CMake для сборки проекта

Сборка проекта

- в каталоге исходных кодов:

```
mkdir build
cd build
cmake ..
```

Сборка проекта

```
make
```

или в общем случае

```
cmake --build  . --parallel
```

Hello, World

- main.cpp

```
#include <iostream>
int main(int argc, char** argv)
{
    std::cout << "Hello, World!" << std::endl;
    return 0;
}
```

- CMakeLists.txt

```
cmake_minimum_required(VERSION 3.20)
project(hello)
add_executable(main main.cpp)
```

Разделение кода на *.cpp и *.h файлы

- main.cpp

```
#include "foo.h"
int main(int argc, char** argv)
{
    hello_world();
    return 0;
}
```

- foo.h

```
void hello_world();
```

- foo.cpp

```
#include <iostream>
void hello_world()
{
    std::cout << "Hello, World!" << std::endl;
}
```

- CMakeLists.txt

```
cmake_minimum_required(VERSION 3.20)
project(hello_world)
set(SOURCE_EXE main.cpp)
set(SOURCE_LIB foo.cpp)
# STATIC/SHARED
add_library(src_lib STATIC ${SOURCE_LIB})
add_executable(main ${SOURCE_EXE})
target_link_libraries(main src_lib)
```

Подпроект

- main.cpp

```
#include "foo.h"
int main(int argc, char** argv)
{
    hello_world();
    return 0;
}
```

- подпроект «foo», каталог «./foo/»:
  - foo.h

```
void hello_world();
```

  - foo.cpp

```
#include <iostream>
void hello_world()
{
    std::cout << "Hello, World!" << std::endl;
}
```

  - CMakeLists.txt

```
cmake_minimum_required(VERSION 3.20)
project(foo)
set(SOURCE_LIB foo.cpp)
# build "foo" library
add_library(foo STATIC ${SOURCE_LIB})
```

- CMakeLists.txt

```
cmake_minimum_required(VERSION 3.20)
project(hello_world)
set(SOURCE_EXE main.cpp)
include_directories(foo)
add_executable(main ${SOURCE_EXE})
add_subdirectory(foo)
target_link_libraries(main foo)
```

Внешняя библиотека boost и *.cpp, *.h файлы

- main.cpp

```
#include "foo.h"
#include "lib.h"

int main(int argc, char** argv)
{
    hello_world();
    hello_lib(argc, argv);
    return 0;
}
```

- foo.h

```
void hello_world();
```

- foo.cpp

```
#include <iostream>
void hello_world()
{
    std::cout << "Hello, World!" << std::endl;
}
```

- lib.h

```
void hello_lib(int, char**);
```

- lib.cpp

```
#include <boost/program_options/options_description.hpp>
#include <boost/program_options/option.hpp>
#include <boost/program_options/variables_map.hpp>
#include <boost/program_options/parsers.hpp>
#include <iostream>

namespace po = boost::program_options;

void hello_lib(int argc, char** argv)
{
    po::options_description desc("Allowed options");
    desc.add_options()
```

```
        ("help, h", "This is help message")
        ;

    po::variables_map vm;

    try {
        po::store(po::parse_command_line(argc, argv, desc), vm);
        po::notify(vm);
    }
    catch (const boost::wrapexcept<po::unknown_option>& e) {
        std::cerr << "Error: Unknown option: " << e.get_option_name() <<
std::endl;
        std::cerr << "Use --help for usage information." << std::endl;
    }

    if (vm.count("help")) {
        std::cout << desc << "\n";
    }
}
```

- CMakeLists.txt

```
cmake_minimum_required(VERSION 3.20)
project(hello_world)
find_package( Boost 1.40 COMPONENTS program_options REQUIRED )
include_directories( ${Boost_INCLUDE_DIR} )
set(SOURCE_EXE main.cpp)
set(SOURCE_LIB foo.cpp lib.cpp)
add_library(src_lib STATIC ${SOURCE_LIB})
add_executable(main ${SOURCE_EXE})
target_link_libraries(main src_lib ${Boost_LIBRARIES})
```

