

Compilação Multiplataforma com CMake

Autor(es): Rafael Alvarenga de Azevedo
Mário Luiz Rodrigues Oliveira



*Inscrições a partir de
16/09 até 09/10*

XI JECT | Jornada de Educação
Ciência e Tecnologia
20 e 21/10/2022

MRA SEBRAE H celula SICOOB Instituto Federal

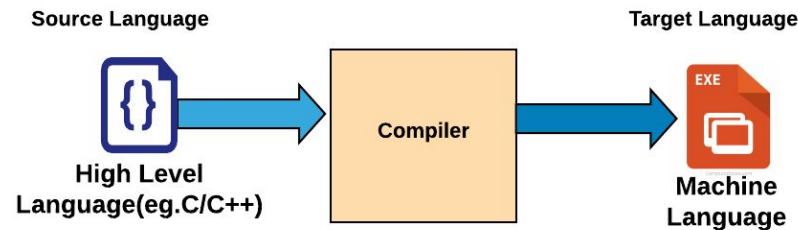
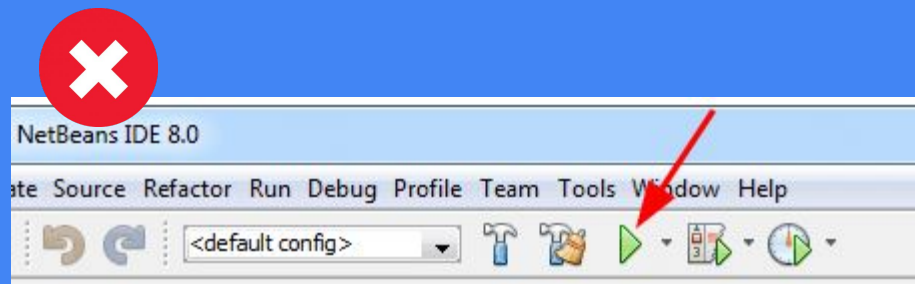
Conteúdo do Minicurso

- ❑ Conceitos básicos
- ❑ Motivações
- ❑ Processo de Compilação
- ❑ Compilação Multiplataforma
- ❑ Ferramenta CMake
- ❑ Prática

O que é um
compilador ?

O que é um compilador ?

Não é o botão PLAY da sua IDE...



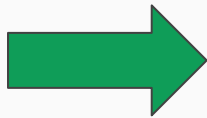
Native Compiler vs Cross Compiler

Compilador **Nativo**

```
$ g++ -o programa.bin main.cpp
```



main.cpp



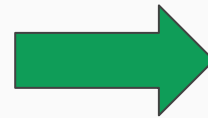
programa.bin

Compilador **Multiplataforma**

```
$ x86_64-w64-mingw32-g++ -o programa.exe main.cpp
```



main.cpp



programa.exe

Caminho Relativo *Caminho Absoluto*

Relativo

Forma de acessar um diretório vizinho através do diretório atual:

`../pasta_vizinha`

Absoluto

Forma de acessar um diretório vizinho diretamente:

`/home/User/pasta_vizinha`

Organização de Diretórios

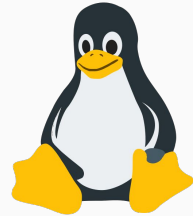
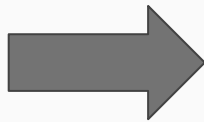
In-source

```
rafa12dev@rafadesk ~/Documents/Lab_C++/PROJETO lsd --tree
├── .
└── SOURCE
    ├── BUILD
    │   └── CMakeCache.txt
    ├── CMakeLists.txt
    ├── lib1.cpp
    ├── lib1.hpp
    ├── lib2.cpp
    ├── lib2.hpp
    └── main.cpp
```

Out-of-source

```
rafa12dev@rafadesk ~/Documents/Lab_C++/PROJETO lsd --tree
├── .
├── BUILD
│   └── CMakeCache.txt
└── SOURCE
    ├── CMakeLists.txt
    ├── lib1.cpp
    ├── lib1.hpp
    ├── lib2.cpp
    ├── lib2.hpp
    └── main.cpp
```

Motivações - Portabilidade



macOS

Motivações - Compilação Condicional

```
1  #include <iostream>
2
3  using std::cout;
4  using std::endl;
5
6  #ifdef _WIN32
7      #define OS_NAME "Windows"
8      #define OS_SLASH_TYPE "\\"
9  #else
10     #define OS_NAME "Linux"
11     #define OS_SLASH_TYPE "/"
12 #endif
13
14 int main(int argc, char** argv) {
15     cout << "Sistema Operacional atual: " << OS_NAME << endl;
16     cout << "Barra usada para nomear diretórios: " << OS_SLASH_TYPE << endl;
17     return 0;
18 }
19
```

Fonte: o próprio autor

Motivações - Manutenção de Makefile

```
254 lines (220 sloc) | 6.43 KB
Raw Blame

1 #####
2 #### Variables and settings
3 #####
4
5 # Executable name
6 EXEC = program
7
8 # Build, bin, assets, and install directories (bin and build root directories are kept for clean)
9 BUILD_DIR_ROOT = build
10 BIN_DIR_ROOT = bin
11 ASSETS_DIR = assets
12 ASSETS_OS_DIR := $(ASSETS_DIR)_os
13 INSTALL_DIR := ~/Desktop/$(EXEC)
14
15 # Sources (searches recursively inside the source directory)
16 SRC_DIR = src
17 SRCS := $(sort $(shell find $(SRC_DIR) -name '*.cpp'))
18
19 # Includes
20 INCLUDE_DIR = include
21 INCLUDES := -I$(INCLUDE_DIR)
22
23 # C preprocessor settings
24 CPPFLAGS = $(INCLUDES) -MMD -MP
25
26 # C++ compiler settings
27 CXX = g++
28 CXXFLAGS = -std=c++17
29 WARNINGS = -Wall -Wpedantic -Wextra
30
31 # Linker flags
32 .....
```

