Building and Modularizing `mpags-cipher` with CMake

- Mark Slater (based on slides from Ben Morgan)

Developer Workflow

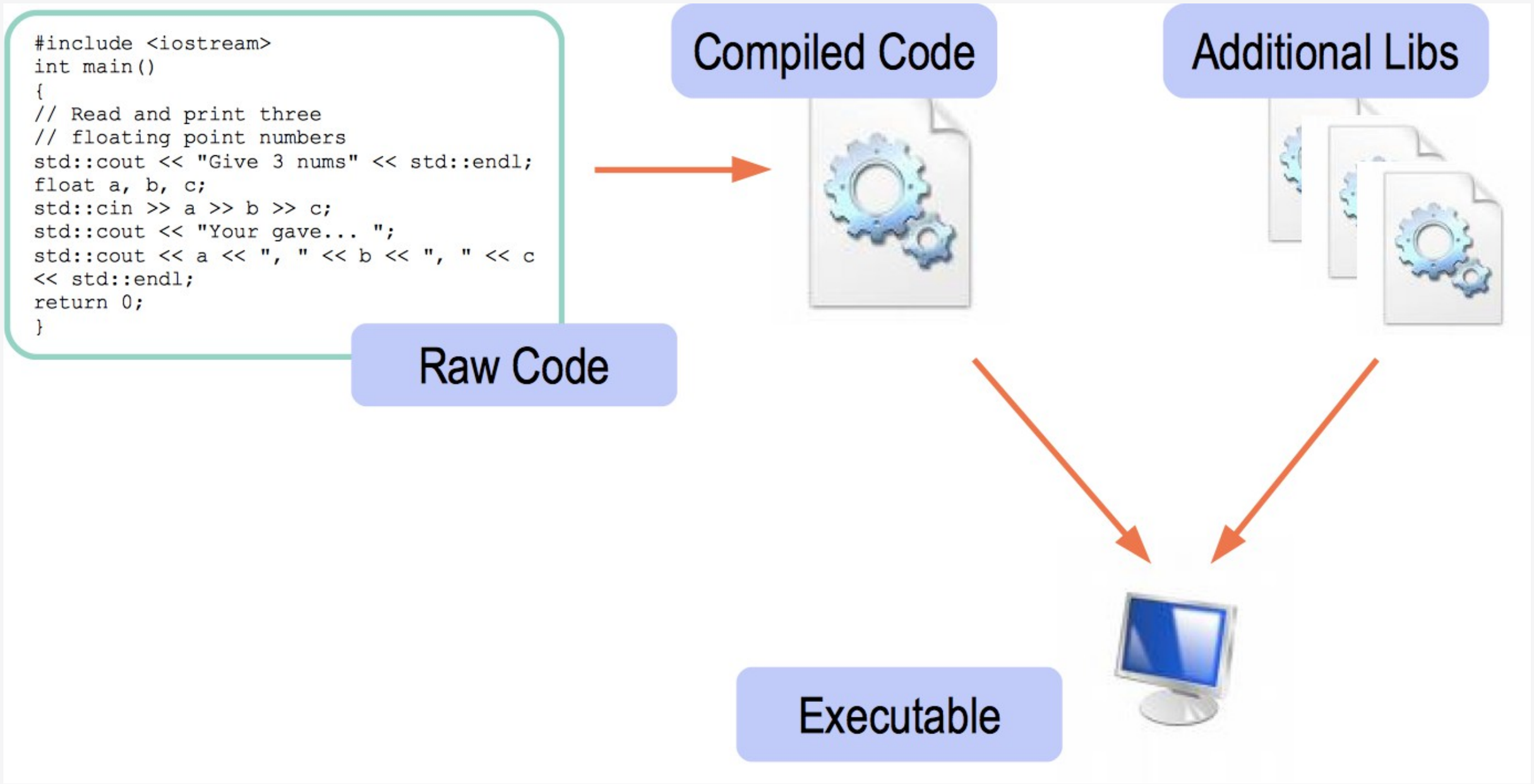
Build and Test

```
$ g++ mpags-cipher.cpp -o mpags-cipher
$ ./mpags-cipher
```

git add/commit
"Add CMake build"

Edit Sources
Add Files

Building a C++ based Executable



```
$ g++ -std=c++11 mpags-cipher.cpp -o mpags-cipher
$ ./mpags-cipher
$ git add mpags-cipher.cpp
$ git commit -m “A change”
$ vim mpags-cipher.cpp
$ g++ mpags-cipher.cpp -o mpags-cipher
$ ./mpags-cipher
$ g                                    pp
$ g                                    anges”
$ v
$ g                                    -o mp
$ ./mpags-cipher
$ git add mpags-cipher.cpp
$ git commit -m “Yet more changes”
$ vim mpags-cipher.cpp
...
```

How to tell other users what to do?

What if they have different compilers and need other flags?

What about system specific features?

What happens when we have more than one file to compile?

Is it really so awkward?

Build Automation

«configure»

Find needed tools
Adapt code for system
Generate build script

«build»

Compile code
Link binary
Test binary

«install»

Install binary on system
Create installer for other systems

GNU Make/Autotools

CMake

SCONS

Qt qmake

Visual Studio

Tools for Build Automation

mpags-cipher.git

README.md
mpags-cipher.cpp
CMakeLists.txt

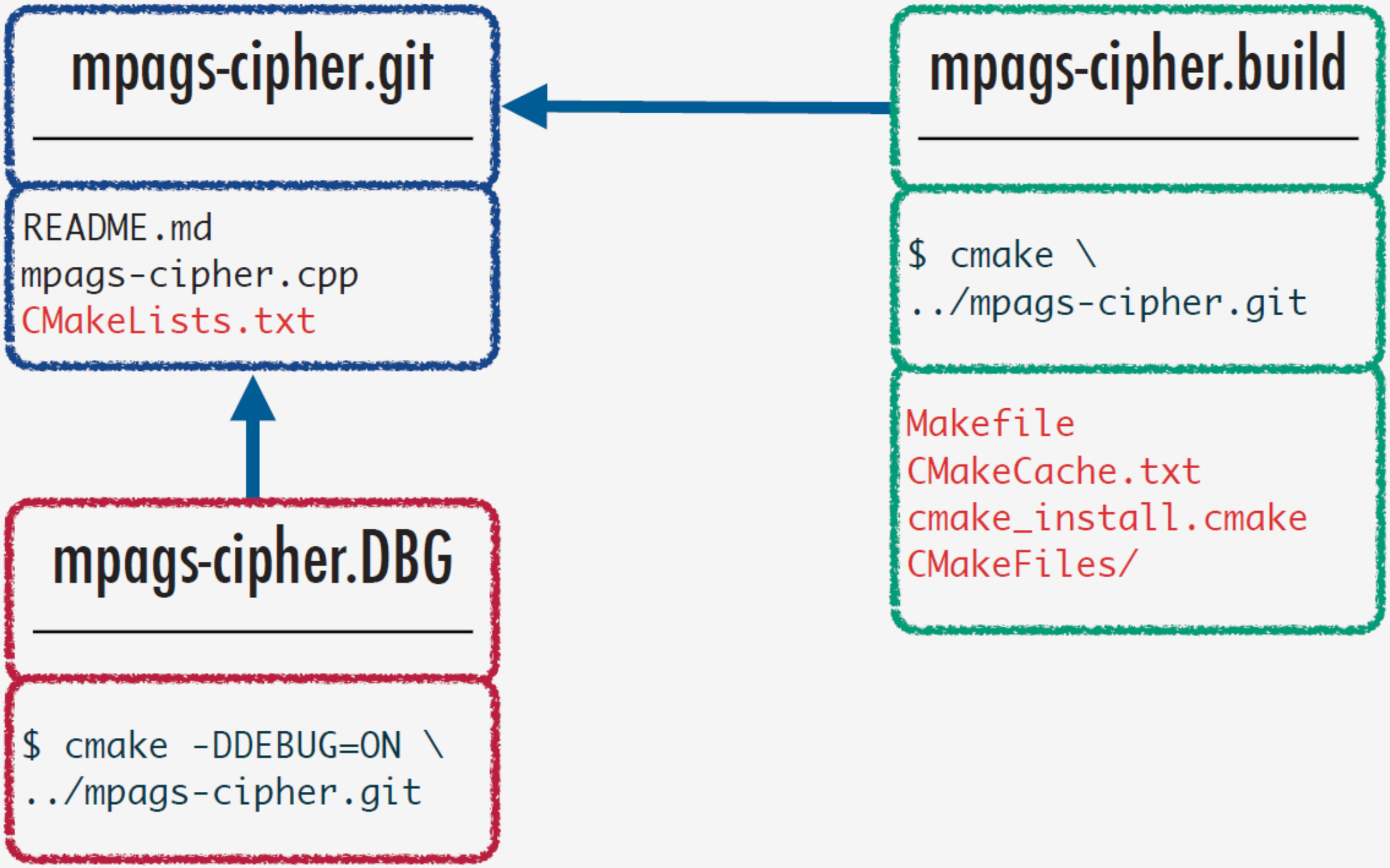
Makefile

mpags-cipher.xcodeproj

mpags-cipher.sln

Visual Studio

CMake - A "Metabuild" System



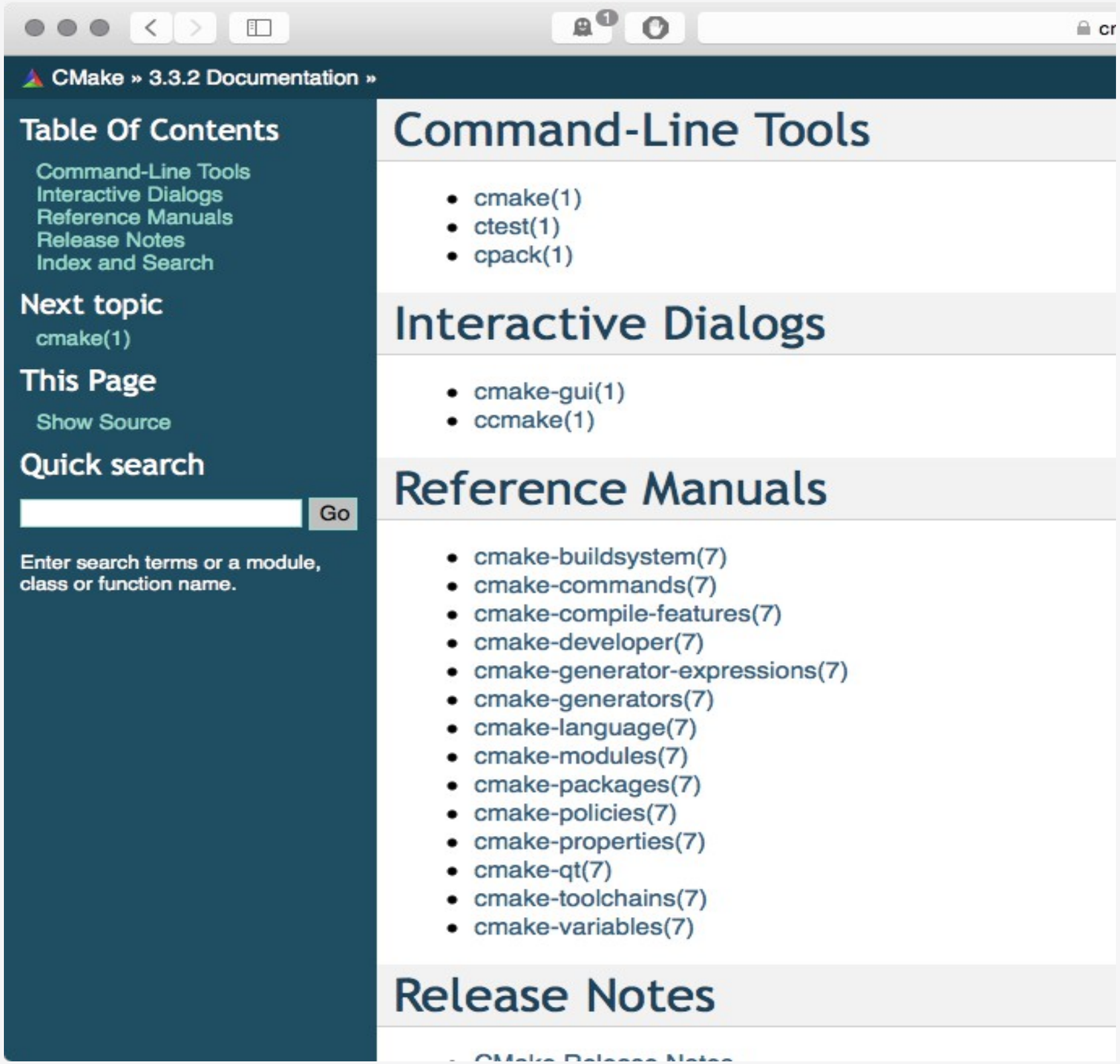
CMake Workflow - Separation of Source and Build