Внешняя библиотека boost и подпроект foo

- main.cpp

```
#include "foo.h"
#include "lib.h"

int main(int argc, char** argv)
{
    hello_world();
    hello_lib(argc, argv);
    return 0;
}
```

- подпроект «foo», каталог «./foo/»:
  - foo.h

    ```
    void hello_world();
    ```

  - foo.cpp

```
#include <iostream>
void hello_world()
{
    std::cout << "Hello, World!" << std::endl;
}
```

- CMakeLists.txt

```
cmake_minimum_required(VERSION 3.20)
project(foo)
set(SOURCE_LIB foo.cpp)
# build "foo" library
add_library(foo STATIC ${SOURCE_LIB})
```

- lib.h

```
void hello_lib(int, char**);
```

- lib.cpp

```
#include <boost/program_options/options_description.hpp>
#include <boost/program_options/option.hpp>
#include <boost/program_options/variables_map.hpp>
#include <boost/program_options/parsers.hpp>
#include <iostream>

namespace po = boost::program_options;

void hello_lib(int argc, char** argv)
{
    po::options_description desc("Allowed options");
    desc.add_options()
        ("help, h", "This is help message")
        ;

    po::variables_map vm;

    try {
        po::store(po::parse_command_line(argc, argv, desc), vm);
        po::notify(vm);
    }
    catch (const boost::wrapexcept<po::unknown_option>& e) {
        std::cerr << "Error: Unknown option: " << e.get_option_name() <<
std::endl;
        std::cerr << "Use --help for usage information." << std::endl;
    }

    if (vm.count("help")) {
        std::cout << desc << "\n";
    }
}
```

- CMakeLists.txt

```
cmake_minimum_required(VERSION 3.20)

project(hello_world)

find_package( Boost 1.40 COMPONENTS program_options REQUIRED )
include_directories( ${Boost_INCLUDE_DIR} )

include_directories(foo)
add_subdirectory(foo)

set(SOURCE_EXE main.cpp)
set(SOURCE_LIB lib.cpp)

add_library(src_lib STATIC ${SOURCE_LIB})

add_executable(main ${SOURCE_EXE})
target_link_libraries(main src_lib foo ${Boost_LIBRARIES})
```

Подпроект с внешней библиотекой boost и подпроект foo

- main.cpp

```
#include "foo.h"
#include "lib.h"

int main(int argc, char** argv)
{
    hello_world();
    hello_lib(argc, argv);
    return 0;
}
```

- подпроект «foo», каталог «./foo/»:
  - foo.h

```
void hello_world();
```

  - foo.cpp

```
#include <iostream>
void hello_world()
{
    std::cout << "Hello, World!" << std::endl;
}
```

  - CMakeLists.txt

```
cmake_minimum_required(VERSION 3.20)
project(foo)
set(SOURCE_LIB foo.cpp)
# build "foo" library
```

```
add_library(foo STATIC ${SOURCE_LIB})
```

- подпроект «lib», каталог «./lib/»:
  - lib.h

```
void hello_lib(int, char**);
```

  - lib.cpp

```
#include <boost/program_options/options_description.hpp>
#include <boost/program_options/option.hpp>
#include <boost/program_options/variables_map.hpp>
#include <boost/program_options/parsers.hpp>
#include <iostream>

namespace po = boost::program_options;

void hello_lib(int argc, char** argv)
{
    po::options_description desc("Allowed options");
    desc.add_options()
        ("help, h", "This is help message")
        ;

    po::variables_map vm;

    try {
        po::store(po::parse_command_line(argc, argv, desc), vm);
        po::notify(vm);
    }
    catch (const boost::wrapexcept<po::unknown_option>& e) {
        std::cerr << "Error: Unknown option: " << e.get_option_name() <<
std::endl;
        std::cerr << "Use --help for usage information." << std::endl;
    }

    if (vm.count("help")) {
        std::cout << desc << "\n";
    }
}
```

  - CMakeLists.txt

```
cmake_minimum_required(VERSION 3.20)
project(lib)
find_package( Boost 1.40 COMPONENTS program_options REQUIRED )
include_directories( ${Boost_INCLUDE_DIR} )
set(SOURCE_LIB lib.cpp)
# build "lib" library
add_library(lib STATIC ${SOURCE_LIB})
target_link_libraries(lib ${Boost_LIBRARIES})
```

