Требования к оформлению

02.02.2022

—

Практикум на ЭВМ
Коровкин Максим Васильевич
Погожев Сергей Владимирович
Севостьянов Руслан Андреевич

Общие требования к работам

Работы выполняются в рамках стандарта C++ 20. Код должен компилироваться и работать на любой платформе: Microsoft Windows®, Linux, macOS®. Это автоматически означает, что все платформенно-зависимые вещи, обычно добавляемые в проект, например, Microsoft Visual Studio, должны быть исключены.

При компиляции программы должны отсутствовать предупреждения. Для GNU Compiler Collection (GCC)/Clang уровень предупреждений задается ключами -Wall -Wextra -Wold-style-cast.

Программы должны представлять собой консольные приложения, обрабатывающие параметры командной строки и использующие стандартные каналы ввода-вывода.

Стандартный набор кодов возврата:

- 0 – успешное завершение программы;
- 1 – некорректные параметры (обычно, в стандартный поток ошибок должно быть выведено описание причины ошибки);
- 2 – внутренняя ошибка программы (в стандартный поток ошибок следует вывести описание ошибки).

Работы выполняются с использованием системы контроля версий. Для работы с системой требуется зарегистрироваться на сайте https://github.com или https://gitlab.com/. После регистрации на сервере необходимо прислать преподавателю свой логин.

В репозитории должен находиться только исходный код и необходимые для компиляции файлы конфигурации. Промежуточные, объектные, бинарные файлы, директории Debug, Release и т. д. не должны включаться в коммиты.

Каждый коммит должен:

- содержать логически единое изменение;
- обладать комментарием, из которого понятно что и зачем было сделано;
- успешно компилироваться и компоноваться;
- проходить unit-тесты, если они есть.

Оформление кода

1. Исходные тексты программы должны содержать не более одного выражения на одной строке.
2. Недопустимо писать функции в одну строку; сигнатура функции всегда должна идти на отдельной строке или нескольких, если сигнатура имеет большую длину.
3. Ставьте пробелы между операторами и операндами:

```
int x = (a + b) * c / d + foo();
```

4. Ставьте пробелы после запятых, например, между аргументами при объявлении или вызове функции:

```
void foo(int x, int y, int z = 0);
```

5. Длина одной строки не должна превышать 120 символов в большинстве случаев и 140 символов всегда. Когда строка становится длиннее допустимого, разделите её на две, сделав перевод на новую строку после оператора, и продолжайте писать:

```
int result = reallyLongFunctionOne() + reallyLongFunctionTwo() +
             reallyLongFunctionThree() + reallyLongFunctionFour();
```

6. Строки не должны заканчиваться пробелами и/или символами табуляции.
7. Оставляйте пустые линии между функциями и между группами выражений:

```
void foo() {
    ...
}
                                // пустая линия
void bar() {
    ...
}
```

8. Файл должен заканчиваться пустой строкой.
9. Все имена должны быть составлены из правильных английских слов или очевидных для читающего сокращений или аббревиатур.
10. Давайте переменным описательные имена, такие как firstName или homeworkScore. Избегайте однобуквенных названий вроде x или c, за исключением итераторов вроде i.
11. Называйте переменные и функции, используя верблюжийРегистр (CamelCase). Называйте классы ПаскальнымРегистром, а константы — в ВЕРХНЕМ_РЕГИСТРЕ.

12. Переменные определяются максимально близко к месту использования. Они должны иметь минимально возможную область видимости.
13. Если переменная используется лишь внутри определенного if, то делайте её локальной, объявляя в том же блоке кода, а не глобальной.
14. Все неизменяющиеся переменные должны быть отмечены квалификатором const.
15. Если определенная константа часто используется в вашем коде, то обозначьте её как const и всегда ссылайтесь на данную константу, а не на её значение:

```
const int VOTING_AGE = 18;
```

16. Никогда не объявляйте изменяемую глобальную переменную. Глобальными переменными должны быть только константы.
17. Аргументы функции, переданные по ссылке или по указателю и не изменяющиеся внутри функции, должны быть отмечены квалификатором const.
18. Отступы должны отражать логическую структуру программы:
    18.1. операторы, выполняемые внутри циклов, должны иметь отступ на 1 больший, чем заголовок цикла;
    18.2. операторы, выполняемые внутри if и else, должны иметь отступ больший, чем строки с if и else;
    18.3. метки операторов case внутри switch должны быть выровнены на тот же отступ, что switch.
    18.4. операторы внутри switch должны иметь на 1 больший отступ, чем оператор switch.
19. Все операторы, входящие в тела циклов и условий, заключаются в фигурные скобки, даже если это один оператор.
20. Фигурные скобки должны быть расставлены единообразно. Допускаются варианты:
    20.1. Открывающая фигурная скобка переносится на следующую строку и имеет тот же отступ, что и предыдущая строка, закрывающая скобка находится на отдельной строке, и ее отступ совпадает с открывающей. Операторы внутри скобок имеют дополнительный отступ. Например:

```
if (condition)
{
    c = a + b;
}
else
{
    c = a - b;
}
```

Ключевое слово else должно находится на отдельной строке.