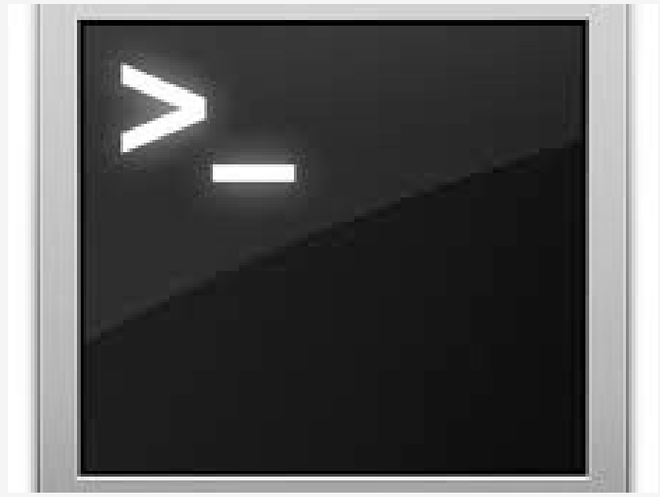
Building `mpags-cipher` with CMake

- In the following walkthrough, we'll bring in **CMake** to automate the build of mpags-cipher and integrate it to our workflow. We'll look at the basic elements of the CMake scripting language and how it helps automate things we've had to do manually so far like compiler selection and flags.

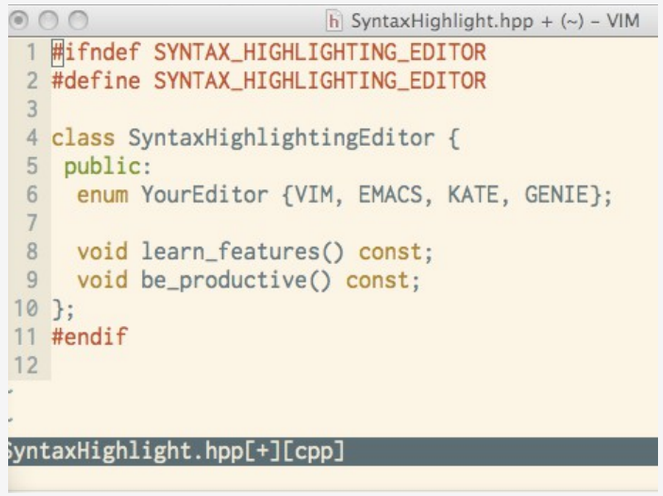
- We'll look again at function definitions and declarations, in particular how these allow us to divide code into separate header files declaring interfaces, and source files defining implementations. With more files to deal with, and more compiler-dependent flags to handle, CMake will be used to help manage these easily and portably.

- Of course, we'll continue to tracks changes using git and don't forget to update the README with new instructions on how to build!





CMake



Tools you'll need

1: Getting CMake

To get the best support for C++11, we're going to use CMake 3.2 or newer. As this is a relatively recent version, your system (if it's a laptop – the desktops are up to date) may not have it installed. To check this, open a terminal and run:

```
$ cmake --version
```

```
EPSC02PN49MFVH8:mpags-cipher.git slatermw$ cmake --version
cmake version 3.9.4

CMake suite maintained and supported by Kitware (kitware.com/cmake).
EPSC02PN49MFVH8:mpags-cipher.git slatermw$
```

Notes

If the version displayed isn't 3.2.SOMETHING or higher then you'll need to install cmake – let us know if this is the case!

If you see "command not found" or similar, then there is no CMake installed on the system, so you'll also need to install it.

2: Getting Help With CMake

There's plenty of documentation on the CMake website, and of course **`man cmake`** will give a good overview. For quick lookup of the commands, variables and properties that comprise CMake's scripting language, it's hard to beat CMake's command line help interface:

```
$ cmake --help-command project
```

```
EPSC02PN49MFVH8:mpags-cipher.git slatermw$ cmake --help-command project
project
-------

Set a name, version, and enable languages for the entire project.

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

Sets the name of the project and stores the name in the
``PROJECT_NAME`` variable.  Additionally this sets variables

* ``PROJECT_SOURCE_DIR``,
  ``<PROJECT-NAME>_SOURCE_DIR``
* ``PROJECT_BINARY_DIR``,
  ``<PROJECT-NAME>_BINARY_DIR``

If ``VERSION`` is specified, given components must be non-negative integers.
If ``VERSION`` is not specified, the default version is the empty string.
The ``VERSION`` option may not be used unless policy ``CMP0048`` is
set to ``NEW``.

The ``project()`` command stores the version number and its components
in variables

* ``PROJECT_VERSION``,
  ``<PROJECT-NAME>_VERSION``
* ``PROJECT_VERSION_MAJOR``,
```

Useful Help Commands

--help-variable NAME
Print help on variable

--help-property NAME
Print help on property

--help-<type>-list
List names of available <type> (command,variable or property)

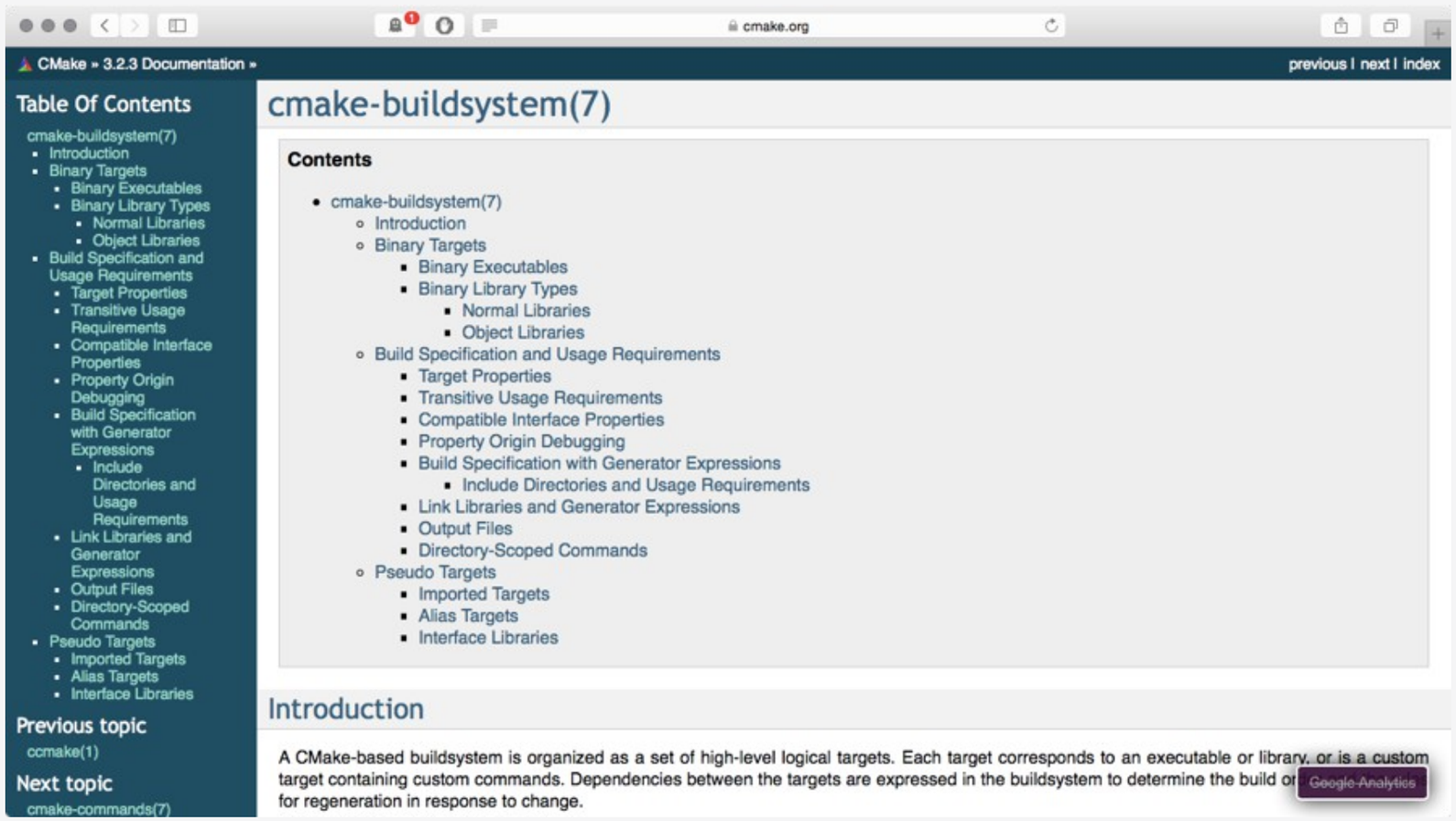
--help-commands
List help for all commands

3: The CMake Scripting Language

CMake scripts are written in its own scripting language. This is generally simple to use and has most of the flow control and conditional statements familiar from C++ (and other languages). It is now generally well documented on the CMake site, so refer to these through the course:

https://cmake.org/cmake/help/v3.2/manual/cmake-buildsystem.7.html

https://cmake.org/cmake/help/v3.2/manual/cmake-language.7.html



Notes

The language is very simple so we won't go into it in detail, other than to refer you to the linked documentation.

It does have its idiosyncrasies, but works well in its domain of describing software builds!