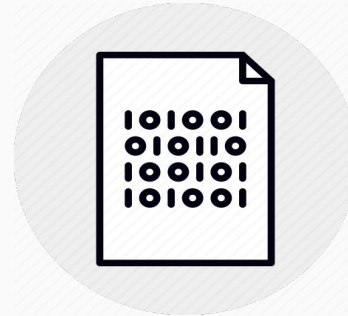
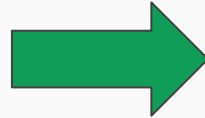
Fonte: <https://github.com/KRMisha/Makefile>

Processo de Compilação

```
$ g++ -o programa.bin main.cpp
```

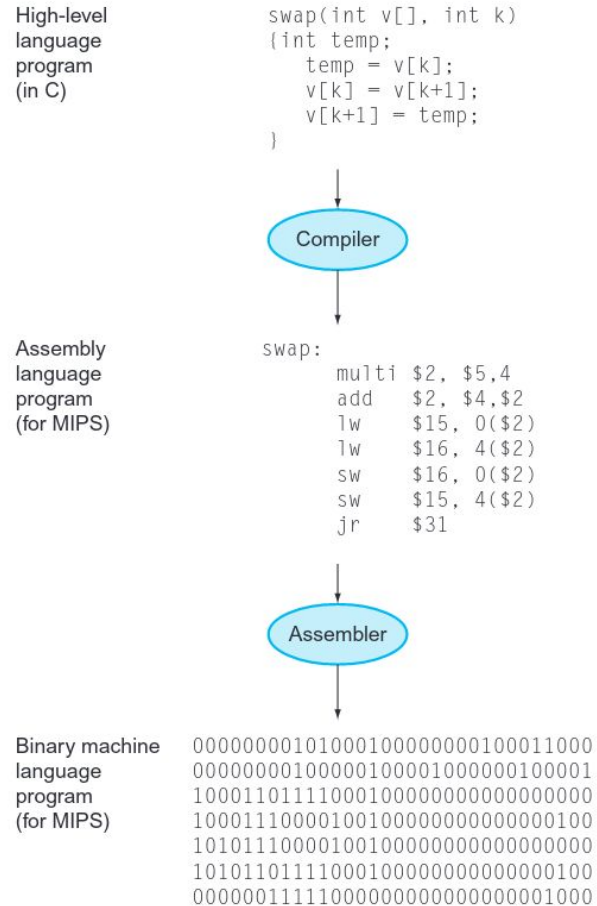


main.cpp



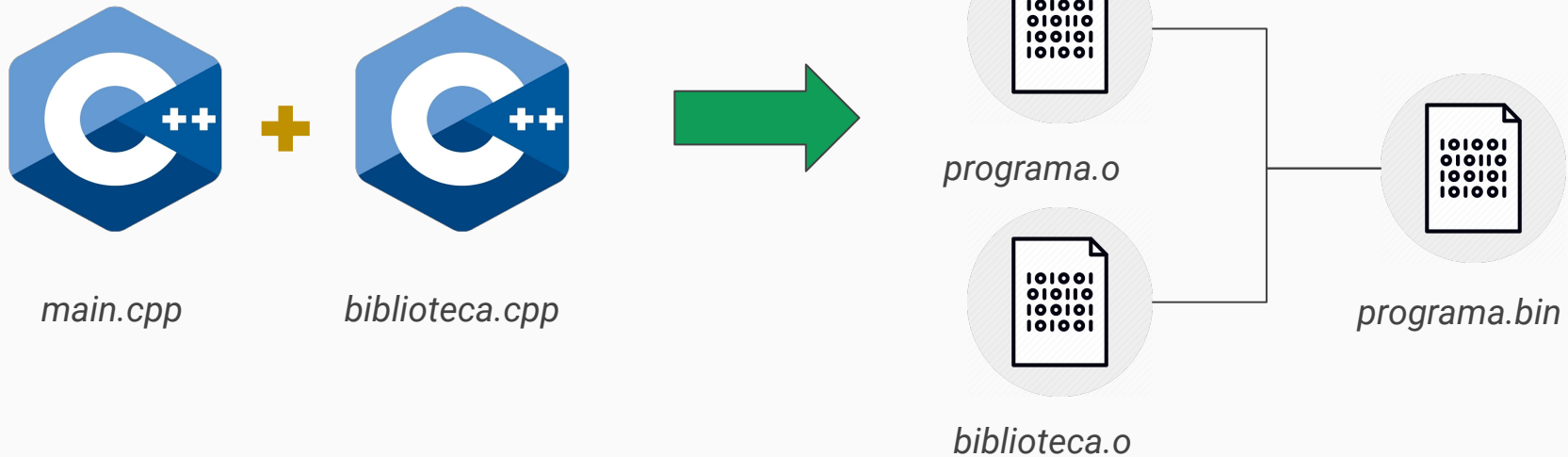
programa.bin

Processo de Compilação



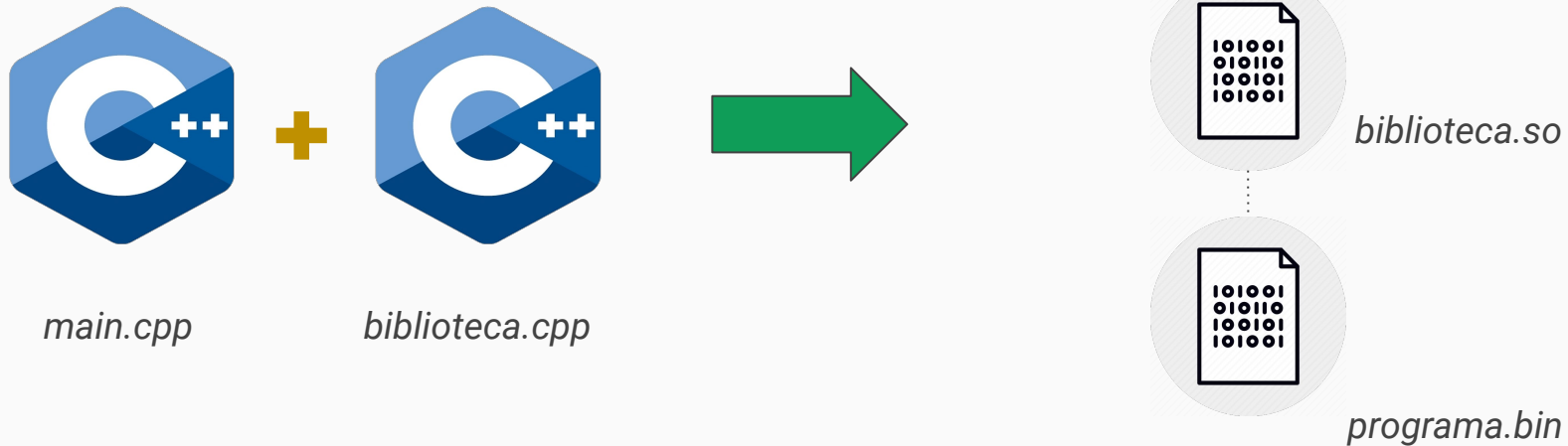
Linkando bibliotecas

Linkagem **estática**