20.2. Открывающая фигурная скобка завершает строку с оператором, закрывающая скобка находится на отдельной строке, и ее отступ совпадает с отступом строки, содержащей открывающую скобку (так называемые «египетские скобки»). Операторы внутри скобок имеют дополнительный отступ. Например:

```
if (condition) {
    c = a + b;
} else {
    c = a - b;
}
```

Ключевое слово else должно находиться на строке вместе с закрывающей скобкой.

21. Ключевые слова if и else могут быть объединены на одной строке для уменьшения отступов:

```
if (condition1) {
    c = a + b;
} else if (condition2) {
    c = a - b;
}
```

22. Имена классов должны начинаться с большой буквы. Если имя состоит из нескольких слов, они идут подряд без символов подчеркивания, каждое слово начинается с большой буквы (CapsCase). Например, Queue или RoundedRectangle. Исключением из правил могут являться функторы, имена которых могут быть в нижнем регистре. В этом случае отдельные слова в имени разделяются подчеркиваниями.
23. Имена классов должны представлять собой имена существительные с определениями и дополнениями. Исключения: имена функторов должны иметь в основе глагол, так как представляют собой действие.
24. Структуры в терминах C (без конструкторов и деструкторов, а также без методов) допустимо называть в нижнем регистре и разделять слова подчеркиваниями. В этом случае имя должно иметь суффикс «_t».
25. Секции внутри класса должны быть упорядочены единообразно: в порядке убывания или в порядке возрастания интереса к ним со стороны читающего код. Если упорядочено в порядке убывания интереса, то первой должна идти секция public, так как это интерфейс класса для клиентов; затем секция protected, поскольку она представляет собой интерфейс для наследников; завершает класс секция private, так как эта информация интересна только разработчику класса.
26. Внутри каждой секции информация должна быть упорядочена следующим образом:

26.1. типы;

26.2. поля;
26.3. конструктор по умолчанию;
26.4. конструкторы копирования и перемещения;
26.5. все остальные конструкторы;
26.6. деструктор;
26.7. перегруженные операторы;
26.8. методы.

27. Имена функций и методов должны начинаться с маленькой буквы, дополнительные слова должны начинаться с большой буквы и идти вместе с предыдущим (camelCase), например, draw() или getArea().
28. Имена функций должны начинаться с глаголов, так как они представляют собой действие.
29. Все методы, не изменяющие объект логически, должны быть отмечены квалификатором const.
30. Имена полей класса должны иметь отличительный признак поля, в качестве которого следует использовать либо префикс «m_», либо завершающий символ подчеркивания (предпочтительнее).
31. Недопустимо создавать имена, начинающиеся с подчеркиваний.
32. Имена макроопределений должны быть в верхнем регистре.
33. Перенос длинных операторов на другую строку должен выполняться следующим образом:
    33.1. в начале следующей строки должен находиться перенесенный оператор (за исключением запятой);
    33.2. отступ должен быть больше на 2 единицы отступа;
    33.3. в случае переноса внутри сложного выражения с круглыми скобками, отступ должен отражать вложенность скобок, каждый уровень вложения добавляет 1 отступ на следующий строке.

Пример переноса (ширина строки не соблюдена):

```
if (condition) {
    a = a + b
        * (c
          + d / e);
}
```

34. Список инициализации в конструкторе должен инициализировать не более одной переменной или базового класса на строке. Двоеточие перед списком инициализации должно оставаться строке со списком параметров, недопустимо переносить его на следующую строку. Аналогично оформляются запятые, разделяющие элементы списка инициализации: запятая остается на строке с предыдущим элементом.

35. В списках инициализации массивов допускается применение «висящей» запятой после последнего элемента, так как это уменьшает «визуальный шум» в изменениях при увеличении списка инициализации.
36. Внутри выражений круглые скобки должны применяться для исключения неоднозначностей. За исключением очевидных из школьной арифметики приоритетов, все выражения оформляются так, чтобы при чтении не требовалось знать таблицу приоритетов операторов.
37. Не допускается использование конструкций using namespace на верхнем уровне в заголовочных файлах.
38. Порядок включения заголовочных файлов важен:
    38.1. внутри файла реализации первым включается соответствующий ему заголовок;
    38.2. стандартные заголовки;
    38.3. заголовки использованных библиотек;
    38.4. свои заголовки.
39. Зависимости от заголовков должны быть минимизированы. Не допускается включение заголовков с реализациями, если достаточно объявлений.
40. В файлах реализации все функции должны иметь полные имена, в которых пространства имен и имена классов отделены от имени функции при помощи ::. Например, для функции в пространстве имен lab::detail с именем foo() реализация должна выглядеть так:

```
void lab::detail::foo()
{
...
}
```

Реализация функции в точке объявления допустима только для статических функций внутри единицы трансляции, шаблонных функций, методов шаблонных классов и однострочных методов, наподобие get-методов.

