Последнее изменение: 03.02.2021

Соглашение о стиле кодирования в проекте «Преобразование векторной графики в траекторию движения робота»

Оглавление

Введение

Данные документ описывает правила кодирования в проекте «Преобразования векторной графики в траекторию движения робота» с учётом использования языка программирования C++ и средства автоматического создания документации Doxygen.

Если в соглашении не регламентируется встреченная в коде ситуация, то обратится за расширением соглашения.

Специфические правила, вводимые для данного проекта

Все описываемые в коде объекты кроме функции main должны находиться в пространстве имён srm (Svg to Robot Motions project).

Отступы

Стандартный отступ – 2 пробела.

```
int main (void) {
  return 0;
```

```
}
```

В случае сложных выражений, требующих разбивку на несколько строк, строки, начиная со второй включительно, имеют дополнительный отступ в 2 пробела.

```
std::cout <<
  “Line 1; ” <<
  “Line 2;” <<
  std::endl;
```

Имена

Имена переменных и функций оформляются в стиле camel case. Имена Переменных начинаются со строчной буквы, методов и нестатических функций с заглавной буквы, статических функций (не путать со статическими методами) с символа ‘_’, после которого идёт строчная буква.

```
void ExFunc(int arg) {
  int exVar;
}

static void _statFunc (void) {
}

void class_t::Method1(void) {
}
```

Объявление глобальных переменных начинается с “g_”, статических – с “s_” и продолжается со строчной буквы.

```
static int s_statVar;
static double g_globVar;
```

Использование глобальных переменных (и сингл-тонов) требуется сводить к минимум, а при необходимости заведения глобальных переменных (и сингл-тонов) совещаться с остальными участниками проекта.

Имена типов (структур, классов и переименованных типов) оформляются в стиле snake case и должны заканчиваться на ‘_t’. В случае переименованных типов используются ключевой слово using.

```
class class_example_t {
  …
}

struct struct_example_t {
  …
}

using color_t = unsigned long;
```

Расстановка пробелов

В заголовках функций и методов пробелы между скобками и аргументами отсутствуют. После ‘,’, разделяющей аргументы ставится пробел. При реализации ‘{‘ ставится на той же строчке, что и шапка, через пробел.

```
int Func(int arg1, int arg2);

int Func(int arg1, int arg2) {
 …
}
```

Бинарные операторы обосабливаются пробелами, унарные не обосабливаются.

```
int a = 2 + 6;
double b = -(double)a++;
```

При объявлении переменных знаки “*” и “&” тяготеют к названиям переменных (в т.ч. в заголовках функций).

```
int *a = b;
int &ref = *a;
```

В заголовках функции при участии знаков “*” и “&” в описании возвращаемого значения эти знаки обособляются пробелами.

```
int * Func1(void);
double & Func2(int a);
```

Описание структур и классов

Если у класса или структуры есть friend классы, структуры и функции, то они указываются в начале описания класса.

```
class class1_t {
  friend class class2_t;
  friend void f(int a);
}
```

В описании структур и классов спецификаторы доступа всегда указываются явно на том же уровне, на котором начинается заголовок описания типа.

```
struct struct_t {
private:
  …
protected:
  …
public:
  …
};

class class_t {
private:
  …
public:
  …
};
```

При наследовании “:” обособляется пробелами. Разрешено множественное наследование, если оно оправдано.

```
class c : public a, private b {
  …
}
```

При переопределении метода в дочернем классе обязательно использование ключевого слова override.

```
class c : public a {
  void OverridedFunc(void) override {
    …
  }
}
```

При использовании шаблонов добавляется дополнительный сдвиг в 2 пробела (аналогично с функциями и методами).

```
<template typename type = double>
  class use_type_t {
    …
  }
```

За исключением template-классов все реализации методов писать в .cpp файлах.

Перечислительный тип

Не использовать enum'ы – только enum class. Сами перечисления пишутся в snake case маленькими буквами.

```
enum class mesh_t {
  equally_spaced,
  chebyshev
};
```

О define

По возможности использовать inline функции вместо макроопределений define. Для вычислений на этапе компиляции использовать ключевое слово constexpr.

Если всё-таки требуется define он оформляется в snake case большими буквами.

Обязательные к использованию опциональные возможности языка

Использовать override для перегружаемых функций и noexcept для функции, не бросающих исключения.

Запрещённые возможности языка

Не используется ключевое слово goto.

Не используется ключевое слово using для всего пространства имён.

```
using namespace std;  // нельзя
```

Не используется ключевое слово typedef.

```
typedef double real_t;  // нельзя
```

Не используются аргументы, которые являются неконстантными ссылками.

```
int Func1(big_structure_t &bs);  // нельзя
int Func2(const big_structure_t &bs);  // можно
```

Не используется авто-типирование для элементарных типов, кроме конструкций foreach.

```
auto a = 1;  // нельзя
```

Исключение повторного включения

В файлах заголовков, в которых требуется исключение повторного включения, применяется как pragma once, так и «классическое» исключение повторного включения через ifndef.

header.h

```
#pragma once

#ifndef __HEADER_H_INCLUDED
#define __HEADER_H_INCLUDED
…
#endif /* __HEADER_H_INCLUDED */
```

Комментирование

Комментарии вида // разрешены только в случае строчных комментариев, если комментарий на несколько строк, то должны использоваться комментарии вида /**/.

В начале файле

```
/**
 * @file
 * @brief <краткое описание>
 * @authors <автор(ка)> {[, <автор(ка)>]}
 * @date <дата последнего изменения в формате ДД.ММ.ГГГГ>
 *
 * <подробное описание>
 */
```

Примеры краткого описания: “Main source program file”, “Animation class description header file”.

Описание класса или структуры

```
/**
 * @brief <краткое описание>
 * [@warning <важные уточнения о типе>]
 * [@see <связанные типы>]
 *
 * <подробное описание>
 */
```

Описание функции и метода

Описываются все параметры. Возвращаемое значение описывается всегда, когда оно есть.

```
/**
 * <описание>
 * {[@param[in/out] <имя> <описание аргумента>]}
 * [@throw <тип исключения> <описание>]
 * [@return <описание>]
 * [@see <связанные функции/методы>]
 */
```

Описание полей и глобальных переменных

```
int x;  ///< <описание>
```

Объединение методов в модули

```
/**
 * @defgroup <идентификатор> <название модуля>
 * @brief <краткое описание>
 *
 * <подробное описание>
 * @{
 */
…
/**@}*/
```