Linkando bibliotecas

Linkagem *dinâmica*



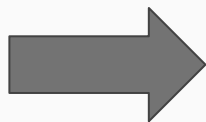
Processo de Compilação

`$ make`

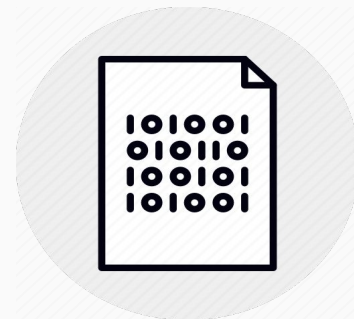
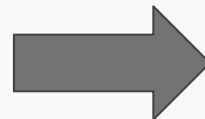
`$ g++ -o programa.bin main.cpp`



Makefile

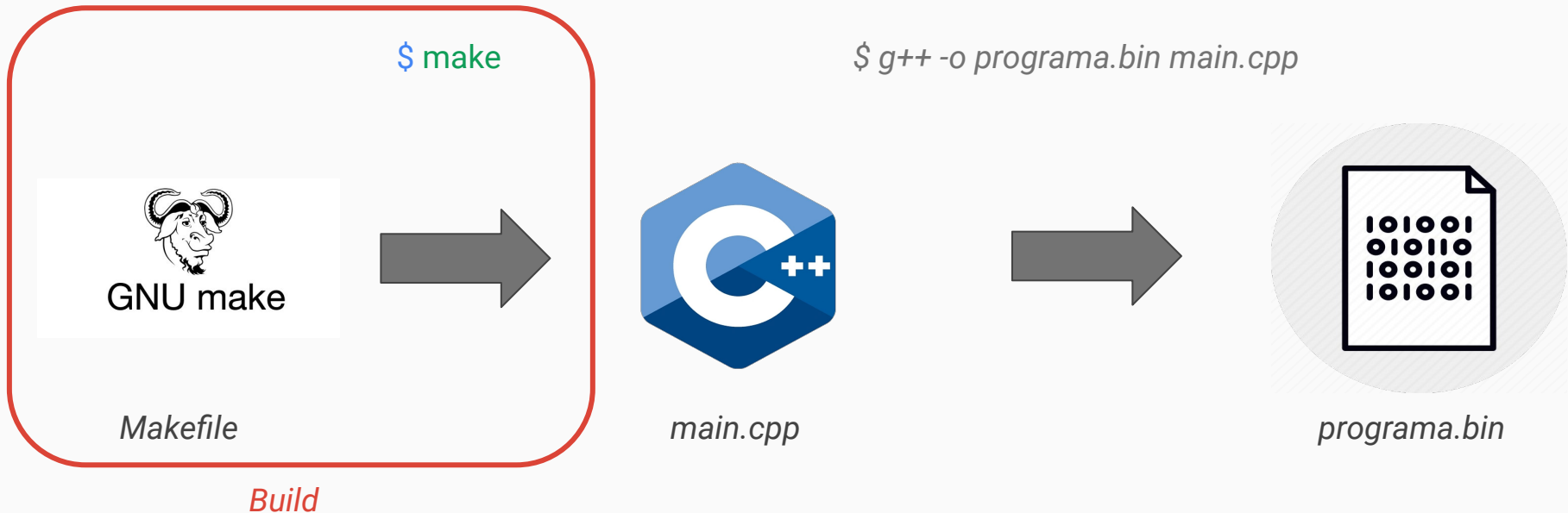


main.cpp



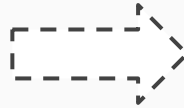
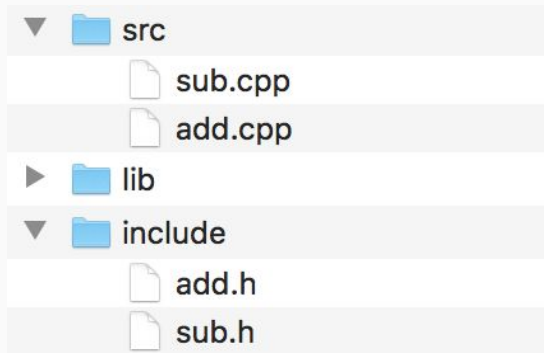
programa.bin

