Советы и правила по программированию C++

Александр Изотов

18 августа 2020 г.

Аннотация

Сделано по книге Ravesli

Объявление и и инициализация переменных

Правило: Если у вас изначально имеется значение для переменной – используйте инициализацию, вместо присваивания.

Хорошей практикой считается всегда инициализировать свои переменные. Это будет гарантией того, что ваша переменная всегда имеет одно и то же значение и вы не получите ошибку от компилятора.

Правило: Убедитесь, что все ваши переменные в программе имеют значения (либо через инициализацию, либо через операцию присваивания).

Функции

1 Код, который появляется более одного раза в программе, лучше переписать в виде функции. Например, если мы получаем данные от пользователя несколько раз одним и тем же способом, то это отличный вариант для написания отдельной функции.

2 Код, который используется для сортировки чего-либо, лучше записать в виде отдельной функции. Например, если у нас есть список вещей, которые нужно отсортировать - пишем функцию сортировки, куда передаём несортированный список и откуда получаем отсортированный.

3 Функция должна выполнять одно (и только одно) задание. Когда функция становится слишком большой, сложной или непонятной – её следует разбить на несколько подфункций. Это называется рефакторинг кода.

Правило: Имена, которые используются внутри функции (включая параметры), доступны/видны только внутри этой же функции.

Идентификатор — это имя переменной, функции, класса или другого объекта в C++. Мы можем определять идентификаторы любыми словами/именами. Тем не менее, есть несколько общих правил, которые необходимо соблюдать:

1 Идентификатор не может быть ключевым словом. Ключевые слова зарезервированы.

2 Идентификатор может состоять только из букв (нижнего или верхнего регистра), цифр или символов подчёркивания. Это означает, что все другие символы и пробелы - запрещены.

3 Идентификатор должен начинаться с буквы (нижнего или верхнего регистра). Он не может начинаться с цифры.

4 C++ различает нижний регистр от верхнего. nvalue отличается от nValue и отличается от NVALUE.

Если вы работаете с чужим кодом, то хорошей практикой будет придерживаться стиля, в котором написан этот код, даже если он не соответствует рекомендациям выше.

* не начинайте ваши имена с символа подчёркивания, так как такие имена уже зарезервированы для ОС, библиотеки и/или используются компилятором.

* используйте в качестве идентификаторов только те имена, которые реально описывают то, чем является объект. Очень характерно для неопытных программистов сокращать имена переменных, дабы сэкономить время при наборе кода или потому что они думают, что всё и так понятно. В большинстве случаев не всё всегда является понятным и очевидным. В идеале переменные нужно называть так, чтобы человек, который первый раз увидел ваш код, понял, как можно скорее, что этот код делает. Через 3 месяца, когда вы будете пересматривать свои программы, вы забудете, как они работают, и будете благодарны самому себе за то, что называли переменные по сути, а не как попало. Чем сложнее код, тем проще и понятнее должны быть идентификаторы.

int ccount	Плохо	Никто не знает, что такое ccount
int customerCount	Хорошо	Теперь понятно, что мы считаем
int i	Плохо*	В большинстве нетривиальных случаев - плохо, простых примерах - может быть (например, в циклах)
int index	50/50	Хорошо, если очевидно, индексом чего является переменная
int totalScore	Хорошо	Всё понятно
int _count	Плохо	Не начинайте имена переменных с символов подчёркивания
int count	50/50	Хорошо, если очевидно, что мы считаем
int data	Плохо	Какой тип данных?
int value1, value2	50/50	Может быть трудно понять разницу между переменными
int numberOfApples	Хорошо	Всё понятно
int monstersKilled	Хорошо	Всё понятно
int int x, y	Плохо*	В большинстве нетривиальных случаев - плохо, в простых примерах - может быть (например, в математических функциях)

*Примечание: Можно использовать тривиальные имена для переменных, которые имеют тривиальное использование (например, для переменных в цикле, в простых математических функциях и т.д.).

