

بسمه تعالی

آموزش CMake

Contents

۲ CMake چیست؟
۲ Build یک پروژه ساده:
۲ استفاده از رابط گرافیکی:
۴ استفاده از خط فرمان :
۴ استفاده از ابزار <code>ccmake</code> :
۵ استفاده از محیط توسعه:
۶ پارامترهای مهم در <code>cmake</code> :
۶ توابع پر کاربرد :
۷ بخش آموزش پیشرفته:
۷ ساختن یک فایل اجرایی:
۹ افزودن زیر پروژه:
۱۰ نوشتن <code>Find</code> برای کتابخانه:
۱۱ افزودن نصب کننده به پروژه:
۱۲ افزودن حذف نصب به پروژه:
۱۳ معرفی چند متغیر پر کاربرد:
۱۴ تعریف تابع در <code>CMake</code> :

CMake چیست؟

cmake یک سازنده Build system است. Build system به استاندارد می‌شود که فایل‌ها و فلگ‌ها و دستورات مورد نیاز برای کامپایل برنامه توسط کامپایلر را آماده می‌کند.

Build یک پروژه ساده:

برای build یک پروژه ساده به سه روش میتوان اقدام کرد:

- استفاده از رابط گرافیکی CMake
- استفاده از خط فرمان
- استفاده از محیط توسعه (Qt Creator, CLion, Visual Studio...)

استفاده از رابط گرافیکی:

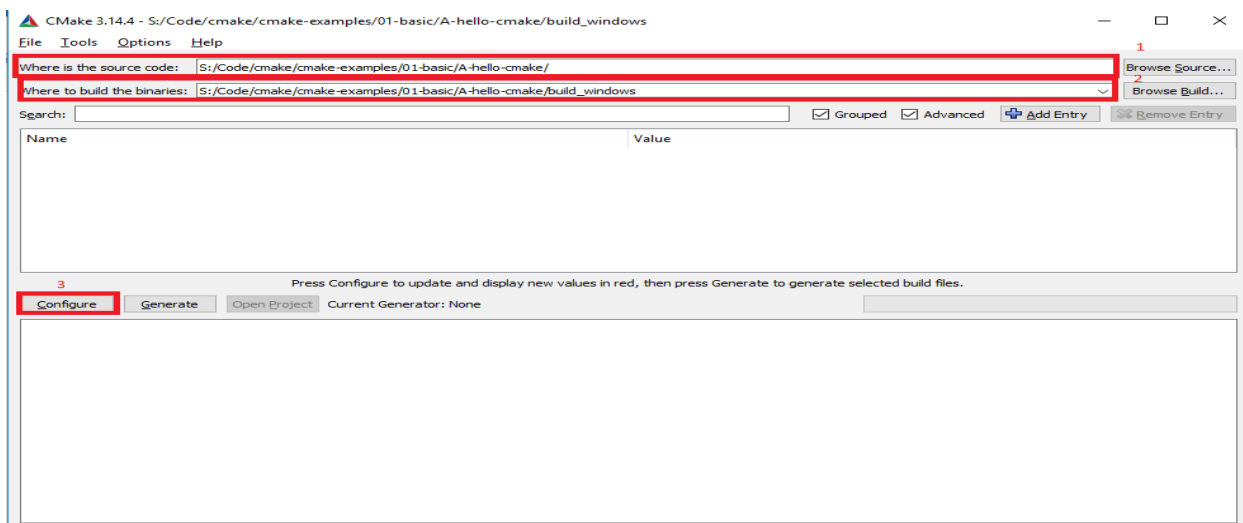
در این روش برای تولید فایل‌های بیلد سیستم پروژه از cmake-gui استفاده می‌کنیم.

نکته: بهتر است همیشه برای کار از یک پوشه به اسم build به همراه اسم سیستم عامل استفاده شود. برای مثال در ویندوز از پوشه build_windows و برای لینوکس از build_linux استفاده کنیم، دلیل این موضوع این است که ممکن است نیاز داشته باشیم یک پروژه را هم برای ویندوز و هم برای لینوکس بیلد کنیم و در صورتی که از پوشه جدا استفاده کنیم فایل‌های هر سیستم عامل جدا ساخته می‌شوند و فایل‌های بیلد سیستم‌عامل دیگر را بازنویسی نمی‌کند.

یک مثال ساده را قدم به قدم جلو می‌بریم:

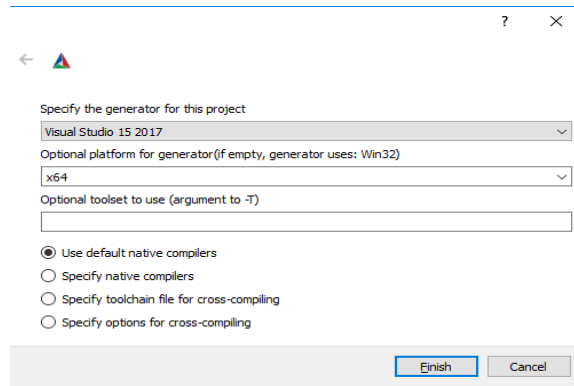
تصمیم داریم فایل‌های بیلد برنامه A-hello-cmake را با استفاده از cmake-gui و برای کامپایلر visual C++ ایجاد کنیم.

