Development Practices Review

Today we are going to develop a simple command to upper case an input string. Along the way, we will review correct development practices.

Starting Off

Lets review what we know already. Given the description, we know that we are going to be producing an executable. We also know that we are responsible developers who test as we go, so we are going to want to produce a static library housing our business logic. We are going to use Catch as our testing framework, because we like Catch, since it is header only. And we are going to be using cmake to put this all together...

Step 1 - create a new project

```
yell/
    src/
    test/
        CMakeListst.txt
        unit/
        include/
             Catch.cxx
    CMakeLists.txt
```

- src is where we are going to put our amazing library.
- test is where we are going to put our main.cpp for our unit tests.
- test/unit is where our unit tests will live

Step 2 - create the main CMakeLists.txt

We know that we are going to have

- a library for yell
- an executable for yell
- a unittest suite for yell

```
cmake_minimum_required(VERSION 3.6)
project(yell)

set(CMAKE_CXX_STANDARD 11)

# add our directory
include_directories(src)
file(GLOB cpps src/*.cpp)
file(GLOB hpps src/*.hpp)
```

```
# add a library
add_library(yell ${cpps} ${hpps})

set(SOURCE_FILES main.cpp)
add_executable(yellit ${SOURCE_FILES})
target_link_libraries(yellit yell)

add_subdirectory(test)
```

Step 3 - setup test

Download Catch.hpp

`https://github.com/philsquared/Catch`

- stick single_include/catch.hpp in test/include
- create a test/main.cpp (i called it unit.cpp)

```
#define CATCH_CONFIG_MAIN
#include "catch.hpp"
```

Create a CMakeLists.txt file in test

- we are going to create a unit executable
- we are going to link against yell

```
include_directories(include)

file(GLOB ucpps unit/*.cpp)
file(GLOB uhpps unit/*.hpp)

add_executable(unit unit.cpp ${ucpps} ${uhpps})
target_link_libraries(unit yell)
```

Step 4 - Create the yell library

In src, create yell.cpp and yell.hpp

We are going to create a single function, yell, in the yell namespace.
yell should take a const reference to a string and return a string.
The returned string should convert the incoming string to upper case.
There are many ways of doing this. One such way is to use algorithm.h from the stl.

for now, we are going to implement a blank function

```
std::string yell::yell(const std::string& input) {
    std::string str{input};
    return str;
}
```

Step 5 - Writes yer Test

`In test/unit add a yelling.cpp file. Now lets write some tests:`

include “yell.hpp”

include “catch.hpp”

TEST_CASE(“Yell converts lower case string to upper case string”) {
auto test = std::string{“this is it”};
auto result = yell::yell(test);
REQUIRE(result == “THIS IS IT”);
}

TEST_CASE(“yell keeps upercase updercase”) {
auto test = std::string{“THIS IS A TEST”};
auto result = yell::yell(test);
REQUIRE(result == “THIS IS A TEST”);
}

TEST_CASE(“yell chews through mixed cases as well”) {
auto test = std::string(“This Is A tesT”);
auto result = yell::yell(test);
REQUIRE(result == “THIS IS A TEST”);

}

run the tests. They fail. Now lets make them pass:

Step 6 - make those tests pass. implement yell.

```
std::string yell::yell(const std::string& input) {
    std::string str;
    std::transform(input.begin(), input.end(),std::back_inserter(str), std::toupper);
    return str;
}
```

Step 7 - implement the executable

Here is my version:

```
#include <iostream>
#include <numeric>
#include "yell.hpp"

class argv_range {
public:
    argv_range(int argc, const char * const argv[])
            : argc_(argc), argv_(argv)
    {
    }

    const char * const *begin() const { return argv_; }
    const char * const *end() const { return argv_ + argc_; }

private:
    const int argc_;
    const char * const *argv_;
};

int main(int argc, char* argv[]) {
    if ( argc < 2) {
        std::cout << "usage: yellit [string]";
        return 0;
    }
    else {
        auto argv_ = argv_range(argc, argv);
        auto results = std::accumulate(argv_.begin()+2, argv_.end(),
                                    std::string(*(argv_.begin()+1)),
                                    [](const std::string& a, const std::string& b){
            return a + " " + b;
        });
        std::cout << yell::yell(results) << std::endl;
    }
    return 0;
}
```