Литерал - это фиксированное значение, которое записывается непосредственно в исходном коде (например: 7 или 3,14159). **Литералы**, переменные и функции ещё известны как *операнды*.

Операнды - это данные, с которыми работает выражение. **Литералы** имеют фиксированные значения, переменным можно присваивать значения, функции же производят определённые значения (в зависимости от типа возврата, исключение: функции типа void).

Операторы. Последним пазлом в выражениях являются **операторы**. С их помощью мы можем объединить операнды для получения нового зна-

чения. Например, в выражении "5 + 2 + является оператором. С помощью + мы объединили операнды 5 и 2 для получения нового значения (7).

В отличие от других языков, C++ не имеет каких-либо ограничений в форматировании кода со стороны программистов. Основное правило заключается лишь в том, чтобы использовать только те способы, которые максимально улучшают читабельность и логичность кода.

Вот пять основных рекомендаций:

Рекомендация #1: Вместо клавиши **Tab** используйте **4 пробел**а. В некоторых IDE по умолчанию стоят тройные пробелы в качестве одного Tab - это тоже нормально (количество пробелов можно легко настроить в соответствующих пунктах меню вашей IDE). Причиной использования пробелов вместо символов табуляции является то, что если вы откроете свой код в другом редакторе, то он сохранит правильные отступы, в отличие от использования клавиши Tab.
Рекомендация #2: Каждый стейтмент функции должен быть с соответствующим отступом (Tab или 4 пробела)
Рекомендация #3: Строки не должны быть слишком длинными. 72, 78 или 80 символов – это оптимальный максимум строки.
Рекомендация #4: Если длинная строка разбита на части с помощью определённого оператора (например, « или +), то этот оператор должен находится в конце этой же строки, а не в начале следующей.
Рекомендация #5: Используйте пробелы для улучшения читабельности вашего кода.

```
nCost          = 57;
nPricePerItem  = 24;
nValue         =  5;
nNumberOfItems =  17;
```

Правило: Используйте угловые скобки для подключения "системных"заголовочных файлов и двойные кавычки для всего остального (ваших заголовочных файлов).

Правило: При подключении заголовочных файлов из стандартных библиотек C++, используйте версии без ".h"(если они существуют). Пользовательские заголовочные файлы должны иметь окончание ".h".

Вот несколько советов по написанию собственных заголовочных файлов:

* Всегда используйте директивы препроцессора. Не определяйте переменные в заголовочных файлах, если это не константы. Заголовочные файлы следует использовать только для объявлений.

* Не определяйте функции в заголовочных файлах.

* Каждый заголовочный файл должен выполнять свою определённую работу и быть как можно более независимым. Например, вы можете поместить все ваши объявления, связанные с файлом A.cpp в файл A.h, а все ваши объявления, связанные с B.cpp в файл B.h. Таким образом, если вы будете работать только с A.cpp, то вам будет достаточно подключить только A.h и наоборот.

* Используйте имена ваших рабочих файлов в качестве имён для ваших заголовочных файлов (например: grades.h работает с grades.cpp).

* Не подключайте одни заголовочные файлы из других заголовочных файлов.

* Не #include файлы .cpp.

В uniform инициализации есть дополнительное преимущество: вы не можете присвоить переменной значение, которое не поддерживает её тип данных - компилятор выдаст предупреждение или сообщение об ошибке. Например: int value{4.5}; // ошибка: целочисленная переменная не может содержать не целочисленные значения

Правило: Используйте uniform инициализацию.

В C++ приоритетным является объявлять переменные как можно ближе к их первому использованию

Правило: Объявляйте переменные как можно ближе к их первому использованию.

Определение целочисленных переменных

short int si;	допустимо
short s;	предпочтительнее
long int li;	допустимо
long long int lli;	допустимо
long long ll;	предпочтительнее

Переполнение

Правило: Никогда не допускайте возникновения переполнения в ваших программах!

