End – преобразования оптимизационные (GINPLE) и переводится практически в машинный язык (RTL). То есть приведение над кодом всех возможных оптимизаций.

Back End – трансляция в машинный язык

Раздельная компиляция — позволяет разбить программу на нескоько отдельных модулей и их уже объединить в бинарник.

Лучший вариант для такой компиляции это каждый модуль довести до объектного файла что бы прошли первые 3 этапа компиляции и уже объектные файлы объединить в бинарник.

+ в большой программе дает возможность пере компилировать не всю программу, а только тот модуль который был изменен и в последствии объединим в бинарник

<https://gcc.gnu.org/onlinedocs/gccint/> - Про GCC