Processo de Compilação



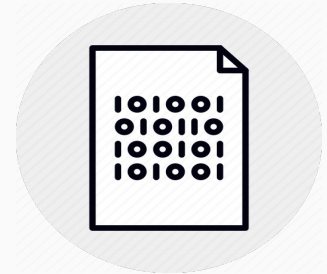
Fonte: Google Imagens

Processo de Compilação - *Build*



Makefile

`$ make`

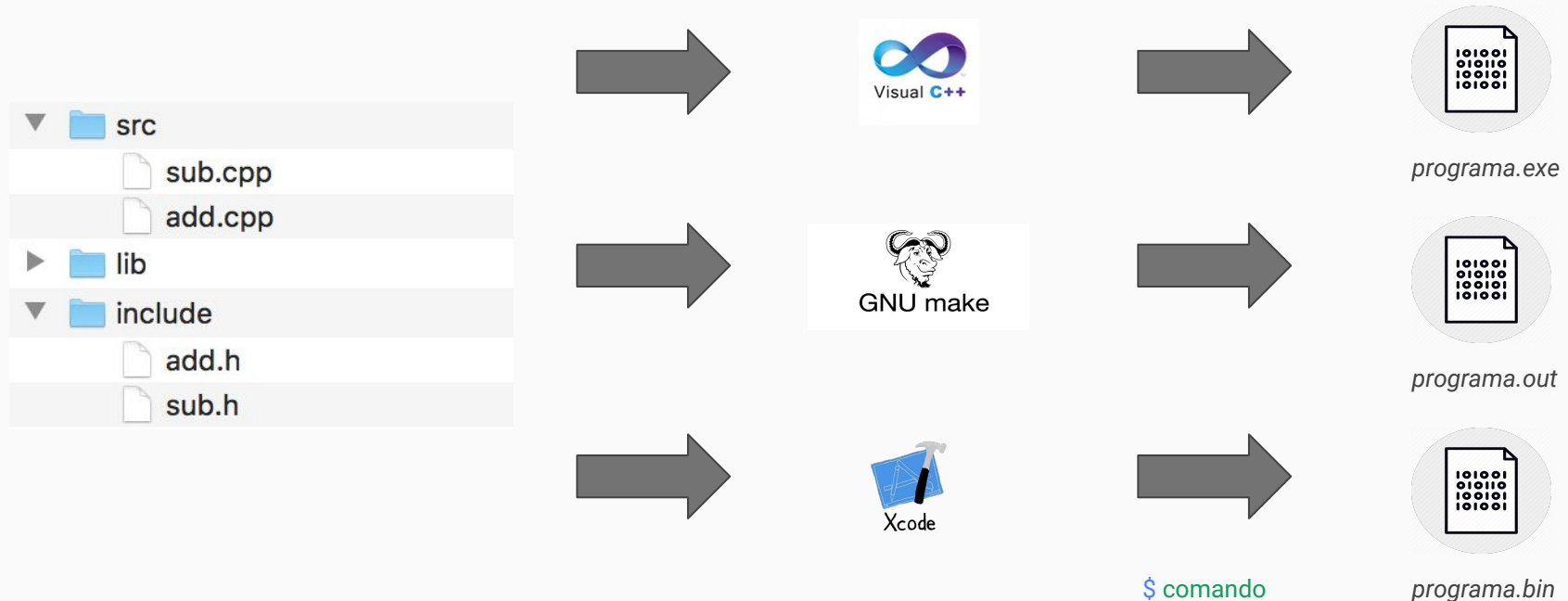


programa.bin

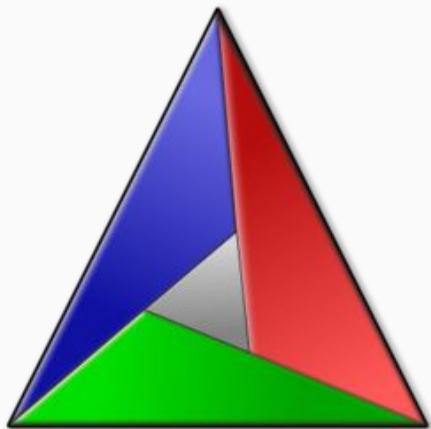
Tipos de *Build*

1. **Debug**: compila seu(ua) executável/biblioteca SEM otimizações e COM símbolos de debug;
2. **Release**: compila seu(ua) executável/biblioteca COM otimizações e SEM símbolos de debug;
3. **RelWithDebInfo**: compila seu(ua) executável/biblioteca COM otimizações menos agressivas e COM símbolos de debug;
4. **MinSizeRel**: compila seu(ua) executável/biblioteca COM otimizações que não aumentam o tamanho do código gerado.

Compilação Multiplataforma



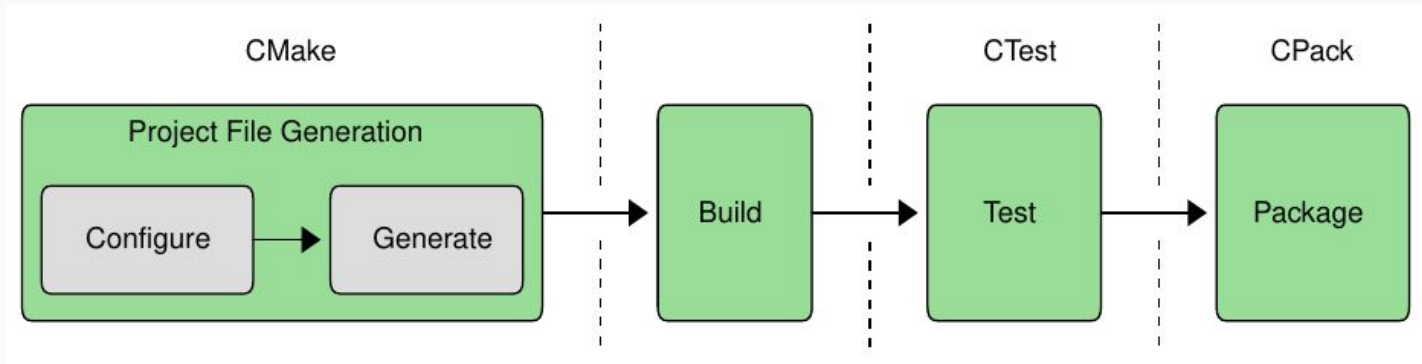
Ferramenta CMake



Fonte: Kitware (2022)

