Форматирование кода

Практически всегда, над созданием одной программы работают несколько программистов. При этом сопровождать программу в дальнейшем может уже другая группа программистов. Становится очевидной необходимость создания единого стандарта кодирования. Если программисты будут придерживаться этих стандартов при написании кода – у всех он будет выглядеть одинаково, что в разы облегчит работу над коллективными проектами. Зачастую контора сама вырабатывает свой собственный стандарт кодирования (coding convention), но есть распространенные стандарты. К ним относятся, например:

Google C++ Style Guide

Coding Standards IBM

Linux kernel coding style

Если посмотреть статистику использования стандартов форматирования кода, становится очевидным, что большинству программистов есть еще над чем работать и к чему стремиться. Ведь больше половины из опрошенных, либо используют свой собственный стиль кодирования, либо вообще не знают, зачем стандарты форматирования и соглашения о кодировании созданы.

Основными задачами всех стандартов и соглашений о кодировании является способствование:

-написанию легко-читаемого кода, понятного для всех;

-написанию безопасного кода (ведь эти стандарты были созданы программистами-практиками, которые знают, какие ошибки может повлечь за собой неправильное оформление кода);

-единообразного кода (у всех структура кода выглядит схоже).

Просмотрев несколько стандартов и соглашений, можно увидеть, что в основном внимание уделяется следующим пунктам:

Имена переменных, констант, функций, классов

Главное при объявлении переменной (функции, класса и т.д.) – дать осмысленное имя, как можно ближе к контексту использования. Допустим, в вашей программе определено больше 20-ти переменных, и всем дано имя в виде буквы алфавита. Согласитесь, даже вам, создателю этой программы, будет тяжело работать с этими переменными. Не говоря уже о том, что нужно так же помнить какие данные хранит каждая из них. Гораздо легче работать с переменными, имеющими обоснованные имена:

```
age – возраст;
```

```
number – номер;
amount – количество;
name – имя.
```

Желательно имена писать не английским транслитом, а английскими словами.

не `vozrast` – а `age`;

не `kolichestvo` – а `amount`.

Если вы не знаете перевода, воспользуйтесь одним из множества онлайн переводчиков. Заодно пополните ваш словарный запас. Правила, согласно которым даются имена переменным, можно почитать в нашей статье Типы данных, переменные и константы в C++.

Константам рекомендуется присваивать имена либо состоящие из букв верхнего регистра (HOURS_IN_DAY, SIZE) либо каждое новое слово с большой буквы, как Google C++ Style Guide (kHoursInDay). Говоря о константах, их советуют использовать везде, где только можно. Не объявляйте переменные хранящие количество дней в неделе и месяцы в году – объявляйте константные переменные в таких случаях. Относительно функций – если функция не изменяет аргумент, передаваемый по ссылке или по указателю, то аргумент должен быть константой.

Тогда как для имен переменных используются существительные, для имен функций необходимо использовать глаголы или глагол + существительное. Так правильней потому, что функция выполняет определенное действие:

```
printData(); – печать данных
enterName(); – ввод имени
showStr(); – показать строку
```

В имени класса первая буква должна быть заглавной:

```
class Employee
class Point
```

Если имя состоит из нескольких слов, написать его можно разными способами.

– каждое новое слово с большой буквы (верблюжий регистр): boxWithApple, amountBoxesForSale

– использовать нижний прочерк между словами: box_with_apple, amount_boxes_for_sale

Не бойтесь давать сложные имена. В среде Microsoft Visual Studio обращаясь, к объявленной переменной (начиная вводить ее имя), вам будет предложено выбрать имя из выпадающего списка. Давать слишком

длинные имена тоже не стоит. Вполне реально отразить суть переменной в нескольких словах.

