Семинар #2: CMake

Часть 1: Системы сборки

Сборка (англ. *build*) – это процесс трансформации исходного кода и ресурсов в исполняемые исполняемые файлы и/или библиотеки.

Система сборки (англ. *build system*) – это программа (или набор программ), предназначенная для автоматизации преобразования исходного кода в исполняемые файлы или библиотеки.

Задачи, которые решают системы сборки

Представьте, что есть проект, написанный на C++, содержащий сотни файлов и зависящий от многих сторонних библиотек. Как его скомпилировать? Можно просто вызвать компилятор `g++` из командной строки и передать ему все файлы проекта, информацию о всех подключаемых библиотеках и все опции компиляции и линковки необходимые для проекта. В теории так можно скомпилировать даже большой проект, однако такой подход имеет множество недостатков. Например, вводимая команда будет очень большой и в ней можно будет легко запутаться. Но это только одна из многих проблем данного подхода.

Вот какие проблемы могут возникнуть, если компилировать программу просто с помощью компилятора (например, `g++`), без использования систем сборки:

- **Большое количество файлов исходного кода**
  Проект может содержать сотни и тысячи файлов исходного кода. Для того, чтобы скомпилировать такой проект придётся передать все эти файлы компилятору. Можно будет легко ошибиться в названии какого-либо файла или просто забыть один из файлов.

- **Множество опций компиляции**
  Часто необходимо скомпилировать исходные файлы с использованием множества различных опций компиляции и линковки. Причём, часть файлов должны быть скомпилированы с одними опциями, а другая часть – с другими опциями. Опции компиляции могут меняться при различных сборках. Например, в один момент мы можем захотеть собрать программу так, чтобы она была как можно более быстрой (то есть передать всем файлам опцию `-O3`). В другой раз мы можем захотеть собрать проект для его дебага (в этом случае нужно будет передать опции `-O0 -g`).

- **Проект может зависеть от внешних библиотек**
  Если проект зависит от внешних библиотек, то каждую библиотеку придётся подключить, используя специальные опции для компиляции (например, `-I`, `-L` и `-l`). Внешние библиотеки могут быть как статическими, так и динамическими, это также нужно учесть при сборке.

- **Внешние библиотеки необходимо найти в системе**
  Если вы собирает проект на своей машине, то вы можете указать полные пути до внешних библиотек. Но если вы создаёте программу, которая должна компилироваться на других системах, то нужно учесть, что на разных компьютерах библиотеки могут находиться в разных местах в системе. Более того, нужных библиотек может просто не быть в системе. Или на системе может находиться нужная библиотека, но быть не подходящей версии. В этих случаях нужно будет вывести сообщение об ошибке либо попытаться загрузить библиотеку из сети прямо во время сборки.

- **Несколько исполняемых файлов и/или библиотек в одном проекте**
  Результатом сборки проекта необязательно является один файл. Проект может содержать множество исполняемых файлов и/или библиотек. Приведём расспространённые примеры когда это случается:

  - **Разная функциональность**: Например, один исполняемый файл – это само приложение, а другой исполняемый файл – это редактор настроек основного приложения.
  - **Разный интерфейс**: Например, один исполняемый файл – это приложение, которое вызывается только через командную строку, а второй файл – это то же самое приложение, но с графическим интерфейсом.

- **Компонентная архитектура приложения**: Например, один исполняемый файл – это приложение-сервере, а второй файл – приложение-клиент.
- **Несколько компонент одной библиотеки**: Если библиотека большая, то она может делиться на несколько компонент. Каждая компонента по сути представляет собой самостоятельную библиотеку. Пользователь может выбрать какие компоненты библиотеки он хочет использовать.

Для компиляции каждого исполняемого файла и каждой библиотеки необходимо вызвать компилятор, корректно указав исполнямые файлы, опции компилятора и подключив внешние библиотеки. Естественно, опции компилятора будут разными для разных компонент проекта. Использование систем сборки позволяет упростить этот процесс.

- **Инкрементальная сборка**
  Предположим, что вы изменили один файл в уже скомпилиронном проекте. Чтобы собрать проект заново, можно перекомпилировать весь проект, однако это можно занять очень много времени (компиляция больших проектов может занимать несколько часов). Вместо этого можно использовать объектые файлы, сохранившиеся с момента предыдущей компиляции для гораздо более быстрой сборки. Если вы не используете систему сборки, то вам придётся самим хранить все эти файлы, самим определять какие файлы не сохранились с момента предыдущей сборки и перекомпилировать заново только изменившиеся файлы. Система сборки берёт эту работу на себя.

- **Кроссплатформенность**
  Мы хотим, чтобы наш проект собирался на разных архитектурах и операционных системах. Но сборка на разных системах может сильно отличаться. Может быть разным всё что угодно: разные системные библиотеки, разные компиляторы, разные опции компиляции, разные форматы файлов, разные расширения файлов и так далее вплоть до разного формата написания путей. Многие системы сборки берут на себя большую часть работы по сборке проекта на разных системах.

