**Программа** – это данные, предназначенные для управления конкретными компонентами системы обработки информации (СОИ) в целях реализации определенного алгоритма

В машине фон Неймана и код программы, и обрабатываемые данные хранятся в одном и том же пространстве памяти. Это отличается от ранних компьютеров, где программа задавалась переключателями или перфокартами и была отделена от данных.

* **Для программиста** программа кажется активной — она выполняет действия, управляет вычислениями.
* **Для процессора** же команды программы — это просто набор данных, которые он считывает, интерпретирует и выполняет шаг за шагом.

**Программное обеспечение** = программа + документация. В отличие от разовых программ, написанных, например, для одного эксперимента, ПО разрабатывается с расчётом на долговременное и многократное применение разными пользователями.

Ряд необходимых свойств ПО:

* Необходимость документирования
* Эффективность
* Надежность
* Возможность сопровождения

Появление ошибок любого уровня не должно приводить к краху системы. Ошибки должны вылавливаться диагностироваться и (если их невозможно исправить) превращаться в корректные отказы

* **Системные** **структуры** данных должны сохраняться **безусловно**
* Сохранение целостности **пользовательских** данных **желательно**

**Системная программа** – программа, предназначенная для поддержания работы СОИ или повышения эффективности её использования

Например: операционные системы, файловые системы, драйверы, утилиты, системы программирования

**Прикладная программа** – программа, предназначенная для решения задачи в определенной области применения СОИ

Например: Текстовые редакторы, табличные редакторы, графические редакторы, браузеры, игры

**Системное программирование** – это процесс разработки системных программ (в том числе, управляющих и обслуживающих). Также это разработка программ сложной структуры.

**Промежуточное (связующее) ПО** – совокупность программ, осуществляющих управление вторичными (**виртуальные** ресурсы, которые создаются **самими программами, а не физические**) ресурсами, ориентированными на решение определенного класса задач

Например: СУБД, модули управления языком интерфейса ИС, программы сбора и предварительной обработки информации

**Промежуточное (связующее) ПО** – комплекс технологического ПО для обеспечения взаимодействия между различными приложениями, системами, компонентами

Например: Веб-сервер, сервер приложений, сервисная шина, система управления контентом

С точки зрения инструментальных средств разработки промежуточное ПО ближе к прикладному, так как не работает на прямую с первичными ресурсами, а использует для этого сервисы, предоставляемые системным ПО

С точки зрения алгоритмов и технологий разработки промежуточное ПО ближе к системному, так как всегда является сложным программным изделием многократного и многоцелевого использования и в нем применяются те же или сходные алгоритмы, что и в системном ПО

Современные тенденции развития ПО состоит в снижении объема как системного, так и прикладного программирования. Основная часть работы программистов выполняется в промежуточном ПО

К функциям системного ПО принято относить:

* Системное ПО создаёт среду, в которой могут работать другие программы.
* Автоматизация разработки новых программ
* Обеспечение надежной и эффективной работы компьютера и компьютерной сети
* Проведение диагностики и профилактики аппаратуры компьютера и компьютерных сетей
* Выполнение вспомогательных технологических процессов (копирование, архивирование, восстановление после сбоев и т.д.)

Существуют следующие группы системного ПО:

* Операционные системы
* Интерфейсные оболочки (ОС) – позволяют пользователю взаимодействовать с операционной системой, текстовые, графические
* Системы управления файлами
* Системы программирования
* Утилиты
* Драйверы – программы, которые позволяют операционной системе взаимодействовать с аппаратными устройствами
* Средства сетевого доступа

**Управляющие программы** — это системные программы, которые выполняют важные функции для правильного функционирования всей системы. Они управляют ресурсами компьютера и взаимодействуют с внешней средой (например, другими устройствами, пользователями или сетью). + Восстановление после неисправностей

**Обслуживающие программы (или утилиты)** — это программы, предназначенные для оказания услуг общего характера пользователям и обслуживающему персоналу СОИ

**Базовое системное ПО** — это **минимальный набор программ**, который необходим для того, чтобы **компьютер мог функционировать**. Оно включает в себя все самые важные компоненты, без которых компьютер не сможет выполнять основные задачи.

**Сервисное системное ПО** — это **программы и комплексы программ**, которые дополняют базовое ПО и создают **удобную среду для работы** других программ и пользователей. Это ПО предоставляет дополнительные функции и удобства, которые делают работу с компьютером и сетью более эффективной.