4: A First CMake Script for mpags-cipher

Starting with CMake is straightforward - we simply create a file named `CMakeLists.txt` (it cannot have any other name) in the root directory (e.g. `mpags-cipher.git`) of the project. Written in CMake's scripting language, this is essentially another program that declares how we want to build our project. To begin with we simply add three commands. The first two to check we're running a suitable version of CMake, and perform minimal system setup for our project. The third sets an internal CMake variable to output more information when we go to build.

```
# - Main CMake buildscript for mpags-cipher
# Comments in a CMake Script are lines begining with a '#'

# - Set CMake requirements then declare project
cmake_minimum_required(VERSION 3.2)
project(MPAGSCipher VERSION 0.1.0)

# - When Makefiles are generated, output all command lines by default
#   Do this to begin with so it's easy to see what compiler command/flags
#   are used. This can also be done by removing the 'set' command and
#   running make as 'make VERBOSE=1'.
set(CMAKE_VERBOSE_MAKEFILE ON)


-uuu:**-F1  CMakeLists.txt   All L13    (CMake)----------------------------------------------
Auto-saving...done
```

Notes

You can supply more detailed version numbers to cmake_minimum_required if you need specific versions. Kitware provide a Version Compatibility Matrix on their wiki

You can choose the name of the project as needed.

Use `cmake --help command` for more info!

5: Creating the Build Directory

As mentioned earlier, a good practice is to perform the build in a separate directory. Even though our current CMake won't build anything yet, we'll get this directory set up first to illustrate the basic usage of Cmake.

Create a build directory for mpags-cipher outside of your repository. The easiest thing to do is create this directory parallel to the repository as shown below (ignore the Documentation and Testing folders in the image!):

```
EPSC02PN49MFVH8:mpags-cipher.git slatermw$ tree -C .
.
├── CMakeLists.txt
├── LICENSE
├── README.md
└── mpags-cipher.cpp

0 directories, 4 files
EPSC02PN49MFVH8:mpags-cipher.git slatermw$ cd ../
EPSC02PN49MFVH8:mpags slatermw$ tree -C .
.
├── mpags-cipher.build
└── mpags-cipher.git
    ├── CMakeLists.txt
    ├── LICENSE
    ├── README.md
    └── mpags-cipher.cpp

2 directories, 4 files
EPSC02PN49MFVH8:mpags slatermw$
```

Notes

This parallel build structure is why we set up a two-level structure for the project and its git repository – we've isolated both the repo and build from other files. You can create the build directory wherever you want. The parallel creation pattern is simply for convenience.

6: Using CMake to Generate Makefiles

With the build directory created, we can now run CMake to generate the build scripts. On UNIX systems, CMake generates Makefiles by default. To see the other build systems supported by CMake, see the "Generators" section of the CMake documentation. To generate the Makefiles, simply run the `cmake` command in your build directory, and pass it the path to your "source directory", i.e. the directory holding the main `CMakeLists.txt` file for the project being built.

```
EPSC02PN49MFVH8:mpags slatermw$ ls
mpags-cipher.build mpags-cipher.git
EPSC02PN49MFVH8:mpags slatermw$ cd mpags-cipher.build/
EPSC02PN49MFVH8:mpags-cipher.build slatermw$ cmake ../mpags-cipher.git/
-- The C compiler identification is AppleClang 7.0.0.7000176
-- The CXX compiler identification is AppleClang 7.0.0.7000176
-- Check for working C compiler: /Applications/Xcode.app/Contents/Developer/Toolchains/XcodeDefa
ult.xctoolchain/usr/bin/cc
-- Check for working C compiler: /Applications/Xcode.app/Contents/Developer/Toolchains/XcodeDefa
ult.xctoolchain/usr/bin/cc -- works
-- Detecting C compiler ABI info
-- Detecting C compiler ABI info - done
-- Detecting C compile features
-- Detecting C compile features - done
-- Check for working CXX compiler: /Applications/Xcode.app/Contents/Developer/Toolchains/XcodeDe
fault.xctoolchain/usr/bin/c++
-- Check for working CXX compiler: /Applications/Xcode.app/Contents/Developer/Toolchains/XcodeDe
fault.xctoolchain/usr/bin/c++ -- works
-- Detecting CXX compiler ABI info
-- Detecting CXX compiler ABI info - done
-- Detecting CXX compile features
-- Detecting CXX compile features - done
-- Configuring done
-- Generating done
-- Build files have been written to: /Users/slatermw/mpags/mpags-cipher.build
EPSC02PN49MFVH8:mpags-cipher.build slatermw$
```

Notes

You can supply the path as a relative path from your build directory to the source directory, or as a full absolute path.

The output shows CMake checking the availability and functionality of the C/C++ compilers. As well as the build scripts it will have also created a text file with the configuration details in: **`CMakeCache.txt`**

7: Makefile

This is the script generated by CMake for use with the make build tool. It's a text file which you can view with a pager like `less`, though we won't need to worry its syntax here.

To use the script, simply type `make` when in the build directory - though it doesn't do much yet! The verbose output (we set `CMAKE_VERBOSE_MAKEFILE` to 'ON' in the `CMakeList.txt`) will help in the next few steps. You can also run `make help` to see the list of "targets". Try building each target using `make <targetname>`

```
EPSC02PN49MFVH8:mpags-cipher.build slatermw$ make
/usr/local/Cellar/cmake/3.9.4_1/bin/cmake -H/Users/slatermw/mpags/mpags-cipher.git -B/Users/slat
ermw/mpags/mpags-cipher.build --check-build-system CMakeFiles/Makefile.cmake 0
/usr/local/Cellar/cmake/3.9.4_1/bin/cmake -E cmake_progress_start /Users/slatermw/mpags/mpags-ci
pher.build/CMakeFiles /Users/slatermw/mpags/mpags-cipher.build/CMakeFiles/progress.marks
/Applications/Xcode.app/Contents/Developer/usr/bin/make -f CMakeFiles/Makefile2 all
make[1]: Nothing to be done for `all'.
/usr/local/Cellar/cmake/3.9.4_1/bin/cmake -E cmake_progress_start /Users/slatermw/mpags/mpags-ci
pher.build/CMakeFiles 0
EPSC02PN49MFVH8:mpags-cipher.build slatermw$ make help
The following are some of the valid targets for this Makefile:
... all (the default if no target is provided)
... clean
... depend
... rebuild_cache
... edit_cache
EPSC02PN49MFVH8:mpags-cipher.build slatermw$ make rebuild_cache
Running CMake to regenerate build system...
/usr/local/Cellar/cmake/3.9.4_1/bin/cmake -H/Users/slatermw/mpags/mpags-cipher.git -B/Users/slat
ermw/mpags/mpags-cipher.build
-- Configuring done
-- Generating done
-- Build files have been written to: /Users/slatermw/mpags/mpags-cipher.build
EPSC02PN49MFVH8:mpags-cipher.build slatermw$
```

Main Targets

`all:`

Build everything, the target built if you just type make.

`clean:`

Remove all built files

`rebuild_cache:`

Force rerun of CMake

`edit_cache:`

Start up ccmake

8: Getting CMake to Build mpags-cipher

To compile and link an executable in CMake, we use its `add_executable` command. This takes the name you want the program to have followed by a space separated list of all the sources that need to be compiled to create the program. Currently, we only have one source, so all we need to add to our CMake script is the single line

```
add_executable(mpags-cipher mpags-cipher.cpp)
```

```
# - Main CMake buildscript for mpags-cipher
# Comments in a CMake Script are lines begining with a '#'

# - Set CMake requirements then declare project
cmake_minimum_required(VERSION 3.2)
project(MPAGSCipher VERSION 0.1.0)

# - When Makefiles are generated, output all command lines by default
#   Do this to begin with so it's easy to see what compiler command/flags
#   are used. This can also be done by removing the 'set' command and
#   running make as 'make VERBOSE=1'.
set(CMAKE_VERBOSE_MAKEFILE ON)

# - Declare the build of mpags-cipher main program
add_executable(mpags-cipher mpags-cipher.cpp)

-uu-:**-F1  CMakeLists.txt   All L15    (CMake)----------------------------------------------
```

Notes

`add_executable` considers all relative paths passed to it as relative to the location of the CMake script it is called in.

After saving the file, don't add/commit it with git yet because we need to test it!

9: Building mpags-cipher

Change back to your build directory (where you ran CMake before). If you already ran cmake in here, simply type `make` and you should find that CMake automatically reruns, before trying to build `mpags-cipher`. Be aware that it will most likely fail – you'll find out how to fix this in the next slide!

```
EPSC02PN49MFVH8:mpags-cipher.build slatermw$ make
/usr/local/Cellar/cmake/3.9.4_1/bin/cmake -H/Users/slatermw/mpags/mpags-cipher.git -B/Users/slat
ermw/mpags/mpags-cipher.build --check-build-system CMakeFiles/Makefile.cmake 0
/usr/local/Cellar/cmake/3.9.4_1/bin/cmake -E cmake_progress_start /Users/slatermw/mpags/mpags-ci
pher.build/CMakeFiles /Users/slatermw/mpags/mpags-cipher.build/CMakeFiles/progress.marks
/Applications/Xcode.app/Contents/Developer/usr/bin/make -f CMakeFiles/Makefile2 all
/Applications/Xcode.app/Contents/Developer/usr/bin/make -f CMakeFiles/mpags-cipher.dir/build.mak
e CMakeFiles/mpags-cipher.dir/depend
cd /Users/slatermw/mpags/mpags-cipher.build && /usr/local/Cellar/cmake/3.9.4_1/bin/cmake -E cmak
e_depends "Unix Makefiles" /Users/slatermw/mpags/mpags-cipher.git /Users/slatermw/mpags/mpags-ci
pher.git /Users/slatermw/mpags/mpags-cipher.build /Users/slatermw/mpags/mpags-cipher.build /User
s/slatermw/mpags/mpags-cipher.build/CMakeFiles/mpags-cipher.dir/DependInfo.cmake --color=
/Applications/Xcode.app/Contents/Developer/usr/bin/make -f CMakeFiles/mpags-cipher.dir/build.mak
e CMakeFiles/mpags-cipher.dir/build
[ 50%] Building CXX object CMakeFiles/mpags-cipher.dir/mpags-cipher.cpp.o
/Applications/Xcode.app/Contents/Developer/Toolchains/XcodeDefault.xctoolchain/usr/bin/c++    -s
td=gnu++11 -o CMakeFiles/mpags-cipher.dir/mpags-cipher.cpp.o -c /Users/slatermw/mpags/mpags-ciph
er.git/mpags-cipher.cpp
[100%] Linking CXX executable mpags-cipher
/usr/local/Cellar/cmake/3.9.4_1/bin/cmake -E cmake_link_script CMakeFiles/mpags-cipher.dir/link.
txt --verbose=1
/Applications/Xcode.app/Contents/Developer/Toolchains/XcodeDefault.xctoolchain/usr/bin/c++   -Wl
,-search_paths_first -Wl,-headerpad_max_install_names  CMakeFiles/mpags-cipher.dir/mpags-cipher.
cpp.o  -o mpags-cipher
[100%] Built target mpags-cipher
/usr/local/Cellar/cmake/3.9.4_1/bin/cmake -E cmake_progress_start /Users/slatermw/mpags/mpags-ci
pher.build/CMakeFiles 0
EPSC02PN49MFVH8:mpags-cipher.build slatermw$
```

