Правила оформления кода

Часто программисты забывают что код программ пишется в первую очередь для людей. Даже если вы пишите программу в одиночку, то посмотрев на нее через месяц, вы не вспомните почему написали тот или иной кусок кода и за что он отвечает.

Хороший код должен, в первую очередь, очень ясно выражать намерения. К сожалению “быстрые и грязные” способы разработки бьют в первую очередь по понимаемости кода. Улучшение кода осознанно откладывается до лучших времен, когда будет пауза чтобы провести рефакторинг. Те самые лучшие времена никогда не наступают, а код начинают читать и дорабатывать сразу же после попадания в source control.

Даже если вы полностью довольны свои кодом (в большинстве случаев программисты свои кодом довольны), то подумайте о том как будет ваш код читать другой человек (в большинстве случаев программисты недовольны чужим кодом).

— «Как важно писать хороший код» http://habrahabr.ru/post/206294/

Общие правила оформления кода на основе Qt Coding Style.

Отступы

- Для обозначения отступа используйте 4 пробела подряд
- Используйте проблелы, а не табуляцию!
    - Для **Visual Studio**: СЕРВИС > Параметры > Текстовый редактор > C/C++ > Табуляция «Вставлять пробелы».

Объявление переменных

- Объявляйте по одной переменной в строке.
- Избегайте, если это возможно, коротких и запутанных названий переменных (Например: “a”, “rbarr”, “nughdeget”).
- Односимвольные имена переменных подходят только для итераторов циклов, небольшого локального контекста и временных переменных. В остальных случаях имя переменной должно отражать ее назначение.
- Заводите переменные только по мере необходимости.

```
1.          // Так не правильно
2.          int a, b;
3.          char *c, *d;
4.
5.          // А так - правильно
6.          int height;
7.          int width;
```

```
char *nameOfThis;
char *nameOfThat;
```

- Функции и переменные должны именоваться с прописной буквы, а если имя переменной или функции состоит из нескольких слов, то первое слово должно начинаться с прописной буквы, остальные – со строчных.
- Избегайте аббревиатур

```
// Не правильно
short Cntr;
char ITEM_DELIM = '\t';

// Правильно
short counter;
char itemDelimiter = '\t';
```

Пробелы

- Используйте пустые строки для логической группировки операторов, где это возможно.
- Всегда используйте одну пустую строку в качестве разделителя
- Всегда используйте один пробел перед фигурной скобкой

```
// Не правильно
if(foo){
}

// Правильно
if (foo) {
}
```

- Всегда ставьте один пробел после ‘*’ или ‘&’, если они стоят перед описанием типов. Но никогда не ставьте пробелы после ‘*’ или ‘&’ и именем переменной.

```
char *x;
const string &myString;
const char * const y = "hello";
```

- Бинарные операции отделяются пробелами с 2-х строн.
- После преобразования типов не ставьте пробелов.
- Избегайте проеобразования типов в стиле C.

```
// Неправильно
char* blockOfMemory = (char* ) malloc(data.size());

// Правильно
char *blockOfMemory =
      reinterpret_cast<char*>(malloc(data.size()));
```

Фигурные скобки

- Возьмите за основу расстановку открывающих фигурных скобок на одной строке с выражением, которому они предшествуют

```
// Неправильно
if (codec)
{
}

// Правильно
if (codec) {
}
```

- Исключение: Тело функции и описание класса всегда открывается фигурной скобкой, стоящей на новой строке

```
static void foo(int g)
{
}

class Moo
{
};
```

- Используйте фигурные скобки в условиях, если тело условия в размере превышает одну линию, или тело условия достаточное сложное и выделение скобками действительно необходимо

```
// Неправильно
if (address.isEmpty()) {
    return false;
}

for (int i = 0; i < 10; ++i) {
    printf("%i", i);
}

// Правильно
if (address.isEmpty())
    return false;

for (int i = 0; i < 10; ++i)
    printf("%i", i);
```

