Step 4-2

In this exercise, we are going to demonstrate how using generator expressions can produce build configuration dependent information.

> A build configuration is a general setting that changes the value of several other smaller parameters.
> The usual values are `Release`, `Debug`, `RelWithDebInfo`, `MinSizeRel`.
> Ex: Selecting `Debug` with by default disable the compiler optimization and include the debug symbols
> Ex: Selecting `Release` will by default make sure the debug symbols are disabled and optimization are turned on

Generator expressions have a specific syntax: `$<...>` and are evaluated during the build system generation phase.
They should not be confused with simple variable unpacking (`${...}`) that are evaluated during the configuration phase.

A special generator expression is the conditional expression one. It looks like `$<condition:true_string>` and will:

> Be evaluated to its right part (`true_string`) if `condition` is 1
> Be evaluated to an empty string if `condition` is 0

Generator expressions are interesting to produce information according to the selected build configuration, but also any information computed at the generation phase like target properties or platform information.

They can also be nested:

> Ex: `$<CONFIG:Debug>` will be evaluated as 1 if the selected build configuration is `Debug` or 0 if any other build configuration is selected.
> This means that `$<$<CONFIG:Debug>:true_string>` will be evaluated as `true_string` if the selected build configuration is `Debug` or an empty string for any other build configuration

Config type dependent variables

Let's start by trying CMake variables that depend on the selected config type.
Try to find a variable that will add `-Wall` to the compiler flags when `Debug` is selected.
(The answer is somewhere in cmake-variables)

> Build the project in VERBOSE to check that the flag is correctly added when building in `Debug`

(Select the build type by setting `CMAKE_BUILD_TYPE` (make or ninja) or using `--config` when calling `cmake --build .` (MSVC or XCode, ninja-multiconfig))

Using information that are available only at generate time

Then let's make CMake create a file that will contain the `Tutorial` target compile features to help us debug our project.
The compile features are a good candidate for our example because their full list is computed during the generation phase.

Use the command file with the `GENERATE` and `CONTENT` keyword with the `$<TARGET_PROPERTY:target,property>` generator expression.
Try to use `$<CONFIG>` to name the file according to the selected config type.

```
file(GENERATE
  OUTPUT "Use `$<CONFIG>` to place the generated file in a folder whose
name depend on the selected build configuration"
  CONTENT "Use $<TARGET_PROPERTY:target,property> to put the target
Tutorial compile features as the file's content"
)
```

You can find a full list of target properties here: cmake-properties

Conditional linking

Finally, let's make some conditional linking ! Try to make CMake link `Tutorial` with `tutorial_compiler_flags` only if the `Release` config type is selected.
Hint: You will need to nest the generator expressions `$<condition:true_string>` and `$<CONFIG:cfgs>`
Hint: CMake is fine with linking a target with nothing. Eg: `target_link_libraries(Tutorial PUBLIC )` is valid CMake code and will just do nothing.

> Open the file generated by the file command and see its content.

Bonus

Bonus: why would this code not work to answer the previous question ?

```
if($<CONFIG> STREQUAL Release)
  set(EXTRA_LIBS tutorial_compiler_flags)
endif()
```