ابتدا آدرس پوشه اصلی برنامه که شامل CMakeLists.txt است را در ورودی where is the source code وارد می‌کنیم و آدرس پوشه بیلد را در ورودی where to build the binaries وارد می‌کنیم و سپس دکمه Configure را فشار می‌دهیم.

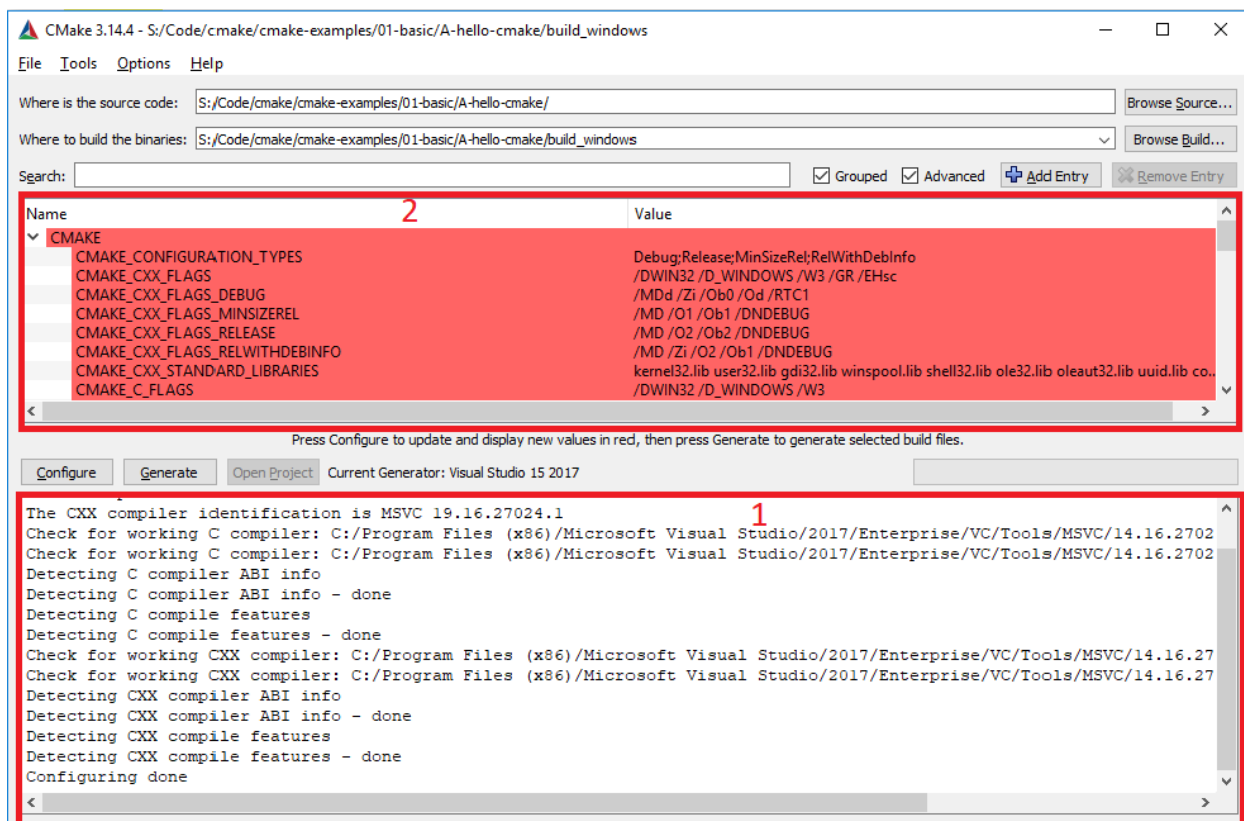


بعد از این کار در صورتی که پوشه **build** تنظیم شده توسط ما وجود نداشته باشد یک پیغام به نمایش در می آید که این پوشه وجود ندارد آیا میخواهید این پوشه ساخته شود؟ که باید گزینه **yes** را انتخاب کنیم و بعد یک دیالوگ برای انتخاب بیلد سیستم و معماری مورد نظر ما به نمایش در می آید که باید موارد مورد نیاز را انتخاب کنیم برای مثال من در ویندوز از کامپایلر **visual studio 15 2017** استفاده می کنیم و معماری **۶۴ بیتی** را مد نظر داریم.