Магические числа

Так лучше не делать! int maxStudents = numClassrooms * 30; setMax(30);

Избегайте "магических"констант в коде (простые, ничего не значащие цифры). Обозначьте их как const и ссылайтесь на константы, а не на значения:

// так делать не стоит:

```
for (int i = 0; i < 3; i++) {
    for (int j = 0; j < 7; j++) {
        if (field[i, j] == 0)
            showError(451);
    }
}
```

// лучше написать так:

```
const int FIELD\_HEIGHT = 3; // высота поля
const int FIELD\_WIDTH = 7; // ширина поля

const int ZERO\_FIELD\_CODE = 451; // код ошибки в случае нулевой ячейки поля

    ...

for (int i = 0; i < FIELD\_HEIGHT; i++) {
    for (int j = 0; j < FIELD\_WIDTH; j++) {
        if (field[i, j] == 0)
            showError(ZERO\_FIELD\_CODE);
    }
}
```

Правило: Старайтесь сократить к минимуму использование магических чисел в ваших программах.

Циклы и условные операторы

Циклы позволяют выполнять один код несколько раз, однако существуют различные виды циклов. Для каждой ситуации один цикл может подойти лучше, а другой хуже.

Используйте цикл for, когда количество повторений заранее известно, а цикл while, если число итераций заранее посчитать невозможно:

```
// повторять 'size' раз
for (int i = 0; i < size; i++) {
    ...
}

// повторять, пока есть что считывать
Point p;
while (std::cin >> p.x >> p.y) {
    ...
\\}
```

При использовании операторов цикла (for / while / do-while) или условных операторов(if / else) всегда ставьте фигурные скобки и соответствующие отступы, даже если тело всего оператора состоит всего из одной строки:

```
// так делать не стоит:
if (isWin(gameField)) return;
    else for (int i = 0; i < freeCells.size(); i++) freeCells[i].calcValue();

// лучше написать так:
if (isWin(gameField)) {
    return;
}
else {
    for (int i = 0; i < freeCells.size(); i++) {
        freeCells[i].calcValue();
    }
}
```

Инкременты

Правило: Используйте префиксный инкремент и префиксный декремент вместо постфиксного инкремента и постфиксного декремента. Версии префикс не только более производительны, но и ошибок с ними (по статистике) меньше.

// Так лучше не писать!
int value = add(x, ++x); // здесь 5 + 6 или 6 + 6? Это зависит от компилятора и в каком порядке он будет обрабатывать аргументы функции

Правило: Не используйте переменную с побочным эффектом (инкремент) больше одного раза в одном стейтменте.

Тернарный оператор

```
std::cout << ((x > y) ? x : y);
bool inBigClassroom = true;
const int classSize = inBigClassroom ? 3 : 5;
std::cout << classSize << std::endl;
```

Правило: Используйте условный тернарный оператор ?: только в простых случаях.

Логические операторы и операторы сравнения

Не рекомендуется использовать операторы == или != с числами типа с плавающей точкой.

Рассмотрим следующее выражение: value1 || value2 && value3. Поскольку приоритет логического И выше, то обрабатываться выражение будет так:

value1 || (value2 && value3) А не так:
(value1 || value2) && value3

Хорошей практикой является использование круглых скобок с операциями. Это предотвратит ошибки приоритета, увеличит читабельность кода и чётко даст понять компилятору, как следует обрабатывать выражения. Например, вместо того, чтобы писать:
value1 && value2 || value3 && value4
лучше записать
(value1 && value2) || (value3 && value4).

Законы Де Моргана

Многие программисты совершают ошибку, думая, что !(x && y) - это то же самое, что и !x && !y. К сожалению, вы не можете "использовать"логическое НЕ подобным образом.
Законы Де Моргана гласят, что:
!(x && y) эквивалентно !x || !y
!(x || y) эквивалентно !x && !y

Другими словами, логические операторы И и ИЛИ меняются местами! В некоторых случаях, это даже полезно: улучшает читабельность.