PROGRAMMING STYLE GUIDE

For the RobotPatient Simulators software development.

Date: 13-02-2023
Version: Work in progress

Template Style Guide: Google C++ Style Guide[1]

1. Summary of Style

The norms in this chapter are the most important to follow.

1.1 File Extensions

Extension	Description
C++ source files	CODE.cpp
C++ header files	CODE.hpp (.hpp is used to make the distinction between C and C++ headers)
C source files	CODE.c
C header files	CODE.h
Unit tests	CODE.cc

Table 1 extension norms

1.2 Naming Conventions

Naming conventions are only meant for the format of it. Check the chapter 2 for the best practises.

1.2.1 Constants

```
const int kDaysInAWeek = 7;
const int kAndroid8_0_0 = 24;  // Android 8.0.0
```

1.2.2 Class Data Members

```
class TableInfo {
  ...
 private:
  std::string table_name_;  // OK - underscore at end.
  static Pool<TableInfo>* pool_;  // OK.
};
```

[1] Google. (z.d.). Google C++ Style Guide. google.github.io. https://google.github.io/styleguide/cppguide.html

1.2.3 Class Format

Sections in public, protected and private order, each indented one space.

```
class MyClass : public OtherClass {
 public:      // Note the 1 space indent!
  MyClass();  // Regular 2 space indent.
  explicit MyClass(int var);
  ~MyClass() {}

  void SomeFunction();
  void SomeFunctionThatDoesNothing() {
  }

  void set_some_var(int var) { some_var_ = var; }
  int some_var() const { return some_var_; }

 private:
  bool SomeInternalFunction();

  int some_var_;
  int some_other_var_;
};
```

1.2.4 Macro Names

```
#define ROUND(x) ...
#define PI_ROUNDED 3.0
```

1.3 The #define Guard

All header files should have #define guards to prevent multiple inclusion. The format of the symbol name should be <PROJECT>_<PATH>_<FILE>_H_.

```
#ifndef FOO_BAR_BAZ_H_
#define FOO_BAR_BAZ_H_

...

#endif  // FOO_BAR_BAZ_H_
```

2. Best Practices

This part of the programming style guide is more subjective and in the grey area of judgement.

- Use gets and setters for class members access and keep the class members private.
- Follow the SOLID principle.
- Pull-request for code changes according to the model.
- Commits with standard commits formats.

Attachment 1: Pull-Request Model

[Title] : TYPE + CONTEXT

- What is been added?
- What is changed?
- What is been removed?
- Pull-Request Type: (Refactor, Bug fix, Feature).
- (Optional: why did you request this pull-request?)

Attachment 2: Commits Best Practise

Write the message in bulletpoints style with these abbreviation:

- ch: what is changed
- add: File/Functionality
- rm: removed files/functionalities