نکته در نسخه های قدیمی تر **CMake** معماری مورد نیاز در جلوی اسم کامپایلر نوشته می شود ولی در نسخه های جدید این دو مورد به صورت جداگانه قابل انتخاب هستند.



بعد از فشردن **Finish** ابزار **CMake** کار خود را شروع کرده و پیام های خروجی را در قسمت سفید پایین صفحه (قسمت ۱ در تصویر پایین) نمایش می دهد و پارامترها نیز در قسمت بالای صفحه (مورد ۲ در تصویر پایین) نمایش داده می شوند.



بعد از کانفیگ بدون خطا باید دکمه **Generate** فشرده شود تا فایل‌های مورد نیاز ساخته شود. بعد از آن اگر محیط توسعه مناسب توسط **cmake-gui** شناسایی شود گزینه **Open Project** برای ما فعال می‌شود و میتوانیم به راحتی پروژه ساخته شده را در محیط توسعه مورد انتظار باز کنیم و ادامه روند کامپایل را پیش بگیریم.

استفاده از خط فرمان :

برای استفاده از خط فرمان که اغلب در لینوکس استفاده می‌شود نیاز است از یک محیط خط فرمانی که **cmake** را می‌شناسد استفاده کنیم برای انجام همان کاری که با **cmake-gui** انجام دادیم در کامند لاین میتوانیم از دستور زیر استفاده کنیم:

```
cmake -Bbuild_windows -H.
```

در اینجا خروجی در محیط خط فرمان نوشته می‌شود اما پارامترها نمایش داده نمی‌شود ولی میتوان با دستور زیر پارامترهای تنظیم شده را دید:

```
cmake -LA
```

و برای تنظیم یا تغییر مقدار یک متغیر میتوان در ادامه دستور **CMake** از الگوی دستوری زیر استفاده کرد:

```
-D<مقدار>=<نوع متغیر>:<نام متغیر>-D
```

مثال:

```
-DOPENCV_DIR:STRING=C:/opencv
```

البته برای سادگی میتوان انتخاب نوع متغیر را به خود **CMake** سپرد و دستور بالا را به صورت زیر خلاصه کرد:

```
-D<مقدار>=<نام متغیر>-D
```

مثال:

```
-DOPENCV_DIR=C:/opencv
```

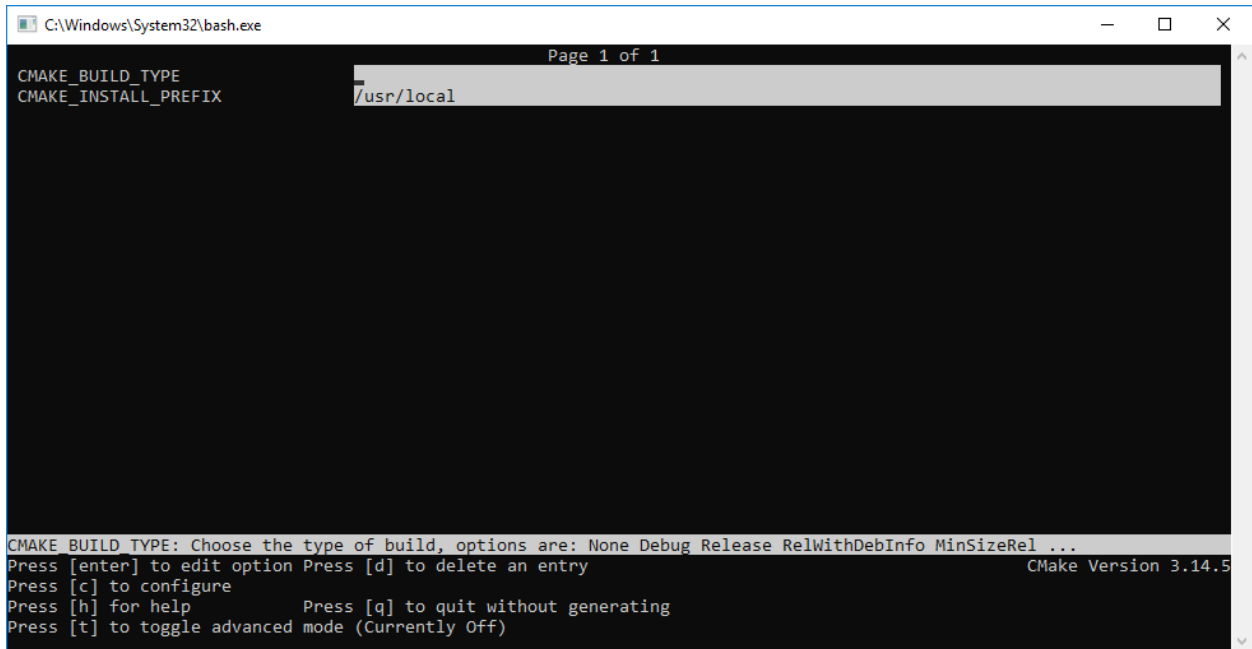
مرحله **generate** نیز به صورت خودکار انجام می‌شود.