**Система программирования** – система, образуемая языком программирования, компилятором или интерпретатором программ, представленных на этом языке, соответствующей документацией, а также вспомогательными средствами для подготовки программ к форме, пригодной для выполнения

Системы программирования включают в себя следующие средства:

* Редактор текста
* Транслятор
* Компоновщик
* Отладчик
* Библиотеки подпрограмм

**Программный модуль** –программа или фрагмент программы, предназначенный для хранения, трансляции, объединения с другими программными модулями и загрузки в оперативную память

Требования к программным модулям:

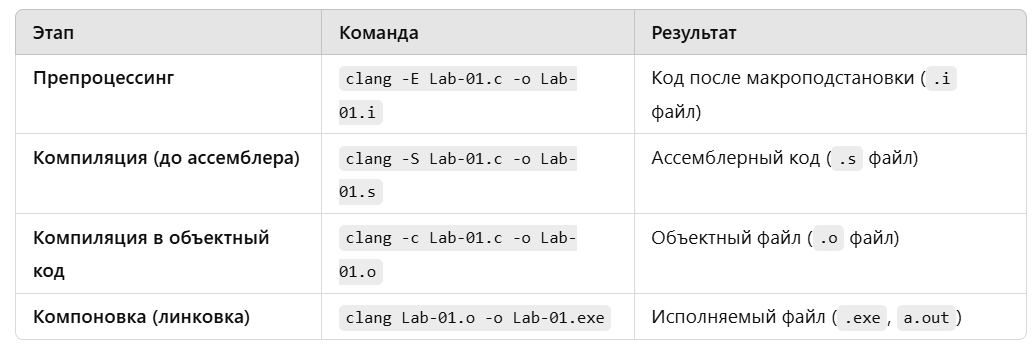
* **Функциональность** – модуль должен выполнять законченную функцию
* **Несвязность** – модуль должен иметь минимум связей с другими модулями, связь через глобальные переменные и области памяти нежелательна
* **Специфицируемость** – входные и выходные параметры модуля должны четко формулироваться

**Исходный модуль** – программный модуль на исходном языке, обрабатываемый транслятором и представляемый для него как целое, достаточное для проведения трансляции

**Транслятор** – системная программа, преобразующая исходную программу на одном языке программирования в программу на другом языке

**Виды трансляторов:**

* Ассемблер — это программа, которая переводит код из языка ассемблера в машинный код (или в промежуточный код).
* Компилятор — это программа, которая переводит весь исходный код программы, написанный на высокоуровневом языке программирования (например, C, C++), в машинный код или в промежуточный код (например, в байт-код для Java). Компилятор делает это за один раз (то есть компиляция происходит перед запуском программы).
* Интерпретатор — это программа, которая построчно выполняет исходный код программы, написанный на высокоуровневом языке программирования. В отличие от компилятора, интерпретатор не создает отдельный исполнимый файл, а выполняет код непосредственно, строка за строкой, в момент работы программы.
* Эмулятор — это программа, которая имитирует работу одного компьютера на другом. Эмулятор позволяет запускать программы, написанные для другой архитектуры (например, программы для старых компьютеров или разных операционных систем), на современной системе.
* Перекодировщик — это программа, которая переводит код с одного языка программирования на другой, сохраняя его функциональность. В отличие от компилятора или интерпретатора, перекодировщик не обязательно переводит код в машинный язык. Он может преобразовывать код с одного высокоуровневого языка в другой.
* Макропроцессор — это программа, которая занимается обработкой макросов в исходном коде. Макрос — это некая предварительно заданная инструкция или шаблон кода, который автоматически заменяется на более длинный фрагмент кода до того, как программа будет скомпилирована или интерпретирована.

****

**Шаг первый – Предварительная обработка кода**:

* Присоединение исходных файлов
* Работа макропроцессоров



**Шаг второй – Анализ**:

* Лексический анализ



### -fsyntax-only

Этот флаг заставляет Clang выполнять только проверку синтаксиса исходного кода

**XClang**: Когда нужно использовать специфичные для Clang опции или плагины, которые требуют передачи параметров непосредственно Clang. XClang сообщает компилятору Clang, что следующее — это параметр, который будет обработан внутри Clang, а не внешним компилятором.

Флаг **-dump-tokens** используется в **Clang** для вывода токенов исходного кода во время процесса препроцессинга или компиляции. Токены — это минимальные единицы исходного кода, которые компилятор разбивает и понимает. Токенами могут быть ключевые слова, операторы, идентификаторы, литералы, скобки и другие элементы языка программирования.

* Синтаксический анализ
* Семантический анализ



**AST** (Abstract Syntax Tree, абстрактного синтаксического дерева) программы. Абстрактное синтаксическое дерево — это структура данных, которая представляет синтаксис исходного кода в виде дерева, где каждый узел соответствует конструкции языка (например, выражению, оператору или типу).

**Шаг третий – Синтез**:

* Генерация машинно-независимого кода



**LLVM (Low-Level Virtual Machine) IR** — это промежуточный формат, который используется в компиляторах LLVM. Он может быть представлен как текст или бинарный формат. (Intermediate Representation — промежуточное представление).