- **Исключение 1**: Используйте скобки, если родительское выражение состоит из нескольких строк / оберток

```
// Правильно
if (address.isEmpty() || !isValid()
    || !codec) {
    return false;
}
```

- **Исключение 2**: Используйте фигурные скобки, когда тела ветвлений if-then-else занимают несколько строчек

```
// Неправильно
if (address.isEmpty())
    return false;
else {
    printf("%s", qPrintable(address));
    ++it;
}

// Правильно
if (address.isEmpty()) {
    return false;
} else {
    printf("%s", qPrintable(address));
    ++it;
}

// Неправильно
if (a)
    if (b)
        ...
    else
        ...

// Правильно
if (a) {
    if (b)
        ...
    else
        ...
}
```

- Используйте фигурные скобки для обозначения пустого тела условия

```
// Неправильно
while (a);

// Правильно
while (a) {}
```

Круглые скобки

- Используйте круглые скобки для группировки выражений:

```
// Неправильно
if (a && b || c)

// Правильно
if ((a && b) || c)

// Неправильно
a + b & c

// Правильно
(a + b) & c
```

Использование конструкции switch

- Операторы case должны быть в одном столбце со switch
- Каждый оператор case должен иметь закрывающий break (или return) или комментарий, котрой предполагает намеренное отсутсвие break & return.

```
switch (myEnum) {
case Value1:
    doSomething();
    break;
case Value2:
    doSomethingElse();
    // проходим дальше
default:
    defaultHandling();
    break;
}
```

Разрыв строк

- Длина строки кода не должна превышать 100 символов. Если надо – используйте разрыв строки.
- Запятые помещаются в конец разорванной линии; операторы помещаются в начало новой строки. В зависимости от используемой вами IDE, оператор на конце разорванной строки можно проглядеть.

```
// Правильно
if (longExpression
    + otherLongExpression
    + otherOtherLongExpression) {
}

```

```
7.
8.          // Неправильно
9.          if (longExpression +
10.              otherLongExpression +
11.              otherOtherLongExpression) {
12.          }
```

Комментарии

Хотя их и утомительно писать, комментарии жизненно необходимы для читаемости вашего кода. Следующие правила описывают что вам следует комментировать и где. Но помните: хотя комментарии и очень важны, наилучший код — самодокументируемый. Гораздо лучше давать разумные имена типам и переменным, чем использовать запутанные имена, которые затем придется объяснять в комментариях.

При написании комментариев пишите для своей публики — следующего участника проекта (преподавателя), которому нужно будет разобраться в вашем коде. Будьте щедры — следующим участником можете быть вы!

- Вы можете использовать синтаксис либо //, либо /* */; хотя // гораздо более употребим. Будьте последовательны в том, как вы комментируете и какой стиль используется где.
- Начинайте каждый **файл** с объявления авторских прав, с последующим описанием содержимого файла.
- Не дублируйте комментарии сразу и в .cpp, и в .h. Продублированные комментарии недопустимы.
- Каждому **объявлению функции** должны предшествовать комментарии объясняющие что функция делает и как ее использовать. Эти комментарии должны быть скорее описательные ("Открывает файл"), чем императивные ("Открыть файл"); комментарии описывают функцию, а не говорят функции что делать.
- Как правило само имя переменной должно быть достаточно описательным чтобы объяснить для чего она используется. Однако, все **глобальные переменные** следует снабжать комментариями описывающими чем они являются и для чего используются.
- Неочевидные **строки** должны иметь комментарий в конце. Эти комментарии следует отделять от кода 2 пробелами.
- вы никогда не должны описывать сам код. Допускается, что персона читающая код знает C++ лучше, чем вы, даже если он или она не знает что вы пытаетесь сделать.
- Обратите внимание на пунктуацию, орфографию и грамматику, легче читать хорошо написанные комментарии, чем плохо написанные.