استفاده از ابزار **ccmake**:

در لینوکس ابزاری با نام **ccmake** موجود است که می‌توانید از آن برای مشاهده و تنظیم متغیرها استفاده کنید نحوه استفاده از آن به این صورت است که باید در پوشه اصلی (**root**) که در آن **CMakeLists.txt** اصلی وجود دارد این ابزار را در ترمینال فراخوانی کرده و به عنوان پارامتر پوشه بیلد را به آن بدهید، دقت کنید که باید ابتدا **cmake** انجام شود تا این ابزار بتواند متغیرها را لود کرده و تغییر دهد. دقت کنید که تنظیم متغیر در ترمینال موجب ایجاد تغییرات در فایل **CMakeLists.txt** نخواهد شد و فقط متغیر در کش پوشه **build** تغییر خواهد کرد بنابراین اگر پوشه **build** حذف گردد و یا سی‌میک در پوشه دیگری انجام شود متغیرها همانند ابتدا و قبل از تنظیم دستی توسط شما در ترمینال خواهند بود. نحوه استفاده از **ccmake** به صورت زیر است:

```
ccmake build_linux
```

و خروجی مشاهده شده همانند تصویر زیر خواهد بود:



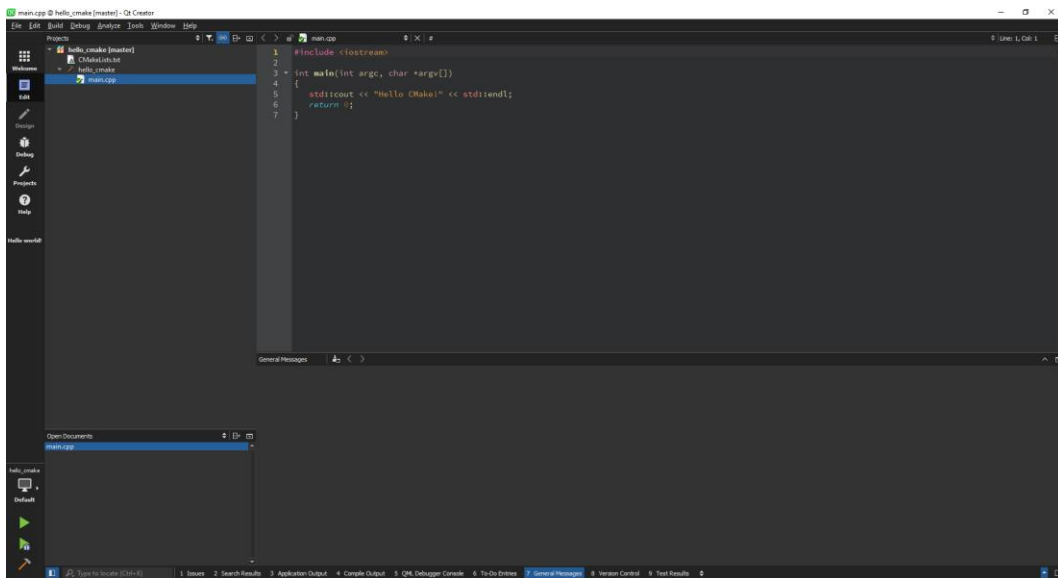
```
C:\Windows\System32\bash.exe
Page 1 of 1
CMAKE_BUILD_TYPE
CMAKE_INSTALL_PREFIX /usr/local

CMAKE BUILD TYPE: Choose the type of build, options are: None Debug Release RelWithDebInfo MinSizeRel ...
Press [enter] to edit option Press [d] to delete an entry
Press [c] to configure
Press [h] for help
Press [t] to toggle advanced mode (Currently Off)
CMake Version 3.14.5
```

برای بالا و پایین رفتن بین متغیرها می‌توانید از کلیدهای فلش بالا و پایین استفاده کنید و همانطور که در پایین تصویر موجود است برای تغییر یک متغیر کافیست کلید `enter` را فشار دهید و اقدام به تغییر نمایید.