Комментарии

1. **Заглавный комментарий.** Размещайте заглавный комментарий, который описывает назначение файла, вверху каждого файла. Предположите, что читатель вашего комментария является продвинутым программистом, но не кем-то, кто уже видел ваш код ранее.
2. **Заголовок функции / конструктора.** Разместите заголовочный комментарий на каждом конструкторе и функции вашего файла. Заголовок должен описывать поведение и / или цель функции.

3. **Параметры / возврат.** Если ваша функцию принимает параметры, то кратко опишите их цель и смысл. Если ваша функция возвращает значение — кратко опишите, что она возвращает.
4. **Исключения.** Если ваша функция намеренно выдает какие-то исключения для определенных ошибочных случаев, то это требует упоминания.
5. **Комментарии на одной строке.** Если внутри функции имеется секция кода, которая длинна, сложна или непонятна, то кратко опишите её назначение.
6. **TODO.** Следует удалить все // TODO комментарии перед тем, как заканчивать и сдавать программу.

Требования к дизайну программы

Общий дизайн выполненной работы должен удовлетворять следующим требованиям, в порядке убывания приоритета:

1. Удовлетворять как минимум базовой гарантии безопасности исключений, предпочтительно удовлетворять строгой гарантии.
2. Если вы используете один и тот же код дважды или более, то найдите способ удалить излишний код, чтобы он не повторялся. К примеру, его можно поместить во вспомогательную функцию. Если повторяемый код похож, но не совсем, то постарайтесь сделать вспомогательную функцию, которая принимает параметры и представляет разнящуюся часть.

```
// Плохая практика
foo();
x = 10;
y++;
...

foo();
x = 15;
y++;

// Хорошая практика
helper(10);
helper(15);
...

void helper(int newX) {
    foo();
    x = newX;
    y++;
}
```

3. Переместите общий код из выражения if / else, чтобы он не повторялся:

```
// Плохая практика
if (x < y) {
    foo();
    x++;
    std::cout << "Привет!";
} else {
    foo();
    y++;
    std::cout << "Привет!";
}

// Хорошая практика
foo();
if (x < y) {
    x++;
} else {
    y++;
}
std::cout << "Привет!";
```

4. Ввод-вывод данных и логика их обработки должны быть строго разделены.

```
// Плохая практика
void doubleVal()
{
    int val;
    std::cout << “Enter val: “;
    std::cin << val;

    std::cout << “Double val: ”;
    std::cout << 2*val << ‘\n’;
}

// Хорошая практика
int getVal()
{
    int val;
    std::cout << “Enter val: “;
    std::cin << val;
    return val;
}

int doubleVal(int val)
{
    return 2*val;
}
```

```
int printVal(int val)
{
    std::cout << “Val is “ << val << ‘\n’;
}

int main()
{
    int val = getVal();
    int doubled = doubleVal(val);
    printVal(doubled );
    return 0;
}
```

5. Каждый написанный класс должен перегружать все операторы, имеющие смысл для него. Например, класс матрицы должен перегружать операторы сравнения на равенство и неравенство, сложения, вычитания и умножения. Отношение матриц (больше или меньше) для матриц не могут быть однозначно определены и, соответственно, не могут определяться в виде перегруженных операторов.
6. Корректно обрабатывать ошибки при взаимодействии с внешним миром: ошибки ввода-вывода, некорректный пользовательский ввод и так далее.
7. Необходимо корректно разбирать, а также выводить данные в соответствии с настройками пользователя (локализация).
8. Программы не должны содержать лишних сущностей. Например, создавать классы только для одного метода недопустимо. Дизайн должен быть минималистичным настолько, насколько это возможно.
9. Также недопустимо создавать методы, идентичные создаваемым компилятором. Например, написание тривиального (пустого) открытого невиртуального деструктора недопустимо.
10. Все детали реализации должны быть скрыты от клиентского кода. Если это шаблон, который невозможно скрыть, он должен быть помещен во вложенное пространство имен detail.
11. За исключением самых простых, работы не должны выполняться в рамках одной единицы трансляции. Должно присутствовать разумное деление по файлам и правильно сформированные заголовочные файлы с минимальными зависимостями.

Список источников информации

1. Google C++ Style Guide. URL: https://google.github.io/styleguide/cppguide.html
2. Stanford C++ Style Guide». URL: http://stanford.edu/class/archive/cs/cs106b/cs106b.1158/styleguide.shtml