Notes

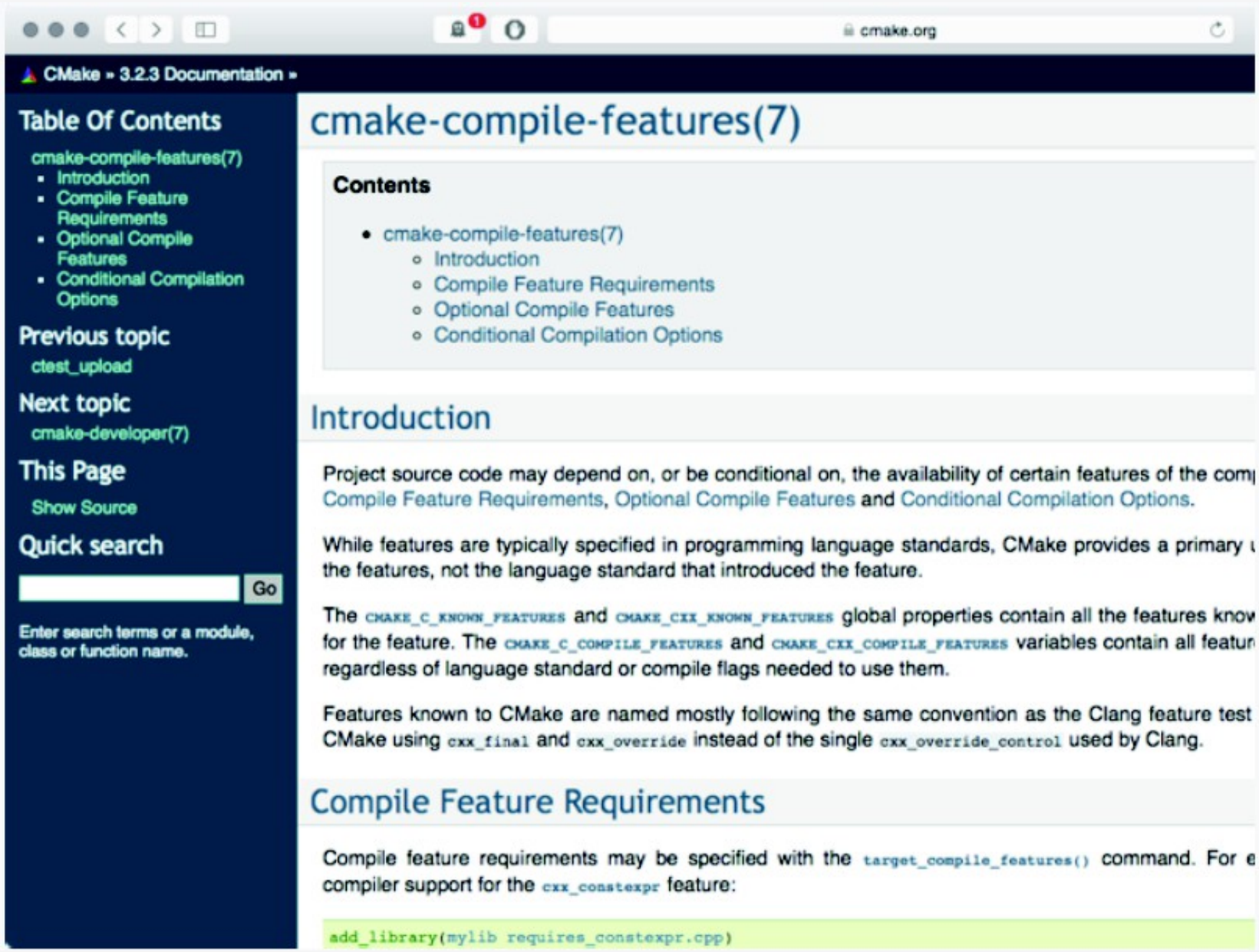
CMake generates build scripts that track changes to your CMake build scripts. Thus you don't need to rerun CMake all the time, just run make, and it'll automatically run CMake for you.

If you do need to start from scratch you can simply remove CmakeCache.txt and rerun cmake, or just remove the build directory.

10: Adding Compiler Flags for C++11

You may have found that this failed to compile. As mentioned last week, some compilers need to have a flag set to enable the C++11 standard. We could add this in CMake by hand (and we'll see how to do this for other flags later), but we'd have to hard code in knowledge of different compilers and which versions support different versions of the C++ Standard. Instead, we're going to use an easier method, CMake Compile Features:

https://cmake.org/cmake/help/v3.2/manual/cmake-compile-features.7.html



Notes

Not all compilers require flags to select the C++ standard. The Microsoft compiler is the main example here.

Compile Features provide an easier and platform independent way to specify what we need the compiler to support

11: Compile Features For mpags-cipher

To declare the compile features we need for `mpags-cipher`, we use CMake's `target_compile_features` command. This takes the name of the "target" (program or library) a "scope" flag and a (scoped) list of the compile features the target's sources use, and require compiler support for. Compile features are simply strings describing the feature used, e.g. `cxx_auto_type` when the code uses, e.g. `auto foo = 1;`

After adding appropriate C++11 features and re-running `make`, you'll see `-std=c++11` added!

```
# - Main CMake buildscript for mpags-cipher
# Comments in a CMake Script are lines begining with a '#'

# - Set CMake requirements then declare project
cmake_minimum_required(VERSION 3.2)
project(MPAGSCipher VERSION 0.1.0)

# - When Makefiles are generated, output all command lines by default
#   Do this to begin with so it's easy to see what compiler command/flags
#   are used. This can also be done by removing the 'set' command and
#   running make as 'make VERBOSE=1'.
set(CMAKE_VERBOSE_MAKEFILE ON)

# - Declare the build of mpags-cipher main program
add_executable(mpags-cipher mpags-cipher.cpp)

target_compile_features(mpags-cipher
  PRIVATE cxx_auto_type cxx_range_for cxx_uniform_initialization
  )
```

```
-uu-:---F1  CMakeLists.txt    All L1     (CMake)----------------------------------------------
Loading /usr/local/share/cmake/editors/emacs/cmake-mode.el (source)...done
```

Try This

Set the variable `CMAKE_CXX_EXTENSIONS` to OFF first to prevent vendor extensions to C++11. Add compile features for mpags-cipher – review the documentation to decide which ones you need.

Use the PRIVATE "scope" flag for the features as shown. We'll look at this in more detail later.

12: How Compile Features Help

We haven't had to concern ourselves with exactly which version of compiler we are using nor what parts of the C++ Standard it supports. With C++ moving to a shorter update cycle, this will become more important. **For example, add** `cxx_binary_literals` **to the compile features of** `mpags-cipher` **and rebuild.** You should see that the `-std` flag has changed to that for the C++14 standard **if your compiler supports it**. Otherwise, you'll get an error when CMake runs telling you that the feature is not known to the compiler, as shown below.

```
EPSC02PN49MFVH8:mpags-cipher.build slatermw$ make
/usr/local/Cellar/cmake/3.9.4_1/bin/cmake -H/Users/slatermw/mpags/mpags-cipher.git -B/Users/slat
ermw/mpags/mpags-cipher.build --check-build-system CMakeFiles/Makefile.cmake 0
/usr/local/Cellar/cmake/3.9.4_1/bin/cmake -E cmake_progress_start /Users/slatermw/mpags/mpags-ci
pher.build/CMakeFiles /Users/slatermw/mpags/mpags-cipher.build/CMakeFiles/progress.marks
/Applications/Xcode.app/Contents/Developer/usr/bin/make -f CMakeFiles/Makefile2 all
/Applications/Xcode.app/Contents/Developer/usr/bin/make -f CMakeFiles/mpags-cipher.dir/build.mak
e CMakeFiles/mpags-cipher.dir/depend
cd /Users/slatermw/mpags/mpags-cipher.build && /usr/local/Cellar/cmake/3.9.4_1/bin/cmake -E cmak
e_depends "Unix Makefiles" /Users/slatermw/mpags/mpags-cipher.git /Users/slatermw/mpags/mpags-ci
pher.git /Users/slatermw/mpags/mpags-cipher.build /Users/slatermw/mpags/mpags-cipher.build /User
s/slatermw/mpags/mpags-cipher.build/CMakeFiles/mpags-cipher.dir/DependInfo.cmake --color=
/Applications/Xcode.app/Contents/Developer/usr/bin/make -f CMakeFiles/mpags-cipher.dir/build.mak
e CMakeFiles/mpags-cipher.dir/build
[ 50%] Building CXX object CMakeFiles/mpags-cipher.dir/mpags-cipher.cpp.o
 Applications/Xcode.app/Contents/Developer/Toolchains/XcodeDefault.xctoolchain/usr/bin/c++    -s
td=gnu++14 -o CMakeFiles/mpags-cipher.dir/mpags-cipher.cpp.o -c /Users/slatermw/mpags/mpags-ciph
er.git/mpags-cipher.cpp
[100%] Linking CXX executable mpags-cipher
/usr/local/Cellar/cmake/3.9.4_1/bin/cmake -E cmake_link_script CMakeFiles/mpags-cipher.dir/link.
txt --verbose=1
/Applications/Xcode.app/Contents/Developer/Toolchains/XcodeDefault.xctoolchain/usr/bin/c++   -Wl
,-search_paths_first -Wl,-headerpad_max_install_names  CMakeFiles/mpags-cipher.dir/mpags-cipher.
cpp.o  -o mpags-cipher
[100%] Built target mpags-cipher
/usr/local/Cellar/cmake/3.9.4_1/bin/cmake -E cmake_progress_start /Users/slatermw/mpags/mpags-ci
pher.build/CMakeFiles 0
EPSC02PN49MFVH8:mpags-cipher.build slatermw$
```

Notes

This is classic example of a build system handling the details for us!

Though other tools don't have "compile features" directly, they all provide a "try-compile" pattern. This is used to exercise the compiler and find out what it can do. Its results can be used to workaround issues or warn the user as needed.

13: Adding Additional Compiler Flags

Compilers provide a vast range of flags, so CMake can't set all of them for us. You'll notice that additional warnings like `-Wall` that we want to use are not yet set. The default flags used by the C++ compiler can be changed by setting the CMake variable `CMAKE_CXX_FLAGS` to a quoted string containing the flags we want to use.

```
# - Main CMake buildscript for mpags-cipher
# Comments in a CMake Script are lines begining with a '#'

# - Set CMake requirements then declare project
cmake_minimum_required(VERSION 3.2)
project(MPAGSCipher VERSION 0.1.0)

# - When Makefiles are generated, output all command lines by default
#   Do this to begin with so it's easy to see what compiler command/flags
#   are used. This can also be done by removing the 'set' command and
#   running make as 'make VERBOSE=1'.
set(CMAKE_VERBOSE_MAKEFILE ON)

# - Declare the build of mpags-cipher main program
add_executable(mpags-cipher mpags-cipher.cpp)

# - Don't allow C++ Compiler Vendor Extensions
set(CMAKE_CXX_EXTENSIONS OFF)

# - Use our standard set of flags
set(CMAKE_CXX_FLAGS "-Wall -Wextra -Werror -Wfatal-errors -Wshadow -pedantic")

target_compile_features(mpags-cipher
  PRIVATE cxx_auto_type cxx_range_for cxx_uniform_initialization
  )

-uu-:**-F1  CMakeLists.txt    All L26    (CMake)----------------------------------------------------
```

Try This

Set `CMAKE_CXX_FLAGS` to the list we've been using when compiling manually.