استفاده از محیط توسعه:

برای این مورد در `Qt Creator` کافیست `open project` را زده و آدرس فایل `CMakeLists.txt` را داده و آن را باز می‌کنیم و خود `Qt Creator` به صورت خودکار یک مرحله `configure` و `generate` را انجام میدهد و فایل‌های پروژه را برای ما لیست می‌کند ولی در صورتی که خطا داشته باشیم کیوت نمی‌تواند فایل‌های پروژه را لیست کند پس بهتر است وقتی که `CMakeLists.txt` را تکمیل نمودیم و نیاز به ورود پارامتر نداشت آن را در `Qt Creator` باز کنیم.



```
main.cpp @ hello_cmake (master) - Qt Creator
1 #include <iostream>
2
3 int main(int argc, char *argv[])
4 {
5     std::cout << "Hello CMake!" << std::endl;
6     return 0;
7 }
```

پارامترهای مهم در cmake :

CMAKE_PREFIX_PATH: این پارامتر نگهدارنده آدرس(های) مورد نیاز برای پیدا کردن کتابخانه‌ها و پکیج‌هاست به این معنی که وقتی cmake میخواهد به دنبال کتابخانه یا پکیجی جستجو کنید از این آدرس(ها) استفاده می‌کند.

CMAKE_INSTALL_PATH: این آدرس مسیر پیشفرض CMake برای نصب پروژه است به این معنی که یک پروژه استاندارد نوشته شده بعد از build وقتی install می‌شود در این آدرس قرار می‌گیرد.

CMAKE_CXX_FLAGS: شامل فلگ‌های مورد نیاز برای کامپایل است.

توابع پرکاربرد :

نکته : در CMake برای دسترسی به مقدار یک متغیر از الگوی زیر استفاده می‌شود:

{نام متغیر}\$

برای مثال برای دسترسی به مقدار CMAKE_INSTALL_PREFIX و پرینت کردن آن در کد cmake به صورت زیر عمل می‌کنیم:

```
message(${CMAKE_INSTALL_PREFIX})
```

در صورتی که متغیر ورودی خالی باشد cmake خطا خواهد داد.

نکته : cmake به بزرگی و کوچکی حروف توابع حساس نیست یعنی در مثال بالا میتوان تابع را با حروف تمام بزرگ نوشت:

```
MESSAGE(${CMAKE_INSTALL_PREFIX})
```

برای تنظیم متغیر در کد میتوان از تابع set استفاده کرد که ورودی اول نام متغیر مورد نیاز است و ورودی دوم که با فاصله یا Enter جدا می‌شود مقدار متغیر است برای مثال:

```
SET(CMAKE_INSTALL_PREFIX "C:/Program Files/project_name")
```

برای افزودن preprocessors Definitions از تابع زیر استفاده می‌شود برای مثال در اینجا میخواهیم ماکرو NOMINMAX را اضافه کنیم :

```
add_definitions(-DNOMINMAX)
```

برای پیدا کردن پکیج نیز از تابع find_package استفاده می‌شود و نام پکیج به عنوان ورودی داده می‌شود که cmake با استفاده از فایل package.cmake یا packageConfig.cmake یا FindPakage.cmake به دنبال پکیج مورد نظر جستجو می‌کند. در صورتی که REQUIRED به عنوان ورودی بعدی تابع پاس داده شود به این معنی است که پیدا شدن این پکیج برای گذر از این مرحله ضروریست و در صورت پیدا نشدن خطا دریافت خواهیم کرد برای مثال برای پیدا کردن پکیج Qt5Core به صورت زیر عمل می‌کنیم:

```
find_package(Qt5Core REQUIRED)
```

یک نحوه دیگر صدا کردن تابع find_package به صورت زیر است:

```
find_package(Qt5 COMPONENTS Core Gui Widgets Network REQUIRED)
```

فراخوانی بالا معادل این است که تک به تک Qt5 را به موارد جلوی COMPONENTS اضافه کنیم و find_package بزنیم.

بخش آموزش پیشرفته:

ساختن یک فایل اجرایی:

فرض کنیم یک پروژه ساده Hello CMake داریم برای ساخت یک فایل اجرایی در CMake از تابع `add_executable` استفاده می‌کنیم. یک مثال ساده از CMake برای یک پروژه ساده با یک فایل `main.cpp` به صورت زیر است:

```
1 # Set the minimum version of CMake that can be used
2 # To find the cmake version run
3 # $ cmake --version
4 cmake_minimum_required(VERSION 3.5)
5
6 # Set the project name
7 project (hello_cmake)
8
9 # Add an executable
10 add_executable(${PROJECT_NAME} main.cpp)
11
```

نکته: متغیر `PROJECT_NAME` نگهدارنده نام پروژه است و می‌توانیم بجای آوردن نام پروژه به صورت مستقیم از نام پروژه استفاده کنیم از این متغیر استفاده کنیم و در صورت تغییر نام پروژه نیاز به تغییر در جاهای مختلف پروژه نیست.

ورودی اول تابع `add_executable` نام فایل باینریست (نام فایل `exe`) و ورودی های بعدی فایل‌هایی که نیاز داریم در فایل خروجی باشند.

