Задания 02

Системы сборки

Введение

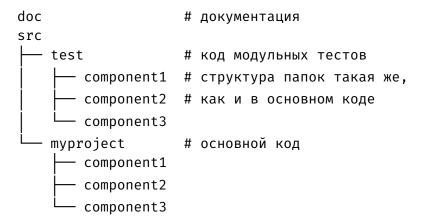
Самым важным элементом любого проекта является система сборки — программа, которая генерирует команды для сборки вашего исходного кода. Чем быстрее эта система собирает код и чем больше рутинных операций автоматизирует, тем быстрее идет разработка, и тем проще настроить непрерывную интеграцию — автоматизированную сборку и тестирование вашей программы. В задачи системы сборки входит

- поиск зависимостей (заголовочных файлов и библиотек),
- генерация различных версий кода в зависимости от платформы, на которой происходит сборка,
- генерация вспомогательных файлов,
- генерация команд для компиляции всех исходный файлов.

Как правило, системы сборки поддерживают опции для включения или отключения различных компонент программы. Результатом работы системы сборки является директория, в которой находятся сгенерированные файлы, а также файл с дальнейшими командами для подчиненной (более низкоуровневой) системы сборки. К высокоуровневым системам относятся autoconf, cmake, meson, к низкоуровневым — make, ninja. Мы будем изучать meson и ninja.

Структура проекта

Любой проект начинается с дерева директорий, в каждой из которых находится файл для сборки. В meson это файл meson.build. В проектах на C++ с небольшими вариациями используется следующее дерево директорий.



Здесь myproject — имя проекта, component N — логическая единица проекта. Как правило, каждая компонента собирается в отдельную библиотеку (или исполняемый файл), которая затем присоединяется к основному исполняемому файлу. В больших проектах из одной компоненты могут получиться несколько библиотек или исполняемых файлов. При такой схеме название проекта и название компоненты являются частью пути до заголовочного файла, а имя файла совпадает с именем класса, который в нем объявлен. В больших проектах кроме класса в файле могут быть объявлены вспомогательные функции или классы. Чтобы использовать класс Ship из компоненты сотропеnt 1 в каком-либо файле проекта, его следует подключить следующим образом:

#include <myproject/component1/ship.hh>

Расширения файлов в Linux произвольны, на сборку проекта это никак не влияет.

В небольших проектах структура упрощается путем исключения директорий компонент и хранении всех файлов с исходным кодом в директории myproject. Именно такую структуру я вам рекомендую использовать в заданиях.

Meson build

Корневой файл meson.build для вышеописанной упрощенной структуры выглядит следующим образом.

```
project(
    'myproject', # название проекта
    'cpp', # язык
    version: '0.1.0', # версия кода
    meson_version: '>=0.46', # минимально поддерживаемая версия Meson
    default_options: ['cpp_std=c++11'] # используемый стандарт C++
)
subdir('src')
```

Команда project должна быть первой командой в корневом файле. Команда subdir исполняет команды из файла meson.build в директории src. Этот файл выглядит так.

```
# сохранение пути до текущей директории
# для подключения заголовочный файлов
# в виде <myproject/...>
src = include_directories('.')
subdir('myproject')
Файл meson.build в myproject выглядит так.
myproject src = files([
    'main.cc' # список исходных файлов
])
executable(
    'myproject', # название исполняемого файла
    include_directories: src, # где ищутся заголовочные файлы
    sources: myproject_src, # список исходный файлов
    dependencies: [], # зависимости проекта (если имеются)
    install: true # устанавливать ли файл
)
```

Для инициализации директории, в которой будет происходить сборка, нужно ввести следующие команды в корне проекта.

```
meson . build
cd build
ninja
```

После первичной инициализации после любого изменения кода достаточно набрать ninja для пересборки проекта. При этом пересоберется только измененная и зависимые от нее части кода.

Задания

- 1. Создайте собственный класс Vector по образу и подобию std::vector. Класс должен иметь
 - методы begin, end, size,
 - методы push_back (с копированием и перемещением), pop_back, erase.
 - конструктор копирования, move-конструктор,
 - оператор присваивания с копированием и с перемещением,
 - метод swap внутри класса, через который реализуется перемещение,
 - функция swap вне класса для совместимости с STL алгоритмами.

Класс должен быть шаблоном, единственный аргумент которого является типом элемента контейнера. Класс не должен использовать другие классы STL (в особенности, класс std::vector).

- 2. Проверьте работоспособность класса с помощью следующей программы.
 - Создается вектор из std::ofstream.
 - Вектор заполняется открытыми потоками с последовательными именами файлов (например a, b, c).
 - Перемешайте элементы вектора с помощью std::shuffle с генератором std::random_device.
 - Запишите в каждый поток его порядковый номер в векторе.

Проект нужно собрать c опцией -Db_sanitize=address, чтобы отловить возможные ошибки:

meson configure -Db sanitize=address

Ссылки:

- Описание методов std::vector: ru.cppreference.com.
- Peaлизация std::vector в стандартной библиотеке компилятора GCC: /usr/include/c++/8/bits/stl_vector.h. Этот файл есть в любом дистрибутиве Linux, в котором установлен компилятор, но путь может отличаться.