cmake

安装

cmake：https://cmake.org/download/

MinGW：https://sourceforge.net/projects/mingw-w64/files/mingw-w64/

构建和运行

- 新建一个构建目录

```
mkdir build
```

- 进入该目录并配置项目

```
cd build
cmake ../src
```

如果不是使用默认的Generator，应当添加`-G`选项：

```
cmake -G "MinGW Makefiles" ../Step1
```

- 构建

```
cmake --build .
```

- 运行

说明

- cmake命令不区分大小写，但是参数、变量区分大小写
- 参数用空格或分号隔开
- 使用`${VAR}`引用变量
- 引号可加可不加，但如果字符串中有空格必须加

概念

- 目标文件（`target`）：可执行文件（`add_executable`）、库文件（`add_library`）
- 命令（cmake-command）：下面要讲的函数
- 变量（cmake-variable）：以`CMAKE_`开头的变量名
- 属性（cmake-properties）：文件/文件夹都有各自的属性

命令

cmake_minimum_required

设置最低cmake版本。

```
cmake_minimum_required(VERSION <min>)
```

```
cmake_minimum_required(VERSION 3.10)
```

project

设置项目名。

```
project(<PROJECT-NAME> [<language-name>...])
project(<PROJECT-NAME>
        [VERSION <major>[.<minor>[.<patch>[.<tweak>]]]]
        [DESCRIPTION <project-description-string>]
        [HOMEPAGE_URL <url-string>]
        [LANGUAGES <language-name>...])

# 项目名会被存储在变量 PROJECT_NAME 和 CMAKE_PROJECT_NAME 中
# PROJECT_SOURCE_DIR 等价于 <PROJECT-NAME>_SOURCE_DIR
# PROJECT_BINARY_DIR 等价于 <PROJECT-NAME>_BINARY_DIR

# 如果定义了版本号
# 版本号被保存在 PROJECT_VERSION 和 <PROJECT-NAME>_VERSION 中
# 主版本号被保存在 PROJECT_VERSION_MAJOR 和 <PROJECT-NAME>_VERSION_MAJOR 中
# 次版本号被保存在 PROJECT_VERSION_MINOR 和 <PROJECT-NAME>_VERSION_MINOR 中
```

```
project(Tutorial)
project(Tutorial C CXX)
project(Tutorial VERSION 2.3 LANGUAGES CXX)
```

add_executable

用指定的源文件为项目添加可执行文件。

```
add_executable(<name> [WIN32] [MACOSX_BUNDLE]
               [EXCLUDE_FROM_ALL]
               [source1] [source2 ...])

# <name>即生成可执行文件的名字（与项目名没有关系），在一个项目中必须唯一
# 如windows系统会生成<name>.exe文件
```

```
add_executable(Tutorial tutorial.cxx)
```

message

打印信息。

```
message([<mode>] "message text" ...)

# STATUS 前缀为--的信息
# SEND_ERROR 产生错误，跳过生成过程
# FATAL_ERROR 产生错误，终止运行
```

set

将变量设置为指定值。

```
set(<variable> <value>)
```

设置C++标准

```
set(CMAKE_CXX_STANDARD 11)
```

设置输出文件位置

```
# 设置运行时目标文件（exe、dll）的输出位置
set(CMAKE_RUNTIME_OUTPUT_DIRECTORY ${CMAKE_BINARY_DIR}/bin)

# 设置存档目标文件（lib、a）的输出位置
set(CMAKE_ARCHIVE_OUTPUT_DIRECTORY ${CMAKE_BINARY_DIR}/lib)
```

option

定义一个开关。

```
option(<variable> "<help_text>" [value])

# value的值为 ON 或 OFF ，默认为 OFF
# 命令行 -D<variable>=ON/OFF
```

configure_file

将输入文件进行替换并生成输出文件。

```
configure_file(<input> <output>)

# 输入文件中形如 @VAR@ 或 ${VAR} 的字符串会被替换为这些变量的当前值，如果未定义则被替换为空字符串
# 其他规则见下
```

```
#cmakedefine VAR ...
// 会被替换为以下两行之一，取决于VAR是否被设置
#define VAR ...
/* #undef VAR */
```

例:

```
// config.h.in文件：
#cmakedefine FOO_ENABLE
#cmakedefine FOO_STRING "@FOO_STRING@"
```

```
# CMakeLists.txt文件
option(FOO_ENABLE "Enable Foo" ON)
if(FOO_ENABLE)
    set(FOO_STRING "foo")
endif()
configure_file(foo.h.in foo.h)
```

```
// 如果FOO_ENABLE为ON，则生成以下内容的.h文件
#define FOO_ENABLE
#define FOO_STRING "foo"

// 如果FOO_ENABLE为OFF，则生成以下内容的.h文件
/* #undef FOO_ENABLE */
/* #undef FOO_STRING */
```

include_directories

指定所有目标的头文件路径。

```
include_directories(dir1 [dir2 ...])

# 目录会被添加到当前文件的 INCLUDE_DIRECTORIES 属性中
# 当前文件的每一个目标文件的 INCLUDE_DIRECTORIES 属性也会添加该目录
```

target_include_directories

指定目标的头文件路径。

```
target_include_directories(<target>
                           <INTERFACE|PUBLIC|PRIVATE> [items1...]
                           [<INTERFACE|PUBLIC|PRIVATE> [items2...] ...])

# 目标文件有 INCLUDE_DIRECTORIES 和 INTERFACE_INCLUDE_DIRECTORIES 两个属性
# INCLUDE_DIRECTORIES 对内头文件目录
# INTERFACE_INCLUDE_DIRECTORIES 对外头文件目录
```

	INCLUDE_DIRECTORIES	INTERFACE_INCLUDE_DIRECTORIES
PRIVATE	√	
INTERFACE		√
PUBLIC	√	√

参考：https://zhuanlan.zhihu.com/p/82244559

add_subdirectory

添加源文件目录。

```
add_subdirectory(source_dir [binary_dir] [EXCLUDE_FROM_ALL])

# binary_dir 指定编译结果存放的位置
```

add_library

用指定的源文件生成库。

```
add_library(<name> [STATIC | SHARED | MODULE]
            [EXCLUDE_FROM_ALL]
            [<source>...])

# STATIC 静态库
# SHARED 动态库
# 生成的库文件名为 lib<name>.xxx
```

target_link_libraries

为目标链接库。

```
target_link_libraries(<target>
                      <PRIVATE|PUBLIC|INTERFACE> <item>...
                     [<PRIVATE|PUBLIC|INTERFACE> <item>...]...)

# item 可以是target名、绝对路径（必须保证文件存在）
```

区分

```
# 头文件目录
include_directories()
target_include_directories()

# 链接时库目录
link_directories()
target_link_directories()

# 链接库
link_libraries()
target_link_libraries()

# 都推荐使用以target_开头的函数
```

安装

cmake代码

在对应目录的`CMakeLists.txt`中使用。

```
install(TARGETS <target> DESTINATION <dir>)
install(FILES <file> DESTINATION <dir>)
install(PROGRAMS <非目标文件的可执行程序> DESTINATION <dir>)    # 如脚本
install(DIRECTORY <dir> DESTINATION <dir>)  # 安装目录
```

```
install(TARGETS MathFunctions DESTINATION lib)
install(FILES MathFunctions.h DESTINATION include)
install(DIRECTORY doc/ DESTINATION doc)
```

命令行

```
cmake --install .                   # 安装到默认目录 CMAKE_INSTALL_PREFIX
cmake --install . --prefix <dir>    # 安装到指定目录
```