ساخت یک کتابخانه استاتیک:

```

1  cmake_minimum_required(VERSION 3.5)
2
3  project(hello_library)
4
5  #####
6  # Create a library
7  #####
8
9  #Generate the static library from the library sources
10 | add_library(${PROJECT_NAME} STATIC
11     src/Hello.cpp
12 )
13
14 | target_include_directories(${PROJECT_NAME}
15     PUBLIC
16     ${PROJECT_SOURCE_DIR}/include
17 )
18
19
20 #####
21 # Create an executable
22 #####
23
24 # Add an executable with the above sources
25 | add_executable(hello_binary
26     src/main.cpp
27 )
28
29 # link the new hello_library target with the hello_binary target
30 | target_link_libraries( hello_binary
31     PRIVATE
32     ${PROJECT_NAME}
33 )
34

```

برای ساخت کتابخانه از تابع `add_library` استفاده می‌شود که ورودی اول نام کتابخانه و ورودی بعدی نوع کتابخانه (`static` یا `shared`) و ورودی‌های بعدی نام فایل‌های کتابخانه است و برای لینک کردن کتابخانه به فایل باینری از تابع `target_link_libraries` استفاده می‌شود همانطور که در تصویر بالا می‌بینید و از تابع `target_include_directories` برای افزودن دایرکتوری‌های `include` استفاده می‌شود که پوشه‌های مشخص شده را به فایل باینری که در ورودی اول تابع گرفته شده اضافه می‌شود.

ساخت یک کتابخانه داینامیک و افزودن آن به پروژه:


```

1  cmake_minimum_required(VERSION 3.5)
2
3  project(hello_library)
4
5  #####
6  # Create a library
7  #####
8
9  #Generate the shared library from the library sources
10 add_library(hello_library SHARED
11             src/Hello.cpp
12            )
13 add_library(hello::library ALIAS hello_library)
14
15 target_include_directories(hello_library
16                             PUBLIC
17                             ${PROJECT_SOURCE_DIR}/include
18                            )
19
20 #####
21 # Create an executable
22 #####
23
24 # Add an executable with the above sources
25 add_executable(hello_binary
26                src/main.cpp
27               )
28
29 # link the new hello_library target with the hello_binary target
30 target_link_libraries( hello_binary
31                       PRIVATE
32                       hello::library
33                      )
34

```

برای ساخت کتابخانه داینامیک از واژه **shared** بجای **static** استفاده می کنیم.

نکته :

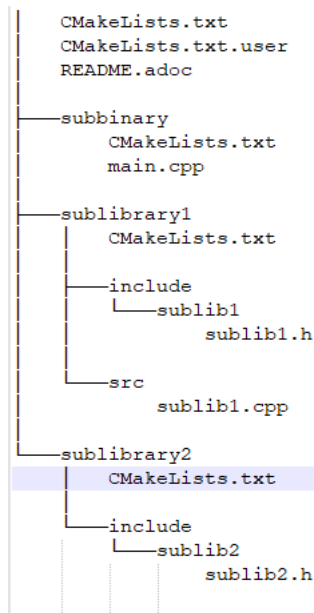
add_library(hello::library ALIAS hello_library)

برای خوانایی بیشتر نوشته شده و میتوان از همان `hello_library` استفاده کرد.

افزودن زیر پروژه:

برای افزودن زیر پروژه از تابع `add_subdirectory` استفاده می کنیم که ورودی نام پوشه مورد نظر را به عنوان ورودی می گیرد و نیاز است در پوشه ای که می خواهیم اضافه کنیم شامل یک `CMakeLists.txt` باشد.

ساختار `cmake` به صورت درختیست به این معنی که اگر متغیری در `CMakeLists.txt` پوشه بالاتر تعریف شده باشد در `CMakeLists.txt` پوشه پایین قابل استفاده است. ساختار پوشه و فایل های زیر مفروض است:

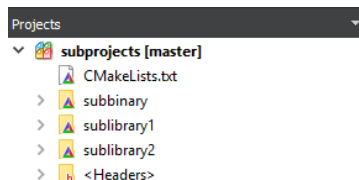


برای افزودن زیر پروژه به پروژه اصلی برای چنین ساختاری کد زیر را در CMakeLists.txt پوشه اصلی می‌نویسیم:

```

1 cmake_minimum_required (VERSION 3.5)
2
3 project(subprojects)
4
5 # Add sub directories
6 add_subdirectory(sublibrary1)
7 add_subdirectory(sublibrary2)
8 add_subdirectory(subbinary)
9
  
```

و هر پوشه CMakeLists.txt خود را خواهد داشت.