Of course, check that the flags are passed to the compiler by rebuilding with make and reviewing the compile commands!

14: Setting the C++ Standard Directly

If you don't know what specific Compile Features you need to include but you **do** know that you need a specific standard, you can also use:

```
set (CMAKE_CXX_STANDARD_REQUIRED ON)
set (CMAKE_CXX_STANDARD 11)
```

to enforce that you need a particular standard and then specifying which one (C++ 11, 14, 17, etc.). We don't need this now but we will need it later to specifically ask for the C++ 14 standard

```
# - Main CMake buildscript for mpags-cipher
# Comments in a CMake Script are lines begining with a '#'

# - Set CMake requirements then declare project
cmake_minimum_required(VERSION 3.2)
project(MPAGSCipher VERSION 0.1.0)

# - When Makefiles are generated, output all command lines by default
#   Do this to begin with so it's easy to see what compiler command/flags
#   are used. This can also be done by removing the 'set' command and
#   running make as 'make VERBOSE=1'.
set(CMAKE_VERBOSE_MAKEFILE ON)

# - Set the C++ Standard directly
set(CMAKE_CXX_STANDARD 14)

# - Declare the build of mpags-cipher main program
add_executable(mpags-cipher mpags-cipher.cpp)

# - Don't allow C++ Compiler Vendor Extensions
set(CMAKE_CXX_EXTENSIONS OFF)

# - Use our standard set of flags
set(CMAKE_CXX_FLAGS "-Wall -Wextra -Werror -Wfatal-errors -Wshadow -pedantic")

target_compile_features(mpags-cipher
  PRIVATE cxx_auto_type cxx_range_for cxx_uniform_initialization
  )


-uu-:---F1  CMakeLists.txt    All L16     (CMake)----------------------------------------------------
Wrote /Users/slatermw/MPAGS-Code/CMakeLists.txt
```

Try This

Try setting the standard to C++ 14 and check the appropriate compiler flag has been added.

Make sure you add the lines before any defined targets!

You can also set this for a particular target using `'set_property'`

15: Reviewing mpags-cipher

Whilst mpags-cipher is relatively simple, it's already over 150 lines long with three functions, `transformChar`, `processCommandLine` and `main`. As we add further functionality like reading/writing files and the ciphers themselves, this complexity will only grow.

Whilst functions will help (and objects later), managing changes to different bits of functionality in a single file will get tricky. In the next few steps we'll see how can separate functionality into separate files to isolate them and allow them to evolve separately.

```
// Standard Library includes
#include <fstream>
#include <iostream>
#include <string>
#include <vector>

// For std::isalpha and std::isupper
#include <cctype>

std::string transformChar(const char in_char)
{
  /*
     take the input character and transform it according to the following:
     - Must be uppercase
     - Ignore alphanumberic characters
     - Change numbers to english words, e.g. '0' -> ZERO

     char in_char: The character to transform

     return: The string generated by the transformed character
  */

  std::string out_text{""};

  // Uppercase alphabetic characters
  if (std::isalpha(in_char)) {
    out_text += std::toupper(in_char);
  }

-uu-:---F1  mpags-cipher.cpp    Top L6     (C++/l Abbrev)----------------------------------------
Wrote /Users/slatermw/mpags/mpags-cipher.git/mpags-cipher.cpp
```

Notes

This will be a very simple exercise where you may argue the separation isn't needed!

The core objectives are to see how we can partition code up between files, compile all the code into a single program, and to illustrate the concept on a interface.

sdcsd

16: Function Definitions and Declarations

Before we can use a function in C++, it must be known to the compiler. You might have seen this already when using `std::cout` if you forgot to include the `iostream` header.

In your `mpags-cipher` program, move the `transformChar` function you've implemented to the end of the `.cpp` file so that the `main` function is the first that appears in the file. Try recompiling - does it work?

```
// Standard Library includes
#include <fstream>
#include <iostream>
#include <string>
#include <vector>

// For std::isalpha and std::isupper
#include <cctype>

//! Process the command line arguments
bool processCommandLine(const std::vector<std::string>& args,
                        bool& helpRequested,
                        bool& versionRequested,
                        std::string& inputFile,
                        std::string& outputFile,
                        std::string& cipher_key,
                        bool& encrypt)
{
  // Status flag to indicate whether or not the parsing was successful
  bool processStatus(true);

  // Add a typedef that assigns another name for the given type for clarity
  typedef std::vector<std::string>::size_type size_type;
  const size_type nArgs {args.size()};

  // Process the arguments - ignore zeroth element, as we know this to be the
  // program name and don't need to worry about it
  for (size_type i {1}; i < nArgs; ++i) {

-uu-:**-F1  mpags-cipher.cpp   Top L10    (C++/l Abbrev)----------------------------------------
```

Notes

At present we only have an **implementation**, or **definition** for `transformChar`

17: No Declaration, No Go

You'll have found the compilation fails with the compiler reporting that `transformChar` is an "*undeclared identifier*" or "*not declared in this scope*".

This occurs because we've tried to **use** the function in main **before the compiler has seen it.**

We could fix this by moving the definition of transformChar back to before main, but instead we'll inform the compiler about it by adding a **function declaration** for it before main

```
EPSC02PN49MFVH8:mpags-cipher.build slatermw$ make
/usr/local/Cellar/cmake/3.9.4_1/bin/cmake -H/Users/slatermw/mpags/mpags-cipher.git -B/Users/slat
ermw/mpags/mpags-cipher.build --check-build-system CMakeFiles/Makefile.cmake 0
/usr/local/Cellar/cmake/3.9.4_1/bin/cmake -E cmake_progress_start /Users/slatermw/mpags/mpags-ci
pher.build/CMakeFiles /Users/slatermw/mpags/mpags-cipher.build/CMakeFiles/progress.marks
/Applications/Xcode.app/Contents/Developer/usr/bin/make -f CMakeFiles/Makefile2 all
/Applications/Xcode.app/Contents/Developer/usr/bin/make -f CMakeFiles/mpags-cipher.dir/build.mak
e CMakeFiles/mpags-cipher.dir/depend
cd /Users/slatermw/mpags/mpags-cipher.build && /usr/local/Cellar/cmake/3.9.4_1/bin/cmake -E cmak
e_depends "Unix Makefiles" /Users/slatermw/mpags/mpags-cipher.git /Users/slatermw/mpags/mpags-ci
pher.git /Users/slatermw/mpags/mpags-cipher.build /Users/slatermw/mpags/mpags-cipher.build /User
s/slatermw/mpags/mpags-cipher.build/CMakeFiles/mpags-cipher.dir/DependInfo.cmake --color=
/Applications/Xcode.app/Contents/Developer/usr/bin/make -f CMakeFiles/mpags-cipher.dir/build.mak
e CMakeFiles/mpags-cipher.dir/build
[ 33%] Building CXX object CMakeFiles/mpags-cipher.dir/mpags-cipher.cpp.o
/Applications/Xcode.app/Contents/Developer/Toolchains/XcodeDefault.xctoolchain/usr/bin/c++    -W
all -Wextra -Werror -Wfatal-errors -Wshadow -pedantic   -std=gnu++14 -o CMakeFiles/mpags-cipher.
dir/mpags-cipher.cpp.o -c /Users/slatermw/mpags/mpags-cipher.git/mpags-cipher.cpp
/Users/slatermw/mpags/mpags-cipher.git/mpags-cipher.cpp:173:15: fatal error: use of undeclared
      identifier 'transformChar'
        inputText += transformChar(inputChar);
                     ^
1 error generated.
make[2]: *** [CMakeFiles/mpags-cipher.dir/mpags-cipher.cpp.o] Error 1
make[1]: *** [CMakeFiles/mpags-cipher.dir/all] Error 2
make: *** [all] Error 2
EPSC02PN49MFVH8:mpags-cipher.build slatermw$
```

Notes

You may see the term **function prototype** used interchangeably with function declaration. For C++ this is o.k., but in older style C there is a difference!

Here, you can **declare** a function without specifying the types of the arguments. A **prototype** is a declaration that includes the number and type of its arguments.

18: Declaring Functions

To declare a function, we add a statement that specifies its return type, name, and types of the arguments it takes, omitting the body, i.e. the implementation or **definition**, of the function. The **declaration** tells the compiler about the function interface, and promises that its **definition** will be found "somewhere else". **Add a declaration for `transformChar` at the beginning of your main program. Check that you can now compile and run the program o.k.**

```
// Standard Library includes
#include <fstream>
#include <iostream>
#include <string>
#include <vector>

// For std::isalpha and std::isupper
#include <cctype>

//! Transliterate char to string
std::string transformChar(const char in);

//! Process the command line arguments
bool processCommandLine(const std::vector<std::string>& args,
                        bool& helpRequested,
                        bool& versionRequested,
                        std::string& inputFile,
                        std::string& outputFile,
                        std::string& cipher_key,
                        bool& encrypt)
{
  // Status flag to indicate whether or not the parsing was successful
  bool processStatus(true);

  // Add a typedef that assigns another name for the given type for clarity
  typedef std::vector<std::string>::size_type size_type;
  const size_type nArgs {args.size()};

  // Process the arguments - ignore zeroth element, as we know this to be the
-uu-:---F1  mpags-cipher.cpp   Top L13    (C++/l Abbrev)----------------------------------------
Wrote /Users/slatermw/mpags/mpags-cipher.git/mpags-cipher.cpp
```

Notes

Function declarations are our first concrete example of an **interface**. The key point to grasp is that an interface frees us from worrying **how** a task is done, just that **it is done**.

The implementation might be intellectually interesting, but knowledge of it is not needed to **use** the interface.

19: Declarations Without Definitions

We've seen what happens when we try to use a function before declaring it, but what happens if its **definition** (implementation) is missing?