**LLVM** (Low-Level Virtual Machine) — это набор технологий, предназначенных для разработки компиляторов и других инструментов, которые работают с промежуточным представлением (IR) кода.

* Генерация ассемблерного кода

clang -S Lab-01C.c -o Lab-01C.s

Машинный код (объектный файл):

clang -c Lab-01C.c -o Lab-01C.o

Компилятор обрабатывает один файл кода за раз, но если он ссылается на функции или переменные из других файлов, он не может их сразу связать.

Такие ссылки называются внешними ссылками. Компилятор оставляет их незавершенными, пока не узнает, где именно находятся эти функции/переменные.

В результате компиляции получается объектный файл (.o, .obj) — это машинный код, но он не может быть запущен, так как не все ссылки разрешены.

Компоновщик (линкер) берет все объектные файлы и соединяет их вместе, заменяя внешние ссылки на реальные адреса функций и переменных.

После работы линкера получается исполняемый файл (.exe, a.out), который можно загрузить в память и запустить.

**Загрузочный модуль** – программный модуль, представленный в форме, пригодной для загрузки в оперативную память для выполнения

**Архитектурная модель ПО** — это основная структура программы, включающая её элементы, их взаимодействие друг с другом и с внешней средой, а также принципы, которые направляют создание и развитие программы.

**Виды организации межпрограммного взаимодействия**:

* На уровне библиотек
* На уровне сервисов, осуществляющих обмен данными по различным протоколам
* На уровне данных
* Через различные технологии интеграции (ESB, CORBA, COM, DCOM, ActiveX и др.)

clang не является компилятором в прямом смысле этого слова, а является «фронтендом» для языка программирования С (для языка C++ таковой будет называться clang++)

Под «фронтендом» в инфраструктуре LLVM понимается транслятор из некоторого языка программирования в промежуточный язык (LLVM IR), т.е. непосредственная генерация объектного кода перекладывается на LLVM

* Популярен
* Поддерживается «большой тройкой» операционных систем и не только
* Входит в крупную инфраструктуру LLVM
* Язык LLVM IR является модульным (отделяет процесс компиляции на несколько этапов), что позволяет более гибко разрабатывать программное обеспечение
* Поддержка «фронтендов» для множества популярных языков программирования
* Хорошая система вывода информации об ошибках в понятном для разработчика виде (за счёт особой внутренней реализации)
* Производительность (не является лучшим во всём, но предоставляет очень хорошую скорость работы конечного приложения)
* Хорошая автоматическая оптимизация кода и не только…

**Система сборки** – это инструмент или набор инструментов, используемых для автоматизации процесса преобразования исходного кода программы в исполняемый файл или библиотеку

Основная задача системы сборки – управлять различными этапами этого процесса, такими как компиляция, компоновка, препроцессинг и другие задачи, связанные с подготовкой программного обеспечения к выполнению

Если упростить концепцию работы таких систем, то они позволяют описать процесс сборки на некотором внутреннем языке, команды которого по сути просто приведут к вызову соответствующих утилит с заданными параметрами

Системы сборки **абстрагируют** различия между платформами, позволяя собирать один и тот же проект под Windows, Linux и macOS **без ручного вмешательства**.

CMake не является традиционной системой сборки

**СMake – это система мета-сборки**

**Система мета-сборки** – это инструмент, который генерирует файлы конфигурации для других систем сборки. В отличие от традиционных систем сборки, которые напрямую управляют процессом компиляции и компоновки, мета-сборки создают промежуточные файлы, которые затем используются другими инструментами для выполнения фактической сборки

Основные особенности систем мета-сборки:

* **Генерация конфигураций**: Мета-сборки создают файлы конфигурации (например, Makefile, Ninja файлы), которые затем используются другими инструментами для сборки проекта
* **Кроссплатформенность**: Они обеспечивают возможность генерации конфигураций для различных платформ и компиляторов, что делает их особенно полезными для кроссплатформенных проектов
* **Абстракция**: Мета-сборки предоставляют более высокий уровень абстракции, позволяя разработчикам описывать процесс сборки в более общем виде, не привязываясь к конкретным инструментам сборки
* **Гибкость**: Поддерживают различные сценарии сборки и позволяют легко изменять конфигурацию сборки в зависимости от требований проекта

Для того, чтобы сгенерировать файлы сборки для какого-либо проекта необходимо добавить в корневую папку проекта конфигурационный файл CMakeLists.txt

Посмотреть список доступных генераторов cmake –help. \* помечен генератор по умолчанию.