Сразу хочу рассказать о «венгерской нотации» – соглашении об именовании переменных и других идентификаторов в коде программ. Суть «венгерской нотации» в том, что имя переменной (функции, массива и т.д.), начинается с префикса, состоящего из одной или нескольких букв. Приведу несколько примеров объявления имен идентификаторов с префиксами:

```
int iAmount или nAmount,
bool bChoice,
int aNumbers (a говорит о том, что aNumbers – это массив),
string sName (строка),
int* pArr (от слова pointer – указатель)
```

Когда мы встретим такое имя в коде, то его префикс будет говорить нам о том, что это за идентификатор и какие данные он хранит.

Фигурные скобки

В некоторых соглашениях и стандартах рекомендовано использовать фигурные скобки в блоках if, else, while, do, for даже если они содержат всего одну строку либо не содержат ничего. Например:

```
for(int example = 0; example < 10; example ++)
{
   cout << example << end;
}
```

Каждую фигурную скобку желательно располагать в отдельной строке. Так очень легко проследить, где блок начинается и где заканчивается.

Пробелы в строке и отступы между строками

Рекомендуется не использовать пробелы в конце строки перед оператором точка с запятой.

При использовании оператора присвоения значения пробелы необходимы с обоих сторон от этого оператора:

```
variable = 0;
```

Это же касается и операторов используемых в арифметических выражениях:

```
variable = a + b – c;
variable=a+b–c; // слитно выглядит так
```

Применяя унарные операторы, пробелы не нужны:

```
variable = -4;
variable++;
```

Многие используют табуляцию в строках, чтобы визуально выделить некоторые блоки кода. Так например выглядит фрагмент кода с вложенными циклами с использованием табуляции:

```
int main()
{
	for (int i = 0; i < 12; i++)
	{
		for (int j = 0; j < 12; j++)
		{
			cout << '@';
		}
		cout << endl;
	}
	return 0;
}
```

и так без нее:

```
int main()
{
for (int i = 0; i < 12; i++)
{
for (int j = 0; j < 12; j++)
{
cout << '@';
}
cout << endl;
}
return 0;
}
```

По поводу отступов между строками кода, основной принцип сводится к тому, чтобы минимизировать их количество. Логика в том, что чем больше код, который умещается на одном экране, тем легче проследить и понять работу программы. Поэтому не используйте больше одной-двух пустых строк между блоками кода или функциями.

Комментирование кода

К сожалению, многие не любят комментировать код, хотя комментарии играют важную роль в поддержании читаемости кода на высоком уровне. Как написано в Google C++ Style Guide «Комментарии важны, но лучше когда код сам говорит за себя. Давать осмысленные имена переменным гораздо лучше, чем давать непонятные названия и затем раскрывать их суть в комментариях»

Оставлять комментарии в коде можно либо используя двойной слэш // (комментирование одной строки), либо /* комментарий */ . Вторым способом можно оформить многострочный комментарий. Но предпочтение авторы некоторых стандартов предлагают отдавать все же комментированию с использованием двойного слэша // .

Каждая функция должна иметь комментарии непосредственно перед ней, которые описывают то, что делает эта функция и как её использовать. Например, перед функцией можно написать: // Открывает файл, или // печатает данные

Так же желательно, чтобы каждое определение класса имело сопроводительный комментарий, описывающий, что это такое и как его следует использовать.

В стандартах форматирования кода можно почерпнуть еще много чего полезного для формирования собственного хорошего стиля программирования. Всего в данной статье не перечислишь. Поэтому постарайтесь найти время и почитать некоторые первоисточники, ссылки на которые были приведены выше в статье. Рекомендую также почитать о системе документирования исходных текстов Doxygen.

В среде разработки ***Microsoft Visual Studio 2013 Express*** *(возможно, и в более ранних версиях)* есть “спасательная комбинация клавиш” Ctrl+K затем Ctrl+F, нажав которую осуществится форматирование выделенного исходного кода. То есть если выделить неотформатированный код и нажать эту комбинацию клавиш – автоматически добавятся и необходимые пробелы, и отступы, и скобки перенесутся в отдельные строки. В общем станет код выглядеть, как для людей.

Применяя на практике форматирование кода, описанное в стандартах и соглашениях о кодировании, вы добьетесь и достойной легкой читаемости кода, и его безопасности.