نوشتن Find برای کتابخانه:

وقتی کتابخانه‌ای می‌نویسیم برای این که فایل‌های کتابخانه به راحتی برای کاربر قابل اضافه کردن باشند نیاز است یک فایل با الگوی اسم زیر ایجاد کنیم (معمولا در آدرس cmake/ Modules قرار میگیرد):

FindLibraryName.cmake

در مثال بالا وقتی آدرس این فایل به متغیر CMAKE_MODULE_PATH اضافه شود بعد از استفاده از `find_package` ابزار cmake به دنبال فایل با الگوی اسم بالا می‌گردد و بعد از پیدا کردن آن فایل نیز اجرا می‌شود و نیاز است در آن فایل ما آدرس‌ها مرتبط با کتابخانه خود را تنظیم کنیم. برای مثال وقتی یک کتابخانه به اسم MyLib داریم باید متغیرهای `MyLib_INCLUDE_DIR` و `MyLib_LIBRARY` را تنظیم کنیم البته بهتر است اگر چندین پوشه برای `include` داریم از `MyLib_INCLUDE_DIRS` و اگر چندین فایل کتابخانه برای اضافه کردن داریم از `MyLib_LIBRARIES` استفاده کنیم.

نکته: برای افزودن یک مقدار به یک لیست در CMake از تابع `list` به صورت زیر استفاده می‌شود:

```
list(APPEND <نام متغیر> "مقدار جدید")
```

اگر بخواهیم موجود بودن یک فایل را بررسی کنیم به صورت زیر عمل می‌کنیم:

```
if(EXISTS file.txt)
```

```
endif()
```

اگر بخواهیم بررسی کنیم که یک زیر رشته در رشته اصلی وجود دارد یا خیر به صورت زیر عمل می‌کنیم :

```
if(${file} MATCHES .tar)
```

```
endif()
```

برای منفی کردن شرط باید یک `NOT` به اول شرط افزود.

برای کپی فایل از تابع فایل با الگوی زیر استفاده می‌کنیم:

```
file(COPY <آدرس پوشه مورد نظر> DESTINATION <فایل یا فایل‌های مورد نظر>)
```

برای حذف فایل:

```
file(REMOVE <فایل مورد نظر>)
```

و همینطور برای نوشتن در یک فایل:

```
file(WRITE <محتوای مورد نظر> <آدرس فایل مورد نظر>)
```

افزودن نصب کننده به پروژه:

ابزار CMake برای نصب سیستم خودکاری فراهم کرده که می‌توانیم با استفاده از تابع `install` از آن به صورت زیر استفاده کنیم.

```
install(TARGETS ${PROJECT_NAME} DESTINATION ${CMAKE_INSTALL_PREFIX}/bin)
```

```
install(FILES ${Headers} DESTINATION ${CMAKE_INSTALL_PREFIX}/include/${PROJECT_NAME})
```

در مثال بالا دستور اول موجب کپی فایل اصلی پروژه (فایل باینری) در پوشه `bin` در داخل پوشه نصب استاندارد می‌شود دقت کنید که همیشه باید آدرس نصب بر اساس متغیر `CMAKE_INSTALL_PREFIX` باشد چون این آدرس به صورت پیشفرض آدرس نصب است و در صورتی که کاربر بخواهد بدون تغییر در کد آدرس را تغییر دهد می‌تواند با تغییر متغیر آدرس این کار را انجام دهد.

دقت کنید که نیاز است اگر این متغیر خالی بود مقدار دهی شود چون ممکن است دیگر کتابخانه‌های موجود در پروژه از این آدرس استفاده کرده باشند.

دستور دوم نیز فایل‌های هدر را که در متغیر `Headers` ذخیره کردیم در آدرس `include` و آدرس اسم پروژه کپی می‌کند که البته این مورد برای کتابخانه‌ها به کار می‌رود.

افزودن حذف نصب به پروژه:

به صورت پیشفرض CMake قابلیت حذف نصب را فراهم نکرده ولی برای حذف نصب کفایت فایل‌های نصب شده را پاک کنیم که لیست تمامی آن‌ها در فایل `install_manifest.txt` در پوشه نصب پروژه نگهداری می‌شود. برای این کار کفایت در بالاترین پوشه‌ی پروژه که شامل `CMakeLists.txt` است یک فایل با نام `cmake_uninstall.cmake.in` بسازیم و درون آن کد زیر را بنویسیم:

```
if(NOT EXISTS "@CMAKE_BINARY_DIR@/install_manifest.txt")

  message(FATAL_ERROR "Cannot find install manifest: @CMAKE_BINARY_DIR@/install_manifest.txt")

endif(NOT EXISTS "@CMAKE_BINARY_DIR@/install_manifest.txt")

file(READ "@CMAKE_BINARY_DIR@/install_manifest.txt" files)
string(REGEX REPLACE "\n" ";" files "${files}")
foreach(file ${files})
  message(STATUS "Uninstalling $ENV{DESTDIR}${file}")
  if(IS_SYMLINK "$ENV{DESTDIR}${file}" OR EXISTS "$ENV{DESTDIR}${file}")
    exec_program(
      "@CMAKE_COMMAND@" ARGS "-E remove \"$ENV{DESTDIR}${file}\""
      OUTPUT_VARIABLE rm_out
      RETURN_VALUE rm_retval
    )
    if(NOT "${rm_retval}" STREQUAL 0)
      message(FATAL_ERROR "Problem when removing $ENV{DESTDIR}${file}")
    endif(NOT "${rm_retval}" STREQUAL 0)
  else(IS_SYMLINK "$ENV{DESTDIR}${file}" OR EXISTS "$ENV{DESTDIR}${file}")
    message(STATUS "File $ENV{DESTDIR}${file} does not exist.")
  endif(IS_SYMLINK "$ENV{DESTDIR}${file}" OR EXISTS "$ENV{DESTDIR}${file}")
endforeach(file)
```