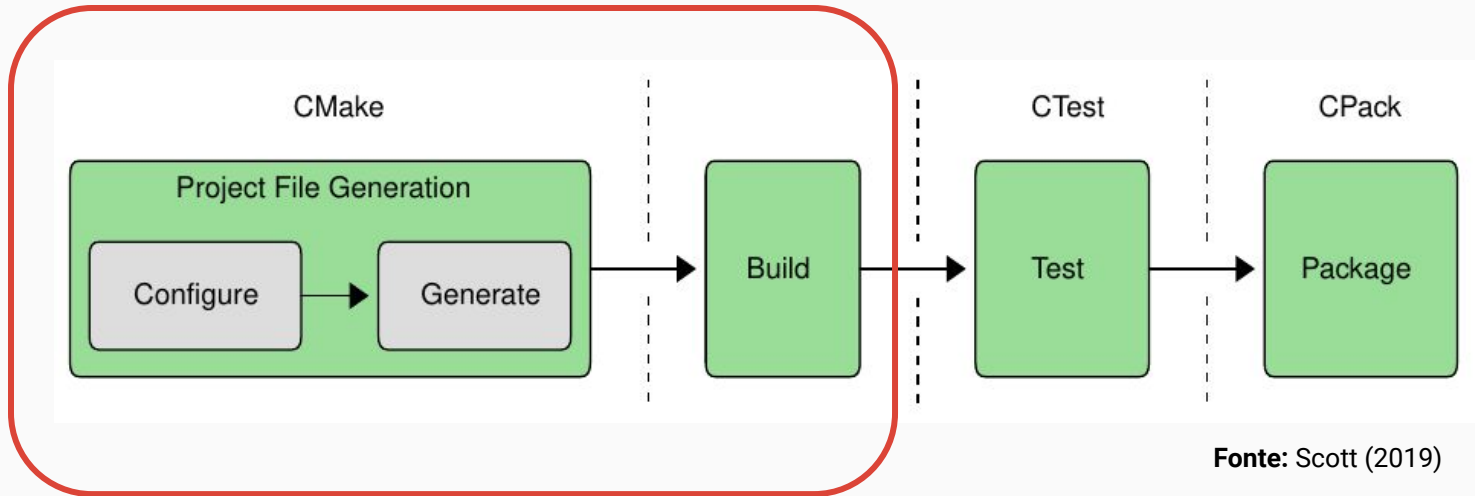
- ❑ Ferramenta de controle do processo de compilação
- ❑ Geração de arquivos de *build* nativos
- ❑ Possui mecanismos para testes automatizados (CTest)
- ❑ Possui mecanismos de empacotamento para distribuição de software (CPack)

Processo de compilação com CMake



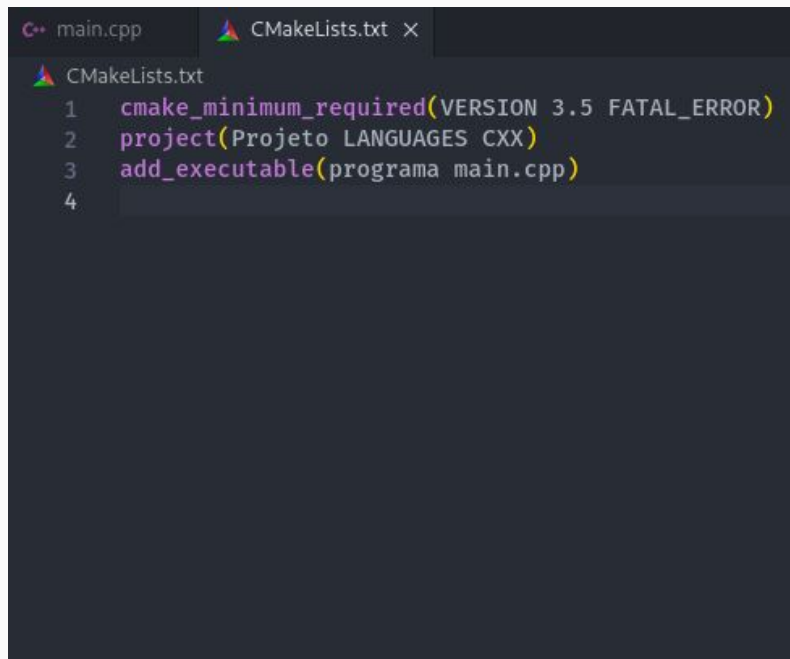
Fonte: Scott (2019)