Comment out the definition of `transformChar` but leave its declaration and usage in place. You should see an error about a *missing or undefined symbol*. This illustrates that a declaration is just a hint to the compiler - it's the linking step of compilation that finds the actual implementation and connects it to where it's used.

```
ermw/mpags/mpags-cipher.build --check-build-system CMakeFiles/Makefile.cmake 0
-- Configuring done
-- Generating done
-- Build files have been written to: /Users/slatermw/mpags/mpags-cipher.build
/usr/local/Cellar/cmake/3.9.4_1/bin/cmake -E cmake_progress_start /Users/slatermw/mpags/mpags-ci
pher.build/CMakeFiles /Users/slatermw/mpags/mpags-cipher.build/CMakeFiles/progress.marks
/Applications/Xcode.app/Contents/Developer/usr/bin/make -f CMakeFiles/Makefile2 all
/Applications/Xcode.app/Contents/Developer/usr/bin/make -f CMakeFiles/mpags-cipher.dir/build.mak
e CMakeFiles/mpags-cipher.dir/depend
cd /Users/slatermw/mpags/mpags-cipher.build && /usr/local/Cellar/cmake/3.9.4_1/bin/cmake -E cmak
e_depends "Unix Makefiles" /Users/slatermw/mpags/mpags-cipher.git /Users/slatermw/mpags/mpags-ci
pher.git /Users/slatermw/mpags/mpags-cipher.build /Users/slatermw/mpags/mpags-cipher.build /User
s/slatermw/mpags/mpags-cipher.build/CMakeFiles/mpags-cipher.dir/DependInfo.cmake --color=
Scanning dependencies of target mpags-cipher
/Applications/Xcode.app/Contents/Developer/usr/bin/make -f CMakeFiles/mpags-cipher.dir/build.mak
e CMakeFiles/mpags-cipher.dir/build
[ 50%] Linking CXX executable mpags-cipher
/usr/local/Cellar/cmake/3.9.4_1/bin/cmake -E cmake_link_script CMakeFiles/mpags-cipher.dir/link.
txt --verbose=1
/Applications/Xcode.app/Contents/Developer/Toolchains/XcodeDefault.xctoolchain/usr/bin/c++  -Wal
l -Wextra -Werror -Wfatal-errors -Wshadow -pedantic -Wl,-search_paths_first -Wl,-headerpad_max_i
nstall_names  CMakeFiles/mpags-cipher.dir/mpags-cipher.cpp.o  -o mpags-cipher
Undefined symbols for architecture x86_64:
  "transformChar(char)", referenced from:
      _main in mpags-cipher.cpp.o
ld: symbol(s) not found for architecture x86_64
clang: fatal error: linker command failed with exit code 1 (use -v to see invocation)
make[2]: *** [mpags-cipher] Error 1
make[1]: *** [CMakeFiles/mpags-cipher.dir/all] Error 2
make: *** [all] Error 2
EPSC02PN49MFVH8:mpags-cipher.build slatermw$
```

Notes

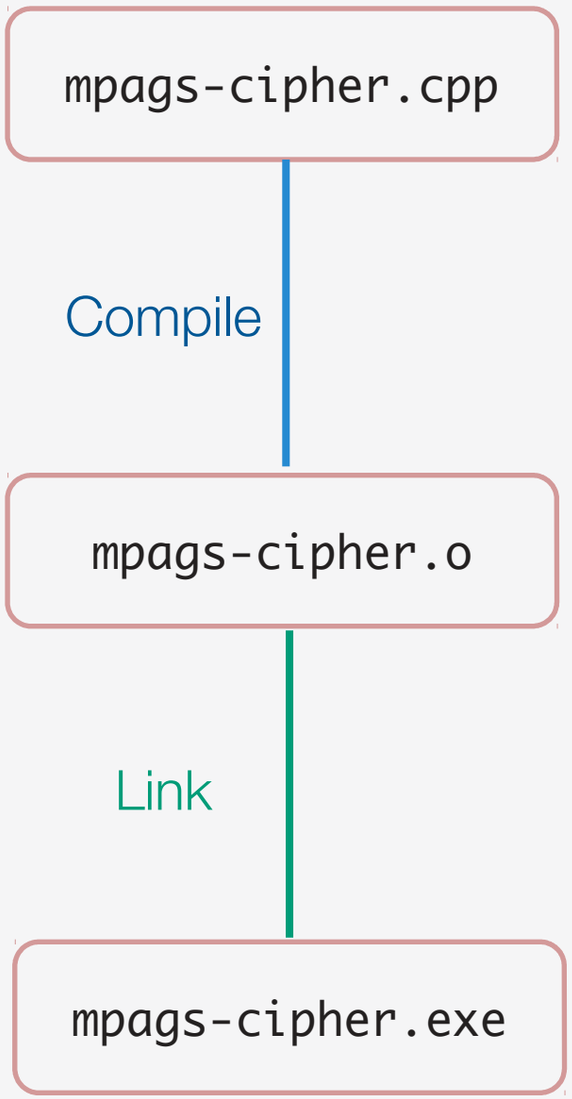
Link errors like that shown here can be much more difficult to resolve as the errors are occurring at the machine code level.

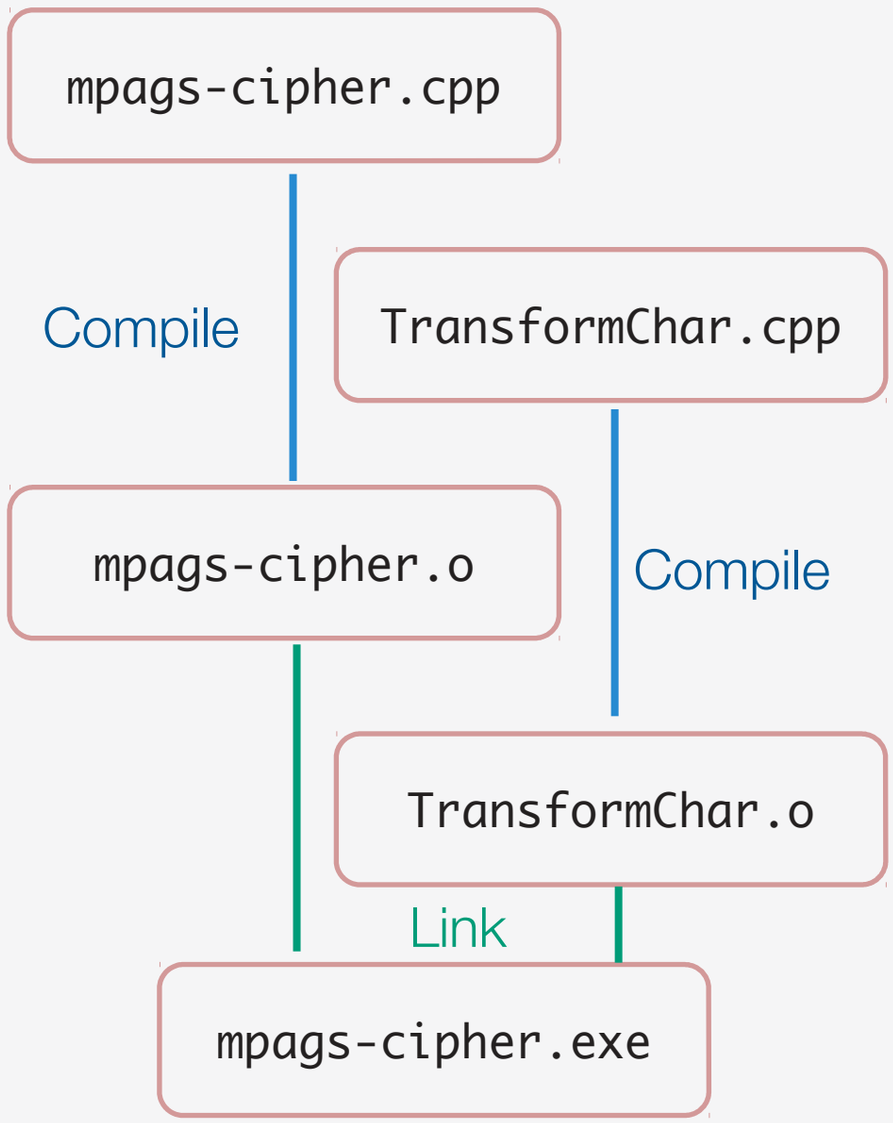
Generally, the "missing symbol" error is most common and simply means the linker has not had all needed files passed to it.

20: Splitting Up Source Code

As it's the linking step that takes care of resolving and connecting together use of functions (in this case) with the actual implementation, we don't have to have all the source code for a program in a single file.

As shown in the diagram below we can split logical blocks of code into separate source files, compile these into object files and finally link these together into the final program





Notes

Whilst this is a very simple case, we can imagine programs with hundreds, if not thousands, of functions.

Isolating functionality into separate files also helps to localise changes to that file only. That leads to cleaner commits and minimises the potential for errors.

21: Compiling transformChar Separately

To begin **modularising** `mpags-cipher`, we'll move the definition of `transformChar` into a separate file.

Create a new file named `TransformChar.cpp` **and a subdirectory called** `MPAGSCipher` **in your working copy of** `mpags-cipher`. **Move your implementation of** `transformChar` **from** `mpags-cipher.cpp` **into this file, but leave the declaration in** `mpags-cipher.cpp`.

```
EPSC02PN49MFVH8:mpags-cipher.git slatermw$ tree -C .
.
├── CMakeLists.txt
├── LICENSE
├── MPAGSCipher
│   └── TransformChar.cpp
├── README.md
└── mpags-cipher.cpp

1 directory, 5 files
EPSC02PN49MFVH8:mpags-cipher.git slatermw$
```

Notes

Modularisation is a general term used here to mean "splitting up into coherent elements".

22: Compiling transformChar Separately

If you try and recompile `mpags-cipher` at this point, you'll see that `TransformChar.cpp` isn't compiled and that you get the "missing symbol" error from before.

We need to tell CMake about the new file, so open `CMakeLists.txt` at the top level of your mpags-cipher working copy, and add `MPAGSCipher/TransformChar.cpp` to the source file list in the `add_executable()` call for `mpags-cipher`. Try recompiling - what happens?

```
# - Main CMake buildscript for mpags-cipher
# Comments in a CMake Script are lines begining with a '#'

# - Set CMake requirements then declare project
cmake_minimum_required(VERSION 3.2)
project(MPAGSCipher VERSION 0.1.0)

# - When Makefiles are generated, output all command lines by default
#   Do this to begin with so it's easy to see what compiler command/flags
#   are used. This can also be done by removing the 'set' command and
#   running make as 'make VERBOSE=1'.
set(CMAKE_VERBOSE_MAKEFILE ON)

# - Declare the build of mpags-cipher main program
add_executable(mpags-cipher
  mpags-cipher.cpp
  MPAGSCipher/TransformChar.cpp
  )

# - Don't allow C++ Compiler Vendor Extensions
set(CMAKE_CXX_EXTENSIONS OFF)

# - Use our standard set of flags
set(CMAKE_CXX_FLAGS "-Wall -Wextra -Werror -Wfatal-errors -Wshadow -pedantic")

target_compile_features(mpags-cipher
  PRIVATE cxx_auto_type cxx_range_for cxx_uniform_initialization cxx_binary_literals
  )

-uu-:---F1  CMakeLists.txt    All L1      (CMake)----------------------------------------------
Loading /usr/local/share/cmake/editors/emacs/cmake-mode.el (source)...done
```

Notes

We must list the file with its path relative to the `CMakeLists.txt` in which it is listed.

23: `#include` With Separate Sources

Each source file is compiled in isolation, so each file must have declarations available for all objects and functions it uses. In the case of `TransformChar.cpp`, it uses `std::string` and functions from `cctype`. If you didn't `#include` the headers for these in it, you're likely to see errors like that shown below when it gets compiled.

