# OOP for Scientific Computing Notes - SoSe 24

Igor Dimitrov

2024-04-22

# Table of contents

# Preface

This is a Quarto book.

To learn more about Quarto books visit https://quarto.org/docs/books.

# 1 Introduction

This is a book created from markdown and executable code.

See Knuth (1984) for additional discussion of literate programming.

# Part I

# CMake Tutorial

Notes from the official CMake Tutorial link

# 2 Step 1

- Introduce CMake basic syntax, commands, and variables.
- Do three exercises and create a simple project.

## 2.1 Exercise 1

- Most basic CMake project is an executable built from a **single file**. Only `CMakeLists.txt` with **three** components is required. This is our **goal** with this exercise.

> **Note**
>
> Stylistically lower case commands are preffered in CMake

### 2.1.1 The Three Basic Commands

1. Any project's top most `CMakeLists.txt` must start by specifying a minimum CMake version using using the `cmake_minimum_required()` command.
2. Afterwards we use the `project()` command to set the **project name**.
3. Finally we use the `add_executable()` to make CMake create an executable using the specified source code files

### 2.1.2 Getting Started

We will build the following c++ file that computes the square root of a number:

```cpp
// A simple program that computes the square root of a number
#include <cmath>
#include <cstdlib> // TODO 5: Remove this line
#include <iostream>
#include <string>

// TODO 11: Include TutorialConfig.h
```

```
int main(int argc, char* argv[])
{
  if (argc < 2) {
    // TODO 12: Create a print statement using Tutorial_VERSION_MAJOR
    //          and Tutorial_VERSION_MINOR
    std::cout << "Usage: " << argv[0] << " number" << std::endl;
    return 1;
  }

  // convert input to double
  // TODO 4: Replace atof(argv[1]) with std::stod(argv[1])
  const double inputValue = atof(argv[1]);

  // calculate square root
  const double outputValue = sqrt(inputValue);
  std::cout << "The square root of " << inputValue << " is " << outputValue
            << std::endl;
  return 0;
}
```

- We complete the initial 3 TODOS of the `CMakeLists.txt`:

```
# TODO 1: Set the minimum required version of CMake to be 3.10
cmake_minimum_required(VERSION 3.10)

# TODO 2: Create a project named Tutorial
project(Tutorial)

# TODO 7: Set the project version number as 1.0 in the above project command

# TODO 6: Set the variable CMAKE_CXX_STANDARD to 11
#         and the variable CMAKE_CXX_STANDARD_REQUIRED to True

# TODO 8: Use configure_file to configure and copy TutorialConfig.h.in to
#         TutorialConfig.h

# TODO 3: Add an executable called Tutorial to the project
# Hint: Be sure to specify the source file as tutorial.cxx
add_executable(Tutorial tutorial.cxx)

# TODO 9: Use target_include_directories to include ${PROJECT_BINARY_DIR}
```

### 2.1.3 Build and Run

1. create a build directory:

   ```
   mkdir build
   ```

2. change into the build directory and build with `cmake`:

   ```
   cd build
   cmake ../
   ```

3. Actually compile/link the project with

   ```
   cmake --build .
   ```

Now an executable `Tutorial` has been created and can be run with

```
./Tutorial 3.0
```

with the output

```
The square root of 3 is 1.73205
```

All good!

## 2.2 Exercise 2

# Part II

# fundamentals of c++

# 3 Basic Concepts of C++

- variables and types
- pointers and references
- control structures
- functions and templates
- classes and inheritance
- namespaces and structure

## 3.1 Variables, Temporaries, Literals

some stuff comes here...

## 3.2 Introducing New Types

- `enum`

```
enum Color = {RED, BLUE, GREEN}
```

- `struct`

## 3.3 Pointers

```
include <iostream>

int main(int argc, char const *argv[])
{
    int i = 5;
    int *p1 = &i;
    int *p2 = new int;

    std::cout << "i: " << i << std::endl
```

```
              << "*p1: " << *p1 << std::endl
              << "p1: " << p1 << std::endl
              << "&p1: " << &p1 << std::endl
              << "p2: " << p2 << std::endl
              << "*p2: " << *p2 << std::endl;
    delete p2;
    return 0;
}
```

output:

```
i: 5
*p1: 5
p1: 0x7fff8d568184
&p1: 0x7fff8d568188
p2: 0x55c014358eb0
*p2: 0
```

- release memory with `delete`.
- deleting too early -> bugs, too late -> memory leaks

## 3.4 References

References are **aliases for an existing entity**. k

```
include <iostream>

int main(int argc, const char** argv) {

    int a = 4;
    std::cout << "a: " << a <<std::endl;
    int &b = a;
    b = 5;
    std::cout << "a: " << a << std::endl
              << "b: " << b << std::endl;

    return 0;
}
```

output:

```
a: 4
a: 5
b: 5
```

## 3.5 Rvalue (double) References

Two uses:

- **range-based** `for` loops
- **move semantics**

lvalue references refer to entities, rvalue references refer to literals.

## 3.6 Const-Correctness

Marks something that can't be modified.

```
include <iostream>

int main(int argc, char const *argv[])
{
    int n = 5;
    const int j = 4;
    const int &k = n; //k can't be modified, equivalently n can't be modified over k
    n++; //but this changes n and indirectly k (because k references n)


    const int *p1 = &n; // modifiable pointer to const int
    int const *p2 = &n; // same thing
    int *const p3 = &n; // constant pointer to modifiable int

    // p1 = &j -- ok
    // *p1 = 3 -- not ok!
    // p3 = &j -- not ok
    // *p3 = 10 -- ok

    std::cout << "n: " << n << std::endl
              << "j: " << j << std::endl
              << "p1: " << p1 << std::endl
              << "p2: " << p2 << std::endl
              << "p3: " << p3 << std::endl;


    return 0;
}
```

## 3.7 Control Flow

### 3.7.1 If

```
include <iostream>

int main(int argc, char const *argv[])
{
    int i;
    std::cin >> i;

    if (i % 2 == 0) std::cout << i << " is even" << std::endl;
    else std::cout << i << " is odd" << std::endl;
    return 0;
}
```

### 3.7.2 Switch

```
include <iostream>

enum Color {RED, BLUE, GREEN};

int main(int argc, char const *argv[])
{
    int i;
    Color c = RED;

    std::cin >> i;

    switch(i) {
        case 0:
            c = RED;
            break;
        case 1 :
            c = BLUE;
            break;
        case 2 :
            c = GREEN;
            break;
        default :
            std::cout << "error: invalid color" << std::endl;
    }

    std::cout << c << std::endl;

    return 0;
}
```

# 4 Summary

In summary, this book has no content whatsoever.

# References

Knuth, Donald E. 1984. “Literate Programming.” *Comput. J.* 27 (2): 97–111. https://doi.org/10.1093/comjnl/27.2.97.