Processo de compilação com CMake



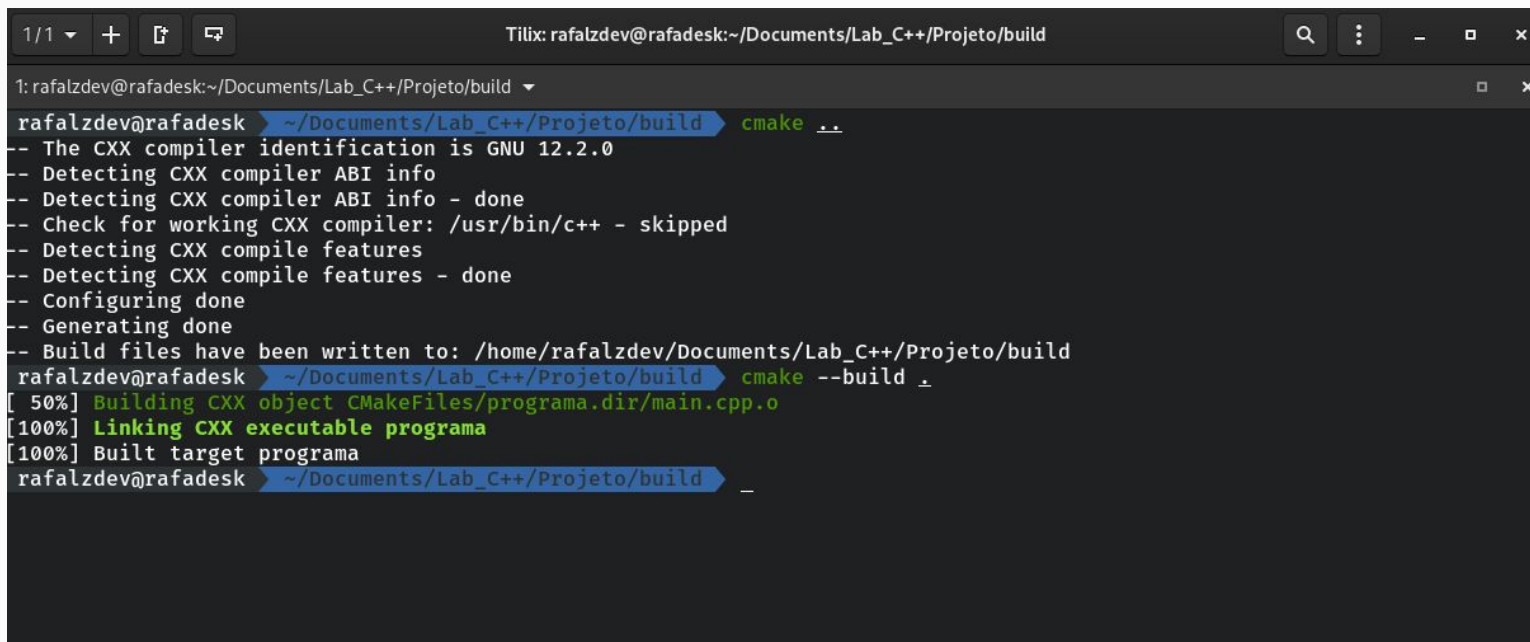
Arquivo *CMakeLists.txt*

Para construir um projeto com CMake, é necessário configurar um projeto em um arquivo *CMakeLists.txt*.

A screenshot of a code editor with a dark theme. The top bar shows two tabs: 'main.cpp' with a C++ icon and 'CMakeLists.txt' with a CMake icon. The 'CMakeLists.txt' tab is active. The code in the editor is as follows:

```
CMakeLists.txt
1 cmake_minimum_required(VERSION 3.5 FATAL_ERROR)
2 project(Projeto LANGUAGES CXX)
3 add_executable(programa main.cpp)
4
```

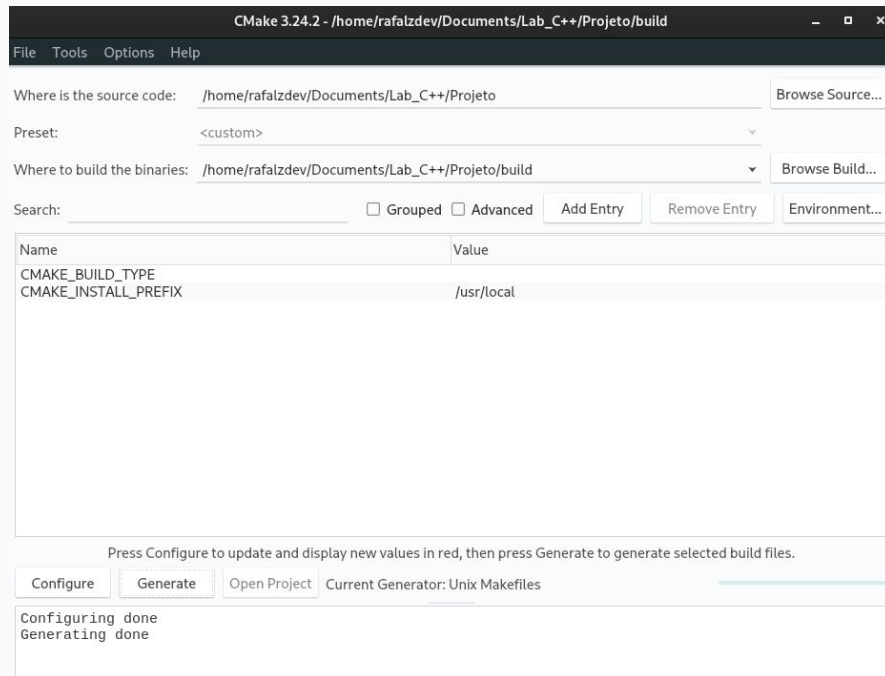
Execução do CMake - CLI



The image shows a terminal window titled "Tilix: rafalzdev@rafadesk:~/Documents/Lab_C++/Projeto/build". The terminal displays the output of the CMake CLI commands. The first command is `cmake ..`, which configures the build system. The output shows that the CXX compiler is identified as GNU 12.2.0, and the build files are generated in the current directory. The second command is `cmake --build .`, which builds the project. The output shows the progress of building the CXX object `CMakeFiles/programa.dir/main.cpp.o` and linking the executable `programa`.

```
1/1 ▾ + ↵ ↻
Tilix: rafalzdev@rafadesk:~/Documents/Lab_C++/Projeto/build
1: rafalzdev@rafadesk:~/Documents/Lab_C++/Projeto/build ▾
rafalzdev@rafadesk > ~/Documents/Lab_C++/Projeto/build cmake ..
-- The CXX compiler identification is GNU 12.2.0
-- Detecting CXX compiler ABI info
-- Detecting CXX compiler ABI info - done
-- Check for working CXX compiler: /usr/bin/c++ - skipped
-- Detecting CXX compile features
-- Detecting CXX compile features - done
-- Configuring done
-- Generating done
-- Build files have been written to: /home/rafalzdev/Documents/Lab_C++/Projeto/build
rafalzdev@rafadesk > ~/Documents/Lab_C++/Projeto/build cmake --build .
[ 50%] Building CXX object CMakeFiles/programa.dir/main.cpp.o
[100%] Linking CXX executable programa
[100%] Built target programa
rafalzdev@rafadesk > ~/Documents/Lab_C++/Projeto/build _
```


