Стиль кода

Автор: Озеров Данил Алексеевич, ИА-032,
email: danilozerovmail@gmail.com,
github: @q2p

Февраль 2022

1 Введение

При разработке личных проектов я стараюсь использовать одинаковый стиль кода вне зависимости от языка. Я решил привести примеры на различных языках, так как каждый из них имеет свои ньюансы при применении различных стилистических приёмов.

2 Табуляция

Для отступов внутри блоков кода используются табы. Размер табов в редакторе - два пробела.

Rust:

```rust
fn main() {
  println!("Hello")
}
```

Если же отступы нужны внутри строки, то внутри строк используются пробелы.

Rust:

```rust
const fn rotate_y_matrix(angle: f32) -> &'static [f32; 9] {
  return &[
     angle.cos(), 0, angle.sin(),
                0, 1,           0,
    -angle.sin(), 0, angle.cos(),
  ];
}
```

3 Именование

Переменные, функции, модули, файлы именуются используя `snake_case`. Константы именуются используя `CONSTANT_CASE`.

C:

```
enum {
  MEANING_OF_LIFE = 42,
};

uint32_t get_random_number() {
  return 4; // chosen by fair dice roll.
            // guaranteed to be random.
}
```

4 Пробелы рядом со специальныи символами

Знаки арифметических операций обрамляются пробелами с обеих сторон.

Rust:

```
let r = (a + b) * c;
```

Скобки при вызове функций не обрамлсяются пробелом слева. Операции взятия указателя ставятся рядом с переменной. После запятой ставится пробел.

Rust:

```
call_func(a, &b);
```

C-подобные языки позволяют убрать фигурные скобки при использовании блока кода, если блок состоит только из одной строки. Я скобки оставляю.

Rust:

```
for i in list {
  i.do_stuff();
}
```

Скобки открывающие блок не переносятся на новую строку.

Rust:

```
if a {
  println!("a")
} else {
  println!("нea")
}
```

При объявлении указателя, звёздочка ставится рядом с типом.

C:

```
void func(uint32_t* pointer);
```

5 Лямбда выражения

Если в языке есть нативная конструкция, то лучше использовать её вместо функций с лямбдами:

Kotlin:

```
for(i in list) {
  println(i)
}
```

Вместо:

Kotlin:

```
list.forEach {
  println(it)
}
```

6 Инициализация переменных

Переменные объявляются только тогда когда становятся необходимыми и живут только в том блоке, в котором они нужны.

Rust:

```
fn do_the(funny: u64) {
  for i in 0..funny {
    let temp = (i as f32).sqrt();
    println!("{}", temp);
  }
}
```

Вместо:

Rust:

```
fn do_the(funny: u64) {
  let mut temp;
  for i in 0..funny {
    temp = (i as f32).sqrt();
    println!("{}", temp);
  }
}
```

7 Неизменяемые переменные

Если язык позволяет, то неизменяемые переменные помечаются как таковые.

TypeScript:

```
function power_of_4(a: number): number {
  const power_of_2 = a * a;
  return power_of_2 * power_of_2;
}
```

8 Константы в C и C++

В C и C++ вместо макросов или переменных с модификатором `const` используются энумераторы, так как они вычисляются во время компиляции, их нельзя изменить вызвав UB и они не подвержены недостаткам макросов.

C:

```
enum {
  MATH_PI         = 3.1415927,
  MATH_E          = 2.7182818,
  MATH_PI_TIMES_E = MATH_PI * MATH_E,
};
```

9 Неоднозначный размер типов

Если язык по умолчанию не имеет жёстких ограничений по размеру типов [2], но такие ограничения можно включить, то используются типы с однозначными размерами.

C:

```
// Может иметь 32, 36 или 48 бит точности
// в зависимости от архитектуры и компилятора
long a;
// Всегда имеет 32 бита точности
int32_t b;
```

10 #ifndef guard

Для избежания циклических включений заголовочных файлов в C и C++ используется не стандартный `#pragma once`

C:

```
#pragma once

void my_function(in8_t a);
```

11 Вермя жизни

Если есть возможность использовать только чистые функции или синглтон, то лучше этим воспользоваться.

Rust:

```
fn make_request(db_connection: DBConnection) -> Option<String> {
  return db_connection.make_request().as_ref();
}
```

Вместо того, чтобы использовать излишние билдеры или фабрики.

Rust:

```
struct DBRequest {
  db_connection: DBConnection,
  result: Option<String>,
}
impl DBRequest {
  fn new(db_connection: DBConnection) -> DBRequest {
    return DBRequest {
      db_connection: db_connection,
      result: None,
    };
  }
  fn make_request(&mut self) {
    self.result = self.db_connection.make_request();
  }
  fn get_response(&self) -> Option<&str> {
    return self.result.as_ref();
  }
}
```

Список литературы

[1] ISO/IEC 9899:2018 - C, §6.7.3 [Электронный ресурс] URL: http://www.open-std.org/jtc1/sc22/wg14/www/docs/n2310.pdf (дата обращения: 06.02.2022).

[2] Joe Nelson - C Portability Lessons from Weird Machines [Электронный ресурс] URL: https://begriffs.com/posts/2018-11-15-c-portability.html (дата обращения: 06.02.2022).

[3] Kotlin Standard Library Documentation [Электронный ресурс] URL: https://kotlinlang.org/api/latest/jvm/stdlib (дата обращения: 06.02.2022).

[4] G. M. Poore The minted package: Highlighted source code in LaTeX [Электронный ресурс] URL: http://tug.ctan.org/tex-archive/macros/latex/contrib/minted/minted.pdf (дата обращения: 06.02.2022).