Resolve errors in the compilation of `TransformChar.cpp`, by adding appropriate includes

```
EPSC02PN49MFVH8:mpags-cipher.build slatermw$ make
/usr/local/Cellar/cmake/3.9.4_1/bin/cmake -H/Users/slatermw/mpags/mpags-cipher.git -B/Users/slat
ermw/mpags/mpags-cipher.build --check-build-system CMakeFiles/Makefile.cmake 0
/usr/local/Cellar/cmake/3.9.4_1/bin/cmake -E cmake_progress_start /Users/slatermw/mpags/mpags-ci
pher.build/CMakeFiles /Users/slatermw/mpags/mpags-cipher.build/CMakeFiles/progress.marks
/Applications/Xcode.app/Contents/Developer/usr/bin/make -f CMakeFiles/Makefile2 all
/Applications/Xcode.app/Contents/Developer/usr/bin/make -f CMakeFiles/mpags-cipher.dir/build.mak
e CMakeFiles/mpags-cipher.dir/depend
cd /Users/slatermw/mpags/mpags-cipher.build && /usr/local/Cellar/cmake/3.9.4_1/bin/cmake -E cmak
e_depends "Unix Makefiles" /Users/slatermw/mpags/mpags-cipher.git /Users/slatermw/mpags/mpags-ci
pher.git /Users/slatermw/mpags/mpags-cipher.build /Users/slatermw/mpags/mpags-cipher.build /User
s/slatermw/mpags/mpags-cipher.build/CMakeFiles/mpags-cipher.dir/DependInfo.cmake --color=
Scanning dependencies of target mpags-cipher
/Applications/Xcode.app/Contents/Developer/usr/bin/make -f CMakeFiles/mpags-cipher.dir/build.mak
e CMakeFiles/mpags-cipher.dir/build
[ 33%] Building CXX object CMakeFiles/mpags-cipher.dir/MPAGSCipher/TransformChar.cpp.o
/Applications/Xcode.app/Contents/Developer/Toolchains/XcodeDefault.xctoolchain/usr/bin/c++    -W
all -Wextra -Werror -Wfatal-errors -Wshadow -pedantic   -std=gnu++14 -o CMakeFiles/mpags-cipher.
dir/MPAGSCipher/TransformChar.cpp.o -c /Users/slatermw/mpags/mpags-cipher.git/MPAGSCipher/Transf
ormChar.cpp
/Users/slatermw/mpags/mpags-cipher.git/MPAGSCipher/TransformChar.cpp:7:1: fatal error: use of
      undeclared identifier 'std'
std::string transformChar(const char in_char)
^
1 error generated.
make[2]: *** [CMakeFiles/mpags-cipher.dir/MPAGSCipher/TransformChar.cpp.o] Error 1
make[1]: *** [CMakeFiles/mpags-cipher.dir/all] Error 2
make: *** [all] Error 2
EPSC02PN49MFVH8:mpags-cipher.build slatermw$
```

Notes

If you are seeing any other errors, or not seeing `TransformChar.cpp` compiled at all, check with us!

24: A Header for TransformChar

To use `transformChar` in `mpags-cipher.cpp` we still have to remember to add the exact declaration for it. Remembering that `#include <header>` verbatim includes the contents of the referenced file, we can instead move the declaration to a header file and `#include` that.

Create a new file named `TransformChar.hpp` under the `MPAGSCipher` subdirectory and move the declaration for `transformChar` into it from `mpags-cipher.cpp`.

```
#include <string>

//! Transliterate char to string
std::string transformChar(const char in);

-uuu:---F1  TransformChar.hpp    All L5     (C++/l Abbrev)----------------------------------
Wrote /Users/slatermw/mpags/mpags-cipher.git/MPAGSCipher/TransformChar.hpp
```

Notes

Remember that if separate source files used `transformChar`, they would each have to write out the declaration by hand. A header saves this potential source of error.

Note the inclusion of the string header. The interface of `transformChar` uses this, so we need to include it.

25: Include Guards

The C++ Standard states that there can be no more than one definition in any translation unit (source file after all `#includes` are expanded). What this means is that in general we should never `#include` a header more than once. As this is impossible to keep track of manually, we can instead use the C++ Preprocessor to only compile the code if a symbol isn't defined.

Enclose the code of `TransformChar.hpp` in `#ifndef/#define/#endif` block as shown below and try recompiling.

```
#ifndef MPAGSCIPHER_TRANSFORMCHAR_HPP
#define MPAGSCIPHER_TRANSFORMCHAR_HPP

#include <string>

//! Transliterate char to string
std::string transformChar(const char in);

#endif // MPAGSCIPHER_TRANSFORMCHAR_HPP
```

```
-uuu:---F1  TransformChar.hpp    All L9     (C++/l Abbrev)----------------------------------------
Wrote /Users/slatermw/mpags/mpags-cipher.git/MPAGSCipher/TransformChar.hpp
```

Notes

All the guard does is prevent the code being included into the same translation unit more than one.

Note that the symbol of the `#define` must be unique. `PROJECT_HEADER_HPP` is usually sufficient, though some add long random hashes as well!

26: `#include` For TransformChar.hpp

To ensure both `mpags-cipher.cpp` and `TransformChar.cpp` can see the declaration, add the line `#include "TransformChar.hpp"` to the top of both files. We use quotes rather than angle brackets as this header is internal to our project rather than external. Try recompiling - what happens?

```
// Standard Library includes
#include <fstream>
#include <iostream>
#include <string>
#include <vector>

// Our project headers
#include "TransformChar.hpp"

//! Process the command line arguments
bool processCommandLine(const std::vector<std::string>& args,
                        bool& helpRequested,
                        bool& versionRequested,
                        std::string& inputFile,
                        std::string& outputFile,
                        std::string& cipher_key,
                        bool& encrypt)
{
  // Status flag to indicate whether or not the parsing was successful
  bool processStatus(true);

  // Add a typedef that assigns another name for the given type for clarity
  typedef std::vector<std::string>::size_type size_type;
  const size_type nArgs {args.size()};

  // Process the arguments - ignore zeroth element, as we know this to be the
  // program name and don't need to worry about it
  for (size_type i {1}; i < nArgs; ++i) {

-uu-:---F1  mpags-cipher.cpp   Top L9     (C++/l Abbrev)----------------------------------------
Wrote /Users/slatermw/mpags/mpags-cipher.git/mpags-cipher.cpp
```

Notes

Though the inclusion of `TransformChar.hpp` in `TransformChar.cpp` is not strictly necessary, it ensures consistency between the declaration and definition.

The use of quotes in `#include` changes the default locations the compiler uses to search for headers.

27: Searching for Header Files

When you recompiled, you encountered the error shown below - the new `TransformChar.hpp` file wasn't found. Compilers only search for header files in a limited set of paths, so when supplying our own headers (or using any others outside the default locations) we need to inform the compiler about these paths.

We'll go back to CMake to set these paths up.

```
EPSC02PN49MFVH8:mpags-cipher.build slatermw$ make
/usr/local/Cellar/cmake/3.9.4_1/bin/cmake -H/Users/slatermw/mpags/mpags-cipher.git -B/Users/slat
ermw/mpags/mpags-cipher.build --check-build-system CMakeFiles/Makefile.cmake 0
/usr/local/Cellar/cmake/3.9.4_1/bin/cmake -E cmake_progress_start /Users/slatermw/mpags/mpags-ci
pher.build/CMakeFiles /Users/slatermw/mpags/mpags-cipher.build/CMakeFiles/progress.marks
/Applications/Xcode.app/Contents/Developer/usr/bin/make -f CMakeFiles/Makefile2 all
/Applications/Xcode.app/Contents/Developer/usr/bin/make -f CMakeFiles/mpags-cipher.dir/build.mak
e CMakeFiles/mpags-cipher.dir/depend
cd /Users/slatermw/mpags/mpags-cipher.build && /usr/local/Cellar/cmake/3.9.4_1/bin/cmake -E cmak
e_depends "Unix Makefiles" /Users/slatermw/mpags/mpags-cipher.git /Users/slatermw/mpags/mpags-ci
pher.git /Users/slatermw/mpags/mpags-cipher.build /Users/slatermw/mpags/mpags-cipher.build /User
s/slatermw/mpags/mpags-cipher.build/CMakeFiles/mpags-cipher.dir/DependInfo.cmake --color=
Scanning dependencies of target mpags-cipher
/Applications/Xcode.app/Contents/Developer/usr/bin/make -f CMakeFiles/mpags-cipher.dir/build.mak
e CMakeFiles/mpags-cipher.dir/build
[ 33%] Building CXX object CMakeFiles/mpags-cipher.dir/mpags-cipher.cpp.o
/Applications/Xcode.app/Contents/Developer/Toolchains/XcodeDefault.xctoolchain/usr/bin/c++    -W
all -Wextra -Werror -Wfatal-errors -Wshadow -pedantic   -std=gnu++14 -o CMakeFiles/mpags-cipher.
dir/mpags-cipher.cpp.o -c /Users/slatermw/mpags/mpags-cipher.git/mpags-cipher.cpp
/Users/slatermw/mpags/mpags-cipher.git/mpags-cipher.cpp:8:10: fatal error: 'TransformChar.hpp'
      file not found
#include "TransformChar.hpp"
         ^
1 error generated.
make[2]: *** [CMakeFiles/mpags-cipher.dir/mpags-cipher.cpp.o] Error 1
make[1]: *** [CMakeFiles/mpags-cipher.dir/all] Error 2
make: *** [all] Error 2
EPSC02PN49MFVH8:mpags-cipher.build slatermw$
```

Notes

We could immediately resolve this, for the GNU compiler, by changing the inclusion to `#include "MPAGSCipher/TransformChar.hpp"`.

This is due to how GNU looks up headers. Other compilers may or may not do this, so we want to be more explicit by using CMake.

28: CMake and Header Search Paths

To add search paths for the compiler, we use CMake's `target_include_directories` command, which takes a **target name and a (scoped) list to directories in which the compiler should search for headers when compiling that target's sources** (which we specified earlier in add_executable).

Add `target_include_directories` for mpags-cipher as shown below. Use PRIVATE scope, and the relative path from the `CMakeLists.txt` file to the location of `TransformChar.hpp`

```
# - Main CMake buildscript for mpags-cipher
# Comments in a CMake Script are lines begining with a '#'

# - Set CMake requirements then declare project
cmake_minimum_required(VERSION 3.2)
project(MPAGSCipher VERSION 0.1.0)

# - When Makefiles are generated, output all command lines by default
#   Do this to begin with so it's easy to see what compiler command/flags
#   are used. This can also be done by removing the 'set' command and
#   running make as 'make VERBOSE=1'.
set(CMAKE_VERBOSE_MAKEFILE ON)

# - Don't allow C++ Compiler Vendor Extensions
set(CMAKE_CXX_EXTENSIONS OFF)

# - Use our standard set of flags
set(CMAKE_CXX_FLAGS "-Wall -Wextra -Werror -Wfatal-errors -Wshadow -pedantic")

# - Declare the build of mpags-cipher main program
add_executable(mpags-cipher
  mpags-cipher.cpp
  MPAGSCipher/TransformChar.cpp
  MPAGSCipher/TransformChar.hpp
  )

target_include_directories(mpags-cipher
  PRIVATE MPAGSCipher
  )
-uu-:**-F1  CMakeLists.txt    Top L18    (CMake)----------------------------------------------
```

Notes

We've used the PRIVATE scope again, because no other build step needs to know about these directories.