- **Поддержка различных компиляторов**
  Даже в рамках одой системы могут существовать множество различных компиляторов C++. Помимо `GCC` популярны такие компиляторы как `Clang` и `MSVC`. Есть также множество других компиляторов. Если мы захотим по какой-либо причине поменять компилятор, то мы сможем легко это сделать только если мы используем систему сборки.

- **Работа с дополнительными файлами проекта**
  Помимо файлов исходного кода и файлов библиотек, проект может содержать множество других файлов, таких как файлы настроек проекта, скрипты на других языках программирования, файлы документации, ресурсы проекта (например, изображения значков в программе или текстуры в игре). При сборке проекта может понадобиться, например, проверить что такие файлы существуют или, например, разархивировать их или скачать файлы из сети.

Системы сборки для языков C и C++

Для языка C++ существуют множество систем сборки, вот самые популярные из них:

- **Make** – классическая система сборки, часто используется в Linux для языка C.
- **CMake** – самая популярная система сборки для проектов на языке C++.
  Является не просто системой сборки, но и генератором для других систем.
- **Meson** – альтернатива CMake, но менее популярна.
- **Bazel** – система сборки для разных языков (C++, Java, Python), разработанная Google.
- **Ninja** – минималистичная система сборки, ориентированная на скорость.
- **MSBuild** – система сборки, разработанная Microsoft, используется в IDE Visual Studio.
- **Qmake** – система сборки библиотеки Qt, но может использоваться и для обычных проектов.

Часть 2: Что такое CMake?

- **CMake – это генератор систем сборки**
  CMake не компилирует код напрямую, а создаёт файлы сборки для других систем, таких как Make, Ninja, Visual Studio или других. Вы можете указать, какую систему использовать. После генерации файлов, соответствующая система собирает проект.

- **CMake – это система сборки**
  CMake может автоматически запускать системы сборки для компиляции проекта. Это позволяет полностью собирать проект, используя только CMake. Например, можно сгенерировать проект для Visual Studio и скомпилировать его без запуска самой Visual Studio. Поэтому CMake часто считают не просто генератором, а полноценной системой сборки.

- **CMake – это язык программирования**
  CMake использует собственный скриптовый язык для описания сборки. Этот язык проще традиционных языков программирования, но при этом остаётся полноценным — он поддерживает переменные, функции и условия, позволяя гибко настраивать процесс сборки.

Установка CMake

Windows, с использованием пакетного менеджера MSYS2

Если вы используете MSYS2, то CMake можно установить, открыв терминал MSYS2 (`Пуск` → `MSYS2 MSYS`) и выполнив следующую команду:

```
$ pacman -S cmake
```

или такую команду:

```
$ pacman -S mingw-w64-x86_64-cmake
```

Windows, с использованием установщика

Перейти на оффициальный сайт: `cmake.org` и скачать установщик для вашей системы оттуда. Для Windows вам потребуется скачать следующее: `cmake.org` → Download → Binary distributions → Windows x64 Installer.

Linux

Используйте пакетный менеджер системы, например:

```
sudo apt install cmake
```

Сборка HelloWorld с помощью CMake

CMake как генератор систем сборки

Часть 3: Простые проекты с использованием CMake

Часть 4: Таргеты

`PUBLIC`, `PRIVATE` и `INTERFACE`

Часть 5: CMake как язык программирования

Печать на экран. Команда `message`.

Переменные в CMake

Команда `string`

Списки в CMake. Команда `list`.

Алгоритм передачи аргументов в функции

Команда `if`

Алгоритм определения истинности/ложности выражения, передаваемого в `if`

Пусть в `if` передаётся одна строка, которую обозначим как `value`:

```
if(value)
```

Для определения истинности/ложности выражения `value`, используется следующий алгоритм:

- Если `value` это одна из следующих строк, то `value` расценивается как истина:
  - `ON`, `YES`, `TRUE`, `Y`
  - Одна из перечисленных строк, с изменёнными регистрами символов.
  - Строка, представляющее ненулевое число.
- Если `value` это одна из следующих строк, то `value` расценивается как ложь:
  - `OFF`, `NO`, `FALSE`, `N`, `IGNORE`, `NOTFOUND`
  - Строка, оканчивающаяся на `-NOTFOUND`
  - Одна из перечисленных строк, с изменёнными регистрами символов.
  - Строка, представляющее число 0.
  - Пустая строка.
- В остальных случаях действуют следующие правила:
  - Если `value` это строка, обрамлённая в "кавычки", то `value` расценивается как ложь.
  - Если `value` это строка, не обрамлённая в "кавычки", то `value` воспринимается как переменная. Значение этой переменной автоматически подставляется. Дальше, используются правила, описанные выше в первых двух пунктах алгоритма. В частности, если переменной с данным именем не существует, то она воспринимается как пустая строка и расценивается как ложь. Если переменная имеет значение неописанное в первых двух пунктах алгоритма, то `value` расценивается как истина.

Циклы CMake

Функции CMake

Работа с файлами в CMake