Execução do CMake - GUI

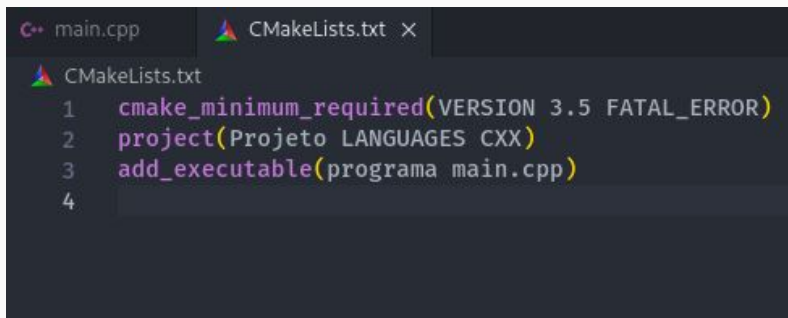


Comandos básicos

cmake_minimum_required

Require a minimum version of cmake.

```
cmake_minimum_required(VERSION <min>[...<policy_max>] [FATAL_ERROR])
```

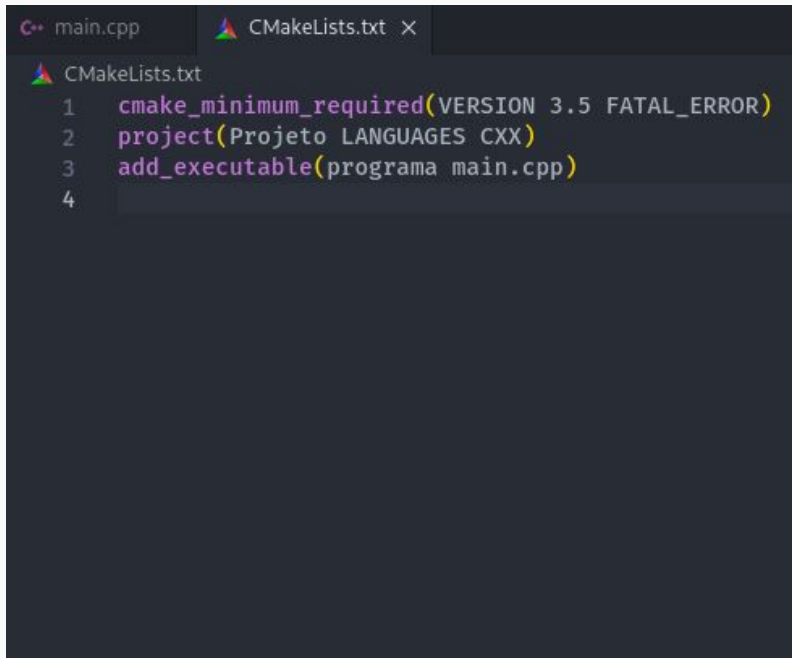


The screenshot shows a code editor with two tabs: 'main.cpp' and 'CMakeLists.txt'. The 'CMakeLists.txt' tab is active, displaying the following code:

```
CMakeLists.txt
1  cmake_minimum_required(VERSION 3.5 FATAL_ERROR)
2  project(Projeto LANGUAGES CXX)
3  add_executable(programa main.cpp)
4
```

Fonte: Kitware (2022)

Comandos básicos

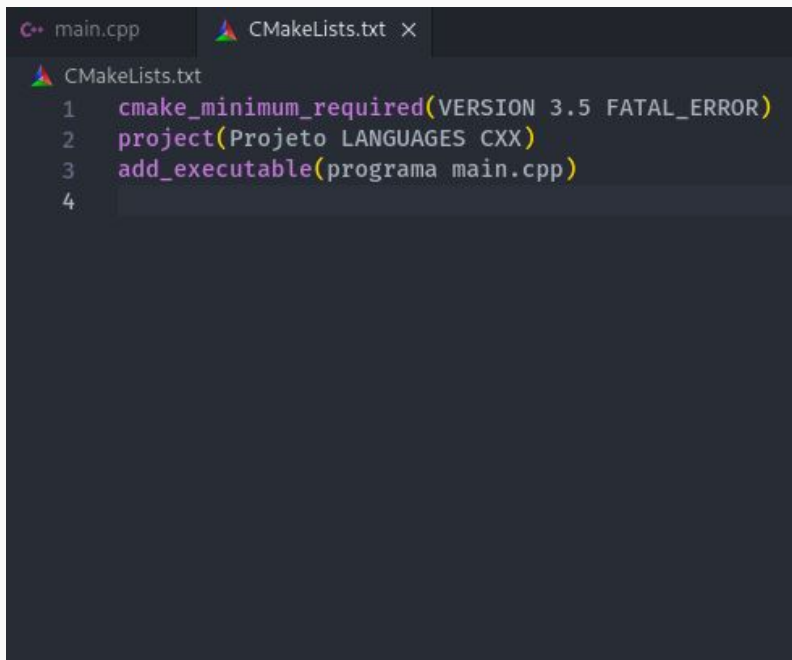


```
main.cpp  CMakeLists.txt x
CMakeLists.txt
1 cmake_minimum_required(VERSION 3.5 FATAL_ERROR)
2 project(Projeto LANGUAGES CXX)
3 add_executable(programa main.cpp)
4
```

```
project(projectName
    [VERSION major[.minor[.patch[.tweak]]]]
    [LANGUAGES languageName ...]
)
```