We've also added the header to the list of sources in `add_executable` to ensure it's visible in IDE projects. This is not related to the header search path.

29: CMake and Header Search Paths

Once you've edited `CMakeLists.txt`, rebuild and resolve any errors.

With a successful build, you should see that an extra option has been added to the compilation commands. For GNU and Clang compilers at least, this takes the form `-I/path/to/your/working/copy/MPAGSCipher`. The `-I<dir>` option tells these compilers to add <dir> to the list of directories under which to search for headers,

```
EPSC02PN49MFVH8:mpags-cipher.build slatermw$ make
/usr/local/Cellar/cmake/3.9.4_1/bin/cmake -H/Users/slatermw/mpags/mpags-cipher.git -B/Users/slat
ermw/mpags/mpags-cipher.build --check-build-system CMakeFiles/Makefile.cmake 0
/usr/local/Cellar/cmake/3.9.4_1/bin/cmake -E cmake_progress_start /Users/slatermw/mpags/mpags-ci
pher.build/CMakeFiles /Users/slatermw/mpags/mpags-cipher.build/CMakeFiles/progress.marks
/Applications/Xcode.app/Contents/Developer/usr/bin/make -f CMakeFiles/Makefile2 all
/Applications/Xcode.app/Contents/Developer/usr/bin/make -f CMakeFiles/mpags-cipher.dir/build.mak
e CMakeFiles/mpags-cipher.dir/depend
cd /Users/slatermw/mpags/mpags-cipher.build && /usr/local/Cellar/cmake/3.9.4_1/bin/cmake -E cmak
e_depends "Unix Makefiles" /Users/slatermw/mpags/mpags-cipher.git /Users/slatermw/mpags/mpags-ci
pher.git /Users/slatermw/mpags/mpags-cipher.build /Users/slatermw/mpags/mpags-cipher.build /User
s/slatermw/mpags/mpags-cipher.build/CMakeFiles/mpags-cipher.dir/DependInfo.cmake --color=
Scanning dependencies of target mpags-cipher
/Applications/Xcode.app/Contents/Developer/usr/bin/make -f CMakeFiles/mpags-cipher.dir/build.mak
e CMakeFiles/mpags-cipher.dir/build
[ 33%] Building CXX object CMakeFiles/mpags-cipher.dir/MPAGSCipher/TransformChar.cpp.o
/Applications/Xcode.app/Contents/Developer/Toolchains/XcodeDefault.xctoolchain/usr/bin/c++   -I/
Users/slatermw/mpags/mpags-cipher.git/MPAGSCipher  -Wall -Wextra -Werror -Wfatal-errors -Wshadow
 -pedantic   -std=c++14 -o CMakeFiles/mpags-cipher.dir/MPAGSCipher/TransformChar.cpp.o -c /Users
/slatermw/mpags/mpags-cipher.git/MPAGSCipher/TransformChar.cpp
[ 66%] Linking CXX executable mpags-cipher
/usr/local/Cellar/cmake/3.9.4_1/bin/cmake -E cmake_link_script CMakeFiles/mpags-cipher.dir/link.
txt --verbose=1
/Applications/Xcode.app/Contents/Developer/Toolchains/XcodeDefault.xctoolchain/usr/bin/c++  -Wal
l -Wextra -Werror -Wfatal-errors -Wshadow -pedantic -Wl,-search_paths_first -Wl,-headerpad_max_i
nstall_names  CMakeFiles/mpags-cipher.dir/mpags-cipher.cpp.o CMakeFiles/mpags-cipher.dir/MPAGSCi
pher/TransformChar.cpp.o  -o mpags-cipher
[100%] Built target mpags-cipher
/usr/local/Cellar/cmake/3.9.4_1/bin/cmake -E cmake_progress_start /Users/slatermw/mpags/mpags-ci
pher.build/CMakeFiles 0
EPSC02PN49MFVH8:mpags-cipher.build slatermw$
```

Notes

By using CMake, we don't have to worry about how different compilers handle include paths. We just tell CMake where the compiler should search and it handles adding the appropriate flags for the compiler in use.

30: Building mpags-cipher Fast

So far, each source file is compiled separately in sequence. However, each compilation is independent with only the linker needing all the object files at the end. Build tools are aware of this, so usually allow compilation in parallel (via extra cores/threads) to speed things up

With make, simply use the -jN argument, with N being the number of parallel "jobs"

```
$ make -j2
```

```
pher.build/CMakeFiles /Users/slatermw/mpags/mpags-cipher.build/CMakeFiles/progress.marks
/Applications/Xcode.app/Contents/Developer/usr/bin/make -f CMakeFiles/Makefile2 all
/Applications/Xcode.app/Contents/Developer/usr/bin/make -f CMakeFiles/mpags-cipher.dir/build.mak
e CMakeFiles/mpags-cipher.dir/depend
cd /Users/slatermw/mpags/mpags-cipher.build && /usr/local/Cellar/cmake/3.9.4_1/bin/cmake -E cmak
e_depends "Unix Makefiles" /Users/slatermw/mpags/mpags-cipher.git /Users/slatermw/mpags/mpags-ci
pher.git /Users/slatermw/mpags/mpags-cipher.build /Users/slatermw/mpags/mpags-cipher.build /User
s/slatermw/mpags/mpags-cipher.build/CMakeFiles/mpags-cipher.dir/DependInfo.cmake --color=
/Applications/Xcode.app/Contents/Developer/usr/bin/make -f CMakeFiles/mpags-cipher.dir/build.mak
e CMakeFiles/mpags-cipher.dir/build
[ 66%] Building CXX object CMakeFiles/mpags-cipher.dir/mpags-cipher.cpp.o
[ 66%] Building CXX object CMakeFiles/mpags-cipher.dir/MPAGSCipher/TransformChar.cpp.o
/Applications/Xcode.app/Contents/Developer/Toolchains/XcodeDefault.xctoolchain/usr/bin/c++   -I/
Users/slatermw/mpags/mpags-cipher.git/MPAGSCipher  -Wall -Wextra -Werror -Wfatal-errors -Wshadow
 -pedantic   -std=c++14 -o CMakeFiles/mpags-cipher.dir/MPAGSCipher/TransformChar.cpp.o -c /Users
/slatermw/mpags/mpags-cipher.git/MPAGSCipher/TransformChar.cpp
/Applications/Xcode.app/Contents/Developer/Toolchains/XcodeDefault.xctoolchain/usr/bin/c++   -I/
Users/slatermw/mpags/mpags-cipher.git/MPAGSCipher  -Wall -Wextra -Werror -Wfatal-errors -Wshadow
 -pedantic   -std=c++14 -o CMakeFiles/mpags-cipher.dir/mpags-cipher.cpp.o -c /Users/slatermw/mpa
gs/mpags-cipher.git/mpags-cipher.cpp
[100%] Linking CXX executable mpags-cipher
/usr/local/Cellar/cmake/3.9.4_1/bin/cmake -E cmake_link_script CMakeFiles/mpags-cipher.dir/link.
txt --verbose=1
/Applications/Xcode.app/Contents/Developer/Toolchains/XcodeDefault.xctoolchain/usr/bin/c++  -Wal
l -Wextra -Werror -Wfatal-errors -Wshadow -pedantic -Wl,-search_paths_first -Wl,-headerpad_max_i
nstall_names  CMakeFiles/mpags-cipher.dir/mpags-cipher.cpp.o CMakeFiles/mpags-cipher.dir/MPAGSCi
pher/TransformChar.cpp.o  -o mpags-cipher
[100%] Built target mpags-cipher
/usr/local/Cellar/cmake/3.9.4_1/bin/cmake -E cmake_progress_start /Users/slatermw/mpags/mpags-ci
pher.build/CMakeFiles 0
EPSC02PN49MFVH8:mpags-cipher.build slatermw$
```

Notes

Generally, setting N to the number of cores is sufficient, though you should be aware of resource limitations on multiuser machines.

Other build tools have similar options, or may even enable parallel builds by default (e.g. ninja)

31: Compiling processCommandLine Separately

You now know everything required to continue out 'modularisation' of mpags-cipher by splitting the `processCommandLine` function out into a separate file just like you did with `transformChar`. **Do this now, remembering to add include guards, to update CMakeLists.txt and commit to github when you're happy!**

```
EPSC02PN49MFVH8:mpags-cipher.git slatermw$ tree -C .
.
├── CMakeLists.txt
├── LICENSE
├── MPAGSCipher
│   ├── TransformChar.cpp
│   └── TransformChar.hpp
├── README.md
└── mpags-cipher.cpp

1 directory, 6 files
EPSC02PN49MFVH8:mpags-cipher.git slatermw$
```

Notes

If you have any problems with this, just let us know!

Walkthrough Summary

This has been a very rapid introduction to CMake, but in the process we've got mpags-cipher building with all the correct flags, and begun the task of separating code into headers and sources based on functionality. Whilst this is quite a bit for such a simple project, it's been straightforward and we'll see the benefits this setup gives in future weeks.

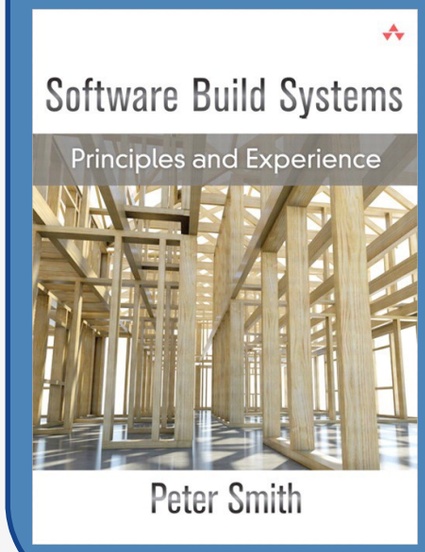
Though we've used CMake as our build tool, the same techniques and use cases apply to other systems like Autotools. The bottom line is always use a good build system!

```
|   |   ├── feature_tests.bin
|   |   ├── feature_tests.c
|   |   ├── feature_tests.cxx
|   |   ├── mpags-cipher.dir
|   |   |   ├── CXX.includecache
|   |   |   ├── DependInfo.cmake
|   |   |   ├── MPAGSCipher
|   |   |   |   └── TransformChar.cpp.o
|   |   |   ├── build.make
|   |   |   ├── cmake_clean.cmake
|   |   |   ├── depend.internal
|   |   |   ├── depend.make
|   |   |   ├── flags.make
|   |   |   ├── link.txt
|   |   |   ├── mpags-cipher.cpp.o
|   |   |   └── progress.make
|   |   └── progress.marks
|   ├── Makefile
|   ├── cmake_install.cmake
|   └── mpags-cipher
└── mpags-cipher.git
    ├── CMakeLists.txt
    ├── LICENSE
    ├── MPAGSCipher
    |   ├── TransformChar.cpp
    |   └── TransformChar.hpp
    ├── README.md
    └── mpags-cipher.cpp

12 directories, 40 files
EPSC02PN49MFVH8:mpags slatermw$
```

Further Reading

CMake Documentation



Software Build Systems Textbook