- CMakeLists.txt

```
cmake_minimum_required(VERSION 3.20)
project(hello_world)
include_directories(foo)
add_subdirectory(foo)
include_directories(lib)
add_subdirectory(lib)
set(SOURCE_EXE main.cpp)
add_executable(main ${SOURCE_EXE})
target_link_libraries(main foo lib)
```

[Qt6] Hello, World

- main.cpp

```
#include <QCoreApplication>
#include <QDebug>

int main(int argc, char** argv)
{
    QCoreApplication app(argc, argv);
    qDebug() << "Hello, World";
    return app.exec();
}
```

- CMakeLists.txt

```
cmake_minimum_required(VERSION 3.20)
project(qt_hello_world)
set(CMAKE_CXX_STANDARD 17)
set(CMAKE_CXX_STANDARD_REQUIRED ON)
set(CMAKE_AUTOMOC ON)
set(CMAKE_AUTORCC ON)
set(CMAKE_AUTOUIC ON)
set(CMAKE_PREFIX_PATH "/opt/Qt/6.4.1/gcc_64/")
find_package(Qt6 REQUIRED COMPONENTS Core)
set(SOURCE_EXE main.cpp)
add_executable(main ${SOURCE_EXE})
target_link_libraries(main PRIVATE Qt6::Core)
```

[Qt6] Hello, World; подпроект

- main.cpp

```
#include <QCoreApplication>
#include "lib/lib.h"
int main(int argc, char** argv)
{
    QCoreApplication app(argc, argv);
    qt_hello();
    return app.exec();
```

```
}
```

- подпроект с библитекой Qt6 «lib», каталог «./lib/»:
  - файл lib.cpp

```
#include <QDebug>
void qt_hello() {
    qDebug() << "Hello, World";
}
```

  - файл lib.h

```
void qt_hello();
```

  - файл CMakeLists.txt

```
cmake_minimum_required(VERSION 3.20)
project(qt_hello)
# Find Qt6 components required for this module
find_package(Qt6 REQUIRED COMPONENTS Core)
set(SOURCE_LIB lib.cpp)
add_library(lib STATIC ${SOURCE_LIB})
target_link_libraries(lib PRIVATE Qt6::Core)
```

- CMakeLists.txt

```
cmake_minimum_required(VERSION 3.20)
project(qt_hello_world)
set(CMAKE_CXX_STANDARD 17)
set(CMAKE_CXX_STANDARD_REQUIRED ON)
set(CMAKE_AUTOMOC ON)
set(CMAKE_AUTORCC ON)
set(CMAKE_AUTOUIC ON)
set(CMAKE_PREFIX_PATH "/opt/Qt/6.4.1/gcc_64/")
find_package(Qt6 REQUIRED COMPONENTS Core)
set(SOURCE_EXE main.cpp)
add_subdirectory(lib)
add_executable(main ${SOURCE_EXE})
target_link_libraries(main PRIVATE lib PRIVATE Qt6::Core)
```

CMake и ALE в VIM

ALE is the Asynchronous Linting Engine for Vim.

Проблема. Линтер не видит заголовочные файлы из внешних подпроектов (используемых библиотек) и выдаёт ошибки в заголовочных файлах.

- Включение генерации файла содержащего параметры сборки проекта `compile_commands.json`:

```
set(CMAKE_EXPORT_COMPILE_COMMANDS ON)
```

- Указание каталога с заголовочными файлами проекта:

```
include_directories(${CMAKE_SOURCE_DIR}/src                \\
                    ${CMAKE_SOURCE_DIR}/ext/project/include)
```

- Настройка пути до `compile_commands.json` в .vimrc

```
let g:ale_cpp_clangd_options = '--compile-commands-dir=build'
```

—

По материалам:

- статьи Введение в CMake с сайта habr.com ;
- Craig Scott - Professional CMake: A Practical Guide;
- Getting started with CMake руководства библиотеки Qt6.