Tutorial CMAKE

1 Objetivo

Esse é um breve tutorial para o CMake para ser aplicado no desenvolvimento do trabalho de Engenharia de Software intitulado Sistema de Provas.

2 CMake

2.1 Instalação

Para utilizar o CMake, será necessário instalá-lo. Ele está disponível tanto no site https://cmake.org/ e no Github oficial https://github.com/Kitware/CMake . No Github, basta ir em Releases e baixar o zip/msi ou tar.gz da última release disponível (cuidado para não baixar só os source files). Depois, disso, basta seguir o procedimento normal de instalação de programas.

Para verificar que o CMake foi instalado, basta digitar “cmake –version” no terminal e ver se funciona. Se não funcionar, pode ser que você esqueceu de adicionar a pasta bin do CMAKE ao PATH.

2.2 O que faz

O CMake é um gerador de Makefiles facilitando a sua criação a partir de um arquivo de configuração específico que o CMake utiliza para funcionar. Desse modo, depois de rodar o CMake, ainda precisaremos executar “make” no terminal para compilar o projeto.

2.3 Arquivo de configuração CMake

O arquivo que o CMake utiliza para funcionar se chama **CMakeLists.txt**. No nosso caso, ele ficará na pasta raiz do projeto. Dentro dele, estarão as configurações para gerar um Makefile apropriado para o projeto. Apesar de ser perfeitamente possível ter mais de um CMakeLists ao longo projeto, eles são necessários só se quisermos dividir o projeto em subprojetos e, possivelmente, bibliotecas. Nesse projeto, até agora, teremos apenas esse na pasta raiz.

Existem alguns comandos que ficarão dentro desse arquivo. O primeiro deles é o que define a versão mínima do CMake necessária para executar esse arquivo de configuração:

cmake_minimum_required(VERSION 3.0)

O VERSION é como se fosse uma variável base do Cmake e o 3.0 indica a versão necessária. Atualmente, no Github, a última versão é a 3.20.

O segundo comando é o que indica o nome do projeto:

project(sistema_provas)

Dessa forma, podemos nos referenciar ao nosso projeto por *sistema_provas* ao longo do arquivo de configuração.

Um terceiro comando útil é o que define a versão mínima do C++ utilizada ao longo do projeto:

set(CMAKE_CXX_STANDARD 11)

O CMAKE_CXX_STANDARD é uma outra variável que será utilizada pelo CMake que acabará virando a flag *-std=c++11* ao chamar o *g++*.

Agora, um dos principais comandos é o que indica os arquivos fonte (.cpp) que serão necessários para gerar o executável do projeto:

add_executable(sistema_provas src/main.cpp src/business/application.cpp)

Pelo próprio nome do comando podemos supor que ele adiciona um executável ao projeto alvo (target) a partir de certos arquivos fonte.

Primeiro, indicamos o nome do executável, no caso, sistema_provas. Esse nome não se refere ao sistema_provas do nome do projeto, é apenas um padrão dar o nome do executável como igual ao nome do projeto.

Em seguida indicamos **todos** os arquivos fonte necessários ao executável. Nesse caso, podemos perceber que a main está dentro da pasta src e o application dentro de src/business. Desse modo, **cada vez que criarmos um novo arquivo .cpp, devemos adicioná-lo nessa lista de arquivos fonte**.

Um outro comando principal é o comando que inclui os .hpp ao executável.

target_include_directories(sistema_provas PUBLIC
${PROJECT_SOURCE_DIR}/include/)

Nesse comando, indicamos o nome do alvo que precisará de arquivos de include. No caso, esse target se refere ao sistema_provas do *add_executable*. O PUBLIC poderia ser PRIVATE também, mas sua explicação pode ser vista em https://github.com/ttroy50/cmake-examples/tree/master/01-basic/C-static-library.

O *${PROJECT_SOURCE_DIR}/include/* indica a pasta onde estarão todos os arquivos .hpp necessários pelo alvo. Dessa forma, **já temos como base ao longo do código a pasta include/**, portanto, se **main.cpp** quisesse importar **include/business/application.hpp**, precisaríamos apenas do comando **#include "business/application.hpp"**. Na verdade, se colocarmos #include "include/business/application.hpp" ou #include "application.hpp", daria um erro de compilação.

Esse comando, se seguirmos a mesma estrutura de pastas de raiz/src, raiz/include e raiz/build ao longo do projeto, não precisará ser alterado.

2.4 Como utilizar o CMake

O CMake tem uma versão GUI quando o instalamos, mas vou explicar apenas seu uso no terminal. O comando para usar o CMake pode ser tão simples como apenas:

```
cmake .
```

Desse modo, estamos dizendo que queremos que o cmake execute procurando o arquivo CMakeLists.txt na mesma pasta que estamos executando o comando. Porém, como é recomendado, devemos executar o comando a partir da pasta raiz/build, porque o cmake gera muitos arquivos que podem poluir a pasta raiz do projeto.

Com o arquivo CMakeLists.txt na pasta raiz do projeto e o terminal na pasta raiz/build, poderíamos executar o CMake com o seguinte comando:

```
cmake ..
```

Dessa forma, o CMake vai procurar seu arquivo de configuração na pasta anterior a que estamos executando o comando, mas irá gerar seus arquivos na pasta atual. Esse deverá ser o padrão no nosso projeto.

Como eu uso Windows, se eu executar apenas o comando cmake .., por padrão, o CMake não criará um arquivo Makefile, ele tentará criar um outro arquivo em outro formato. Para criar um arquivo Makefile, eu tive que adicionar o seguinte comando:

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

Esse comando indica que eu quero usar um gerador (Generator) que seja com base nos Makefiles do MinGW.

(https://stackoverflow.com/questions/39643291/make-without-makefile-after-cmake)

Para isso, obviamente você deverá ter o MinGW instalado e, para verificar os tipos de Generators disponíveis no seu computador, basta executar o seguinte comando:

cmake –help

Que uma lista de Generators disponíveis será mostrada.

Caso isso tudo funcione, um monte de arquivos e pastas serão gerados na pasta build junto com um Makefile. Após isso, podemos executar o comando make no terminal para criar o executável dentro da pasta build.

Para executar o *make*, verifique se a pasta, provavelmente, ProgramFiles/MinGW/bin contém um executável chamado make ou mingw32-make. Caso seja apenas mingw32-make, você deverá digitar mingw32-make no terminal para executar o make na pasta build. Uma alternativa, é simplesmente renomear esse executável para “make.exe” para poder utilizar somente “make” no terminal.