Правила оформления кода

версия 3 от 25.11.2017

Используйте разумные имена для переменных и функций

Программа должна быть хорошо понятна человеку при чтении. Если при чтении программы приходится понимать назначение переменных и функций по тому, как они используются, то читать код становится гораздо сложнее. Неудачно выбранные имена могут привести к тому, что смысл программы может быть неправильно понят. Выбирайте такие имена, которые бы объясняли смысл переменных и функций, тогда код станет гораздо понятнее, и не потребуется писать множество комментариев.

Используйте camelCase

При написании составных слов, например в именах переменных, пишите их слитно без пробелов, при этом каждое новое слово пишется с большой буквы.

Для имён переменных, функций и методов классов используйте lowerCamelCase – первое слово пишется с маленькой буквы:

- переменная:
```
int myVariable;
```
- функция:
```
int myFunction(int firstParameter, int secondParameter);
```

Для имен классов (и других типов) используйте UpperCamelCase – первое слово пишется с большой буквы:

- класс:
```
class MyClass;
```
- перечисления:
```
enum class TrafficLight = {green, yellow, red};
```

Поля класса начинайте с нижнего подчёркивания:

```
class MyClass
{
	int _someData;
}
```

Не дублируйте код

Если в программе есть одинаковые выражения или фрагмента кода, вынесите этот код в отдельную функцию. Верным признаком необходимости создания новой функции является желание скопировать фрагмент кода из одного места программы в другое. В таком случае сразу перенесите этот фрагмент в отдельную функцию.

Если фрагменты кода похожи, но не идентичны, подумайте, не получится ли и их вынести в одну функцию, возможно добавив параметры или условия.

Не используйте «магические константы»

Использование неименованных «магических» констант в коде нежелательно:

- при чтении кода может быть не понятно, что это за число, и почему оно именно такое;
- чаще всего одно и то же число потребуется написать в нескольких местах кода. Если его придётся изменять, можно пропустить одно из использований, что приведёт к ошибке.

Если в коде нужно использовать константу, дайте ей имя, используя const.

Расставляйте пробелы вокруг бинарных операторов

Это улучшает читаемость формул.

Правильно	Неправильно
`z = ((x + y) * z);`	`z=((x+y)*z);` `z = ( ( x + y ) * z );`

Всегда выделяйте блоки условных операторов и циклов скобками

В любой блок условия или цикла может захотеться добавить новое выражение. При этом можно забыть добавить скобки, особенно после программирования на Python, в котором они не нужны. Лучше сразу добавить скобки, чтобы потом не было с этим проблем.

Правильно	Неправильно

`if(x > 0)`
`{`
`  y = sqrt(x);`
`}`
	`if(x > 0)`
	`  y = sqrt(x);`

Расставляйте скобки одинаково

Выберите и используйте для себя один из стилей расстановки скобок (https://ru.wikipedia.org/wiki/Отступ_(программирование)).

Это улучшает читаемость структуры программы. В Visual Studio есть команда меню и сочетание клавиш, которые автоматически форматируют документ (или текущее выделение). Достаточно использовать эту команду.

Правильно – 1 Стиль Олмана	Правильно – 2 Стиль K&R (Kernigan & Richie)	Неправильно

`if(x > 0)`
`{`
`  y = sqrt(x);`
`}`
`else`
`{`
`  cout << "Error";`
`}`
	`if(x > 0) {`
	`  y = sqrt(x);`
	`} else {`
	`  cout << "Error";`
	`}`
		`if(x > 0) { y = sqrt(x); }`
		`else`
		`  {`
		`  cout << "Error";`
		`}`

Не делайте строки слишком длинными

Строка программы должна помещаться на экране. Обычно рекомендуют ограничить максимальную ширину строки в 80 символов.

Если определение или вызов функции получается слишком широким поместите по одному параметру на каждой строке.

```
ReturnType LongClassName::ReallyLongFunctionName(Type par_name1,
                                                 Type par_name2,
                                                 Type par_name3)
{
    callAnotherLongFunction(par_name1 + par_name2 + par_name3,
                            par_name1 + par_name2,
                            par_name1);
    ...
}
```

Длинные математические выражения разбивайте на несколько строк, разбивая по границам логических блоков выражения.

Объявляйте переменные непосредственно перед использованием

В «старых» стандартах языка С переменные можно было объявлять только в начале функции. Однако опыт показал, что это плохо:

- чтобы добавить новую переменную нужно (далеко) бегать по тексту, это отвлекает;
- когда читаешь код, использующий переменную, её определения может быть уже не видно, и не понятно, какого она типа и какое начальное значение;
- можно нечаянно повторно использовать переменную с другой целью и при этом в ней окажется какое-то неожиданное значение.

