Содержание

руководство по стилю в C++

Основано на руководстве от google, но не идентично ему.

Заголовочные файлы

Желательно, чтобы каждый `.cc` файл исходного кода имел парный `.h` заголовочный файл. Также есть известные исключения из этого правила, такие как юниттесты или небольшие `.cc` файлы, содержащие только функцию `main()`.

Независимые заголовочные файлы

Заголовочные файлы должны быть самодостаточными (в плане компиляции): иметь блокировку от повторного включения и включать все необходимые файлы.

Заголовочные файлы должны иметь суффикс `.h`. Другие файлы (не заголовочные), предназначенные для включения в код, должны быть с суффиксом `.inc` и могут не иметь блокировки от повторного включения.

Блокировка от повторного включения

Все заголовочные файлы должны иметь защиту от повторного включения посредством `#define`, а не `#pragma once`. Формат макроопределения должен быть: `<PROJECT>_<PATH>_<FILE>_H`.

```
#ifndef FOO_BAR_BAZ_H
#define FOO_BAR_BAZ_H
...
#endif // FOO_BAR_BAZ_H
```

Подключайте используемые заголовочные файлы

Не полагайся на вложенные подключения заголовочных файлов. Это позволит удалить неиспользуемые `#include`, сохраняя при этом корректность остального кода. Это правило актуально даже для парных файлов: если, например, `foo.h` включает `bar.h`, но `foo.cc` также использует определения из bar.h, то foo.cc должен явно подключать `bar.h` сам, независимо от того, что `foo.h` уже его включает.

Встраиваемые функции

Использование встраиваемых функций может генерировать более эффективный код, особенно когда функции маленькие. Используй эту возможность для get/set функций, других коротких и

критичных для производительности функций.

Не стоит делать функции встраиваемыми, если они превышают 10 строк кода. Избегай делать встраиваемыми деструкторы, тк они неявно могут содержать много дополнительного кода: вызовы деструкторов переменных и базовых классов! Обычно нет смысла делать встраиваемыми функции, в которых есть циклы или операции switch (кроме вырожденных случаев, когда цикл или другие операторы никогда не выполняются).

Имена и Порядок включения

Включай заголовочные файлы в следующем порядке:

1. парный файл (например, foo.h для foo.cc),
2. системные файлы C,
3. стандартная библиотека C++,
4. другие библиотеки,
5. файлы твоего проекта.

Отделяй каждую (непустую) группу файлов пустой строкой.

В подключении не стоит использовать UNIX псевдонимы (`.` и `..`).

При подключении заголовочных файлов используй угловые скобки только если это требуется библиотекой. В частности:

- заголовочные файлы стандартных библиотек C и C++
- системные заголовочные файлы POSIX, Linux и Windows

Область видимости

Пространства имен

Размещай свой код в namespace (с уникальным именем). Не используй директиву `using`, что бы избежать потенциальных конфликтов (коллизий) имен.

В конце объявления многострочного пространства имён добавляй комментарий с именем этого пространства имен.

Классы

Код в конструкторе

Конструктор не должен вызывать виртуальные функции:

1. Если в конструкторе вызываются виртуальные функции, то не вызываются реализации из производного класса. Даже если сейчас класс не имеет потомков, в будущем это может

обернуться проблемой.
2. Если возникнет ошибка при инициализации, будет частично (обычно неправильно) инициализированный объект. Очевидное действие: добавить механизм проверки состояния вроде `bool isValid()`.

Используйте Фибричный Метод (

ссылка!) или метод init() Используйте `init()` только в случае, если у объекта есть флаги состояния, разрешающие вызывать те или иные публичные функции (тк сложно полноценно работать с частично сконструированным объектом).

Неявные преобразования

напоминалка

Неявные преобразования позволяют объект одного типа (source type) использовать там, где ожидается другой тип (destination type), например передача аргумента типа int в функцию, ожидающую double.

Не объявляй неявные преобразования. Используй ключевое слово `explicit` для операторов преобразования типа и конструкторов с одним аргументом.

Функции

Желательно писать маленькие и сфокусированные на одной задаче функции. Если ф-я превышает 40 строк, стоит задуматься о разбитии.

Перегрузка функций

Перегружайте функцию, если нет семантических различий между её вариантами. В этом случае допустимо изменять как типы аргументов, так и квалификаторы (const и др.) или количество аргументов. Делайте перегрузку так, чтобы не было нужды разбираться, какая именно версия функции будет вызвана.

Аргументы по-умолчанию

Аргументы по-умолчанию под запретом для виртуальных функций (где они могут работать некорректно) и для случаев, когда значение для аргумента может измениться. Например, не пишите такой код: void f(int n = counter++);.

Синтаксис указания возвращаемого типа в конце

В C++ есть две формы декларации функций. В старой форме возвращаемый тип указывается перед именем функции:

```
int foo(int x);
```

Новая форма использует auto перед именем функции и завершается возвращаемым типом, указываемым после списка аргументов. Например, предыдущую декларацию можно записать как:

```
auto foo(int x) -> int;
```

Используйте обычный (более старый) стиль декларации функции, когда возвращаемый тип указывается перед именем функции. Новую же форму (возвращаемый тип в конце) используйте либо по явной необходимости (лямбды), либо для улучшения читабельности кода. Причём последний вариант (читабельность) часто свидетельствует о чересчур сложных шаблонах, лучше их избегать.

Специфика C++

Именование

case	**type**	**пример**
PascalCase	class	UserAccount
	struct	Point
	typedef	StringList
	using	IntPtr
	enum	Color
camelCase	функции	calculateTotal
	методы классов	addUser
	accessor-ы, get	getUserName
	mutator-ы, set	setUserEmail
snake_case	переменные	user_age
	параметры	file_path
	члены class (*_)	account_balance_
	члены struct	coordinates
	namespace	my_utilities
UPPER_SNAKE_CASE	макросы	MAX_BUFFER_SIZE
	const	DEFAULT_TIMEOUT
	constexpr	PI
	члены enum	RED

Комментарии

Комментарии в шапке файла

Если файл с кодом (например `.h` файл) объявляет несколько абстракций для «внешнего» использования (общие функции, связанные классы и тп), то добавляйте комментарий в шапку, описывающий эту коллекцию абстракций. Не размещайте в этом комментарии детальное описание отдельных абстракций.

Комментарии классов или структур

```
// Перебор содержимого GargantuanTable.
// Пример:
//     std::unique_ptr<GargantuanTableIterator> iter = table->NewIterator();
//     for (iter->Seek("foo"); !iter->done(); iter->Next()) {
//       process(iter->key(), iter->value());
//     }
class GargantuanTableIterator {
  ...
};
```

Комментарии функций

Объявление функции

Определение функций

Комментарии к переменным

Член данных класса

Глобальные переменные

Форматирование