Out-source сборка – это хранение бинарников и исходников в разных папках

|  |
| --- |
| cmake\_minimum\_required(VERSION 3.10)  project(Lab-01C C)  add\_executable(Lab-01C Lab-01C/Lab-01C.c) |

**Генератор** – по сути, это та система сборки для которой будут сгенерированы файлы сборки

cmake\_minimum\_required(VERSION минимальная версия CMake, которая нужна для кофигурирования проекта)

project(имя проекта + можно поставить языки программирования)

add\_executable - создать исполняемый файл из набора файл с исходным кодом



Генератор указывается через флаг –G

Через флаг –B был указан каталог в котором будут размещены файлы сборки проекта (что по сути делает сборку «out-of-source»)

Флаг –T указывает на используемые инструменты для сборки (данный параметр в данном случае необходим для указания Clang в качестве компилятора, т.к. стандартный способ DCMAKE\_C\_COMPILER задания не работает с VS)

После создания файлов сборки необходимо собственно собрать проект. Для этого необходимо вызвать тот сборщик, генератор которого был указан в параметре –G

CMAKE\_SYSTEM\_NAME — это переменная в CMake, которая указывает название операционной системы, на которой выполняется процесс сборки.

CMAKE\_BINARY\_DIR – корень директории сборки

Прописываем cmake и видим строки Configurating done – сначала конфигурирование (чтение файла CMakeLists.txt и строит свою внутреннюю презентацию проекта), если успешно, то дальше стартует генерация проектных файлов – Generating done.

После генерации в build создаётся файл CMakeCache.txt, при повторном запуске всё быстрее, так как он используется.

Для сборки:

**Cmake -G “генератор” путь\_к\_ CMakeCache.txt**

Параметр --build в CMake используется для **сборки проекта** после генерации файлов сборки.

**cmake --build <путь\_к\_директории> [опции]**

Где <путь\_к\_директории> — это каталог, содержащий файлы сборки (например, build/).

Можем указать конфигурацию, в которой мы хотим забилдить проект –Config Release (иначе по умолчанию)

--target – выбрать конкретную цель конфигурацию

Переменная задаётся через set(имя значение). Все переменные строки. Ставим кавычки на значении, если там есть пробелы. Вывод необъявленной переменной = пустая строка.

Получение значения переменной ${имя переменной}

set(EXECUTABLE\_OUTPUT\_PATH "${CMAKE\_BINARY\_DIR}"). - определяет, куда будут **собираться** исполняемые файлы. CMAKE\_BINARY\_DIR — это **каталог, где происходит сборка** (обычно build/).

add\_subdirectory – подключает подпроект, которые должен содержать свой CMakeLists.txt

**Обязательно ли писать set(EXECUTABLE\_OUTPUT\_PATH "${CMAKE\_BINARY\_DIR}")?**

**Нет, не обязательно.** Но если ты хочешь задать конкретную папку, куда будет помещаться исполняемый файл, тогда да, нужно.

**По умолчанию без set(EXECUTABLE\_OUTPUT\_PATH ...):**

* Если не указывать EXECUTABLE\_OUTPUT\_PATH, то CMake **по умолчанию** помещает исполняемые файлы:
  + В **тот же каталог**, где ты вызвал cmake --build . (обычно build/).
  + В **подкаталоги внутри CMAKE\_BINARY\_DIR** (например, Debug/, Release/ в зависимости от типа сборки).

CMAKE\_SYSTEM\_NAME — это **предопределённая переменная CMake**. Определяет **название операционной системы**, на которой выполняется CMake.

🔹 **Вывод message(...) происходит во время генерации сборки**, когда CMake **обрабатывает CMakeLists.txt**.

* Он **НЕ выводится во время компиляции**, а только при выполнении cmake ...
* -G указывает **генератор сборки** (например, "Visual Studio 17 2022").
* -T указывает **инструментарий компиляции**, который будет использован (например, ClangCL или v142).

**CMAKE\_C\_COMPILER** и **CMAKE\_CXX\_COMPILER** — это переменные, которые указывают путь к компиляторам C и C++ соответственно.

**CMAKE\_FIND\_ROOT\_PATH** — указывает CMake корневой путь для поиска библиотек и заголовочных файлов целевой платформы.

**CMAKE\_FIND\_ROOT\_PATH\_MODE\_INCLUDE ONLY**: CMake будет искать **заголовочные файлы** только в пути, указанном в **CMAKE\_FIND\_ROOT\_PATH**.

**CMAKE\_FIND\_ROOT\_PATH\_MODE\_LIBRARY ONLY**: CMake будет искать **библиотеки** только в этом же пути.

toolchain.cmake — это конфигурационный файл, используемый в CMake для настройки **кросс-компиляции**. Он сообщает CMake, как собирать проект для целевой платформы, отличной от текущей.