Fonte: Scott (2019)

Comandos básicos



A screenshot of a code editor with two tabs: 'main.cpp' and 'CMakeLists.txt'. The 'CMakeLists.txt' tab is active, showing the following code:

```
CMakeLists.txt
1 cmake_minimum_required(VERSION 3.5 FATAL_ERROR)
2 project(Projeto LANGUAGES CXX)
3 add_executable(programa main.cpp)
4
```

```
add_executable(targetName [WIN32] [MACOSX_BUNDLE]
                      [EXCLUDE_FROM_ALL]
                      source1 [source2 ...]
)
```

Fonte: Scott (2019)

Linkando bibliotecas

Passo 1: especificar dados da biblioteca

```
add_library(targetName [STATIC | SHARED | MODULE]
    [EXCLUDE_FROM_ALL]
    source1 [source2 ...]
)
```

Passo 2: linkar biblioteca no executável alvo

```
target_link_libraries(targetName
    <PRIVATE|PUBLIC|INTERFACE> item1 [item2 ...]
    [<PRIVATE|PUBLIC|INTERFACE> item3 [item4 ...]]
    ...
)
```

Criando variáveis

```
set(varName value... [PARENT_SCOPE])
```

Fonte: Scott (2019)

Usando condicionais

```
if(expression1)
    # commands ...
elseif(expression2)
    # commands ...
else()
    # commands ...
endif()
```

Opções para o usuário

Sintaxe:

Provide a boolean option that the user can optionally select.

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

Fonte: Kitware (2022)

Exemplo pela linha de comandos:

```
option(USE_LIBRARY "Compile sources into a library" OFF)
```

```
$ cmake -D USE_LIBRARY=ON ..
```

Fonte: Bast e Remigio (2018)

Exemplo pela GUI:

Search: ☐ Grouped ☐ Advanced

Name	Value
USE_LIBRARY	<input checked="" type="checkbox"/>

Especificando o compilador

Exemplo na linha de comandos:

```
$ cmake -D CMAKE_CXX_COMPILER=clang++ ..
```

Fonte: Bast e Remigio (2018)

Exemplo na GUI:

Search: _____ ☐ Grouped ☐ Advanced Add Entry Remove Entry Environment...

Name	Value
CMAKE_CXX_COMPILER	clang++

Mudando o tipo de *build*

Exemplo:

```
if(NOT CMAKE_BUILD_TYPE)
  set(CMAKE_BUILD_TYPE Release CACHE STRING "Build type" FORCE)
endif()

message(STATUS "Build type: ${CMAKE_BUILD_TYPE}")
```

```
$ cmake -D CMAKE_BUILD_TYPE=Debug ..
```

Fonte: Bast e Remigio (2018)

Controlando *flags* do compilador

Exemplo:

```
list(APPEND flags "-fPIC" "-Wall")
if(NOT WIN32)
  list(APPEND flags "-Wextra" "-Wpedantic")
endif()
```

```
target_compile_options(geometry
PRIVATE
  ${flags}
)
```

Fonte: Bast e Remigio (2018)

Sintaxe:

Add compile options to a target.

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

Fonte: Kitware (2022)

Escolhendo o padrão da linguagem

Dado um executável do seu projeto, é possível definir propriedades dele pelo comando `set_target_properties(...)`:

Sintaxe:

Targets can have properties that affect how they are built.

```
set_target_properties(target1 target2 ...  
    PROPERTIES prop1 value1  
    prop2 value2 ...)
```

Fonte: Kitware (2022)

Exemplo:

```
add_executable(animal-farm animal-farm.cpp)  
  
set_target_properties(animal-farm  
    PROPERTIES  
        CXX_STANDARD 14  
        CXX_EXTENSIONS OFF  
        CXX_STANDARD_REQUIRED ON  
    )
```

Fonte: Bast e Remigio (2018)

Usando loops

Loop por *foreach*:

```
foreach(loopVar arg1 arg2 ...)  
    # ...  
endforeach()
```

```
foreach(loopVar IN [LISTS listVar1 ...] [ITEMS item1 ...])  
    # ...  
endforeach()
```

Loop por *while*:

```
while(condition)  
    # ...  
endwhile()
```

MÃO NA
MASSA

Referências

1. BAST, Radovan; REMIGIO, Roberto Di. **CMake Cookbook**: Building, testing, and packaging modular software with modern CMake. [S. l.: s. n.], 2018. 606 p.
2. GEEKSFORGEES, Virusbuddha. **Difference between Native compiler and Cross compiler**. [S. l.], 3 maio 2020. Disponível em: <https://www.geeksforgeeks.org/difference-between-native-compiler-and-cross-compiler/>. Acesso em: 4 out. 2022.
3. KITWARE, CMake. **CMake**. [S. l.], 1 jan. 2022. Disponível em: <https://cmake.org/>. Acesso em: 13 out. 2022.
4. PATTERSON, David A.; HENNESSY, John L. **Computer Organization and Design**: The Hardware/Software Interface. 5. ed. [S. l.]: Elsevier, 2014.
5. PRAKASH, Abhishek. **Absolute vs Relative Path in Linux**: What's the Difference?. [S. l.], 30 abr. 2021. Disponível em: <https://linuxhandbook.com/absolute-vs-relative-path/>. Acesso em: 8 out. 2022.
6. SCOTT, Craig. **Proffesional CMake**: A Practical Guide. [S. l.: s. n.], 2018.