و در `CMakeLists.txt` نیز کد زیر را اضافه کنیم:

```
# uninstall target
if(NOT TARGET uninstall)
  configure_file(
    "${CMAKE_CURRENT_SOURCE_DIR}/cmake_uninstall.cmake.in"
    "${CMAKE_CURRENT_BINARY_DIR}/cmake_uninstall.cmake"
    IMMEDIATE @ONLY)
endif()
```

```
add_custom_target(uninstall
  COMMAND ${CMAKE_COMMAND} -P ${CMAKE_CURRENT_BINARY_DIR}/cmake_uninstall.cmake)
endif()
```

معرفی چند متغیر پر کاربرد:

CMAKE_CURRENT_SOURCE_DIR: آدرس پوشه سورس فعلی را در خود ذخیره کرده است.

CMAKE_SYSTEM_NAME: نام سیستم عامل میزبان را در خود ذخیره کرده است برای مثال در ویندوز مقدار این متغیر برابر با Windows است.

CMAKE_SYSTEM: نام سیستم عامل بعلاوه اطلاعات نسخه آن را در خود ذخیره کرده است برای مثال Windows-10.0.17134

CMAKE_BUILD_TYPE: نوع build پروژه را در خود نگه می‌دارد (Debug.Release)

Build یک پروژه ساده شامل Qt:

برای یک پروژه ساده ویجت که شامل یک **MainWindow** ساده به همراه **ui** **CMakeLists.txt** به صورت زیر خواهد بود:

```
cmake_minimum_required(VERSION 2.6)
project(testCmake CXX C)
enable_testing()

set(CMAKE_INCLUDE_CURRENT_DIR ON)

set(CMAKE_AUTOMOC ON)
set(CMAKE_AUTOUIC ON)
set(CMAKE_AUTORCC ON)

find_package(Qt5 COMPONENTS Core Widgets REQUIRED)
add_executable(${PROJECT_NAME} "main.cpp" MainWindow.h MainWindow.cpp MainWindow.ui)

include_directories(
  ${Qt5Core_INCLUDE_DIRS}
  ${Qt5Widgets_INCLUDE_DIRS}
)

target_link_libraries(${PROJECT_NAME})
```

Qt5::Core

Qt5::Widgets

)

دقت کنید که مقادیر متغیر های (CMAKE_AUTORCC و CMAKE_AUTOUI و CMAKE_AUTOMOC) باید ON قرار داده شود و به این نکته نیز توجه داشته باشید که فایل های ui نیز مانند فایل های cpp باید به فایل اجرایی افزوده شوند تا وقتی پروژه را در کیوت باز می کنیم بتوانیم از تعریف خودکار اسلات از ui در کد استفاده کنیم.

تعریف تابع در CMake:

برای تعریف تابع در CMake نیاز است یک فایل با پسوند .cmake در پوشه Modules اضافه کرده و پوشه Modules را به CMAKE_MODULE_PATH اضافه کنیم.

```
list(INSERT CMAKE_MODULE_PATH 0 ${CMAKE_SOURCE_DIR}/cmake/Modules)
```

سپس فایل مورد نظر را با اسم آن include کنیم.

فرض کنید یک فایل با نام fonqri.cmake در پوشه Modules ساختیم و پوشه را با دستور بالا به CMAKE_MODULE_PATH افزودیم پس از آن با دستور زیر fonqri را include می کنیم.

```
include(fonqri)
```

حال در فایل fonqri.cmake با تابع function به صورت زیر به تعریف تابع می پردازیم:

```
function(FONQRI_PRINT_POWER)
```

```
    message(STATUS "Max")
```

```
endfunction(FONQRI_PRINT_POWER)
```

در مثال بالا یک تابع ساده به نام FONQRI_PRINT_POWER تعریف کردیم که در آن یک مقدار به عنوان message چاپ می شود.

در صورتی که نیاز داشته باشید تابع آرگومان ورودی دریافت کند کفایت بعد از نام تابع به عنوان ورودی های بعدی تابع function ورودی های خود را بنویسید و از آنها استفاده کنید و در زمان فراخوانی تابع کاربر باید ورودی را به تابع بفرستد.