Лучше объявлять переменные как можно ближе к тому месту, в котором они используются.

Правильно

```
int _tmain(int argc, _TCHAR* argv[])
{
	double x = 0;

	cout << "Hello! Please enter x : ";
	cin >> x;

	// Тут много другого кода

	double y = sqrt(x);
	cout << "sqrt(x) is " << y;

	return 0;
}
```

Неправильно

```
int _tmain(int argc, _TCHAR* argv[])
{
	double x = 0, y = 42.054;

	cout << "Hello! Please enter x : ";
	cin >> x;

	// Тут много другого кода

	y = sqrt(x);
	cout << "sqrt(x) is " << y;

	return 0;
}
```

Обязательно указывайте начальное значение для переменных

Значение неинициализированной переменной может быть любым. Использование (чтение) такого значения приведёт к недетерминированной работе программы, а в некоторых случаях является неопределённым поведением. Чтобы избежать проблем всегда инициализируйте переменные прямо в момент их создания.

Правильно

```
int _tmain(int argc, _TCHAR* argv[])
{
	int sum = 0;

	for (int i = 0; i < 10; i++)
	{
		sum += i;
	}
	cout << "sum is " << sum;

	return 0;
}
```

Неправильно

```
int _tmain(int argc, _TCHAR* argv[])
{
	int i, sum;

	sum = 0;

	for (i = 0; i < 10; i++)
	{
		sum += i;
	}

	cout << "sum is " << sum;

	return 0;
}
```

Возврат значений из функции

Предпочитайте возврат данных из функций по значению. Как правило, возврат ссылки необходим только при перегрузке операторов.

Возврат ссылок может привести к ситуации, когда эта ссылка указывает на уже не существующий объект. Возвращайте ссылку, только если уверены, что объект, на который указывает эта ссылка, будет существовать «достаточно» долго.

Правила передачи параметров в функции

Эти правила направлены на то, чтобы снизить число ошибок и повысить эффективность программы.

1. Аргументы базовых типов передавайте по значению, кроме (нежелательной) ситуации, когда функция должна изменять свои параметры.
2. Аргументы «сложных» типов по возможности передавайте как ссылку на константу. Разумными исключениями из этого правила могут быть, например, следующие ситуации:
    a) внутри функции всё равно нужно сделать копию аргумента для последующей обработки; в этом случае проще сразу передать аргумент по значению;
    b) функции необходимо изменить свои аргументы.
3. Избегайте функций, которые изменяют свои аргументы. Если не удаётся этого избежать, давайте такой функции или аргументу такое имя, из которого было бы очевидно, что аргумент изменится. Для передачи аргумента используйте ссылку.

Используйте const везде, где это возможно

Если значение переменной не должно меняться, обозначайте её с помощью ключевого слова const. Тогда компилятор не даст по ошибке изменить это значение. Используйте const как для переменных, так и для параметров функций.

В директиве include используйте угловые скобки для файлов стандартных библиотек, а кавычки – для своих файлов

Имя файла в директиве #include может быть указано или в угловых скобках (<file>) или в двойных кавычках ("file.h"). Первый вариант предназначается для стандартных библиотек (входящих в поставку компилятора), второй – для заголовочных файлов проекта и дополнительных библиотек. Разница состоит в том, где (по каким путям) компилятор будет искать библиотеки. Соответствующие наборы путей указываются двумя разными опциями компилятора.

Избегайте использования директивы препроцессора #define

Для создания именованных констант используйте const или enum.

Вместо макросов с аргументами используйте обычные функции или шаблонные функции, если они должны работать с данными разных типов. Если вызов функции кажется медленной процедурой, используйте ключевое слово inline, чтобы рекомендовать компилятору вставлять код функции в каждую точку её использования (как сделает препроцессор) вместо вызова.

Для запрета повторной обработки заголовочных файлов используйте директиву

```
#pragma once
```

Объявления должны находиться в заголовочных файлах (.h), а определения в файлах исходного кода (.cpp)

Для того чтобы классы, функции и глобальные переменные были доступны из нескольких файлов, их объявления следует помещать в заголовочные файлы (с расширением .h).

Не помещайте определения (код) методов классов в заголовочные файлы. При работе над большими проектами изменение кода в заголовочном файле приведёт к большему времени перекомпиляции программы, чем если бы код находился в файле исходного кода (.cpp).

Не помещайте объявления переменных и код (определения) функций в заголовочные файлы, так как ваш проект не скомпилируется, если этот заголовочный файл подключить из нескольких файлов исходного кода.