**Рекомендации по написанию кода в C++**

Вы сталкивались с трудностями в понимании чужого кода? Если — нет, то вам еще рано изучать эту тему, если — да, то эта статья и последующие в этом разделе — для вас. Вы никогда не задумывались над тем, почему чужой код намного сложнее понять, чем свой собственный? Все дело в том, что у каждого программиста есть свой стиль написания кода. Например, кто-то не ставит отступы для выделения области действия, а кто-то наоборот — ставит, но слишком большие. Кто-то, сам того не подозревая, дает магические имена переменным, которые ну никак по смыслу не связаны с их назначением. Многие не используют комментарии в коде, хотя это очень важно. Одни программисты пытаются соблюдать код-стандарты, но не полностью, другие — вообще ничего об этом не слышали и слышать не хотят. В итоге получается макаронный код, в нем сложно распознать даже самый простой алгоритм, в таком коде легко допустить ошибку и, самое главное, его трудно читать.

Вы спросите: «А зачем мне разбираться в чужом коде?» Как минимум для того, чтобы использовать уже запрограммированные алгоритмы, вы не будете тратить время на «изобретение своего велосипеда». Кроме того, если вы работаете над большим проектом и не один, а целой командой, вы просто обязаны понимать чужой код, и конечно же, вы обязаны писать код, который будет понятен другим программистам из вашей команды. Что это дает? Это позволяет экономить кучу времени, вам и вашей команде. Но даже, если вы программируете сами для себя, и никто кроме вас не будет разбираться в вашем коде, вы все равно выигрываете, хотя бы потому, что спустя некоторое время, без использования код-стандартов, вам самим сложно будет разобраться в своем же коде. Переменные, которые вы объявляли неделю назад, станут не понятными, не говоря уже о том, что вы не будете знать, как именно работает программа.

Теперь, когда мы знаем, для чего нужны стандарты кодирования, приступим к их изучению. Существует очень много код-стандартов, у каждого из которых есть свои плюсы и минусы, по этой причине, в разных командах — разные стандарты написания кода, которые действуют в рамках данной команды разработчиков.

Данная статья не претендует на звание — «Код-стандарты С++», скорее, — это просто мои рекомендации по написанию кода в языке С++. Все эти рекомендации я не придумал, они существуют и в книгах и в интернете. Я же просто постараюсь вас хотя бы поверхностно с ними ознакомить:

1. **Всегда используйте IDE для программирования.**  
   Ушли те времена, когда необходимо было составлять программы в блокнотах и потом вручную собирать написанный код в объектный файл. Друзья, в мире существует очень много классных программ, которые подсвечивают, код, форматирую его и автоматически его собирают в готовую программу. Кроме всего этого, IDE информирует об возможных ошибках, еще до этапа сборки программы, подсвечивают неиспользуемые переменные, а также легко настраиваются под индивидуальные требования программистов. IDE даже может сама писать код, как это удобно реализовано в MVS, например, мы объявляем оператор switch, IDE уже знает об этом и делает это сама, нам же остается только добавить нужные переменные и константы. Все эти полезности реализованы с одной единственной целью — сделать процесс составления программ намного приятнее и намного быстрее. Так пользуйтесь же всем этим!
2. **Пользуйтесь системами контроля версий.**  
   Если вы работаете над программой долгое время, около недели или больше, не поленитесь создать репозиторий для вашей программы. Зачем это нужно? Когда разработка программы сильно затянулась, то вы уже не всегда сможете вспомнить что и когда добавляли в программный код, откуда взялась та или иная ошибка. Система контроля версий поможет вам ответить на все эти вопросы. В конце концов, если вдруг вы случайно что-то поломаете в программе, система контроля версий всегда сможет восстановить предыдущую, рабочую версию программы. Не поленитесь в этом разобраться, система контроля версий — это очень классная штука.
3. **Размещение фигурных скобок у функций:**

|  |  |
| --- | --- |
| 1  2  3 | int myFunction()  {  } |

1. Нужно чтобы открывающаяся и закрывающаяся скобки находились в отдельных строках, по своему опыту могу сказать, так четко просматривается область видимости функции, её тело и заголовок.
2. **Отступы должны быть табами.**  
   Вместо пробелов, отступы лучше проставлять табами, это намного удобнее, в том смысле, что экономит время. Ведь проще нажать один раз — таб, чем несколько раз пробел. Некоторые говорят, что табами проставлять отступы — это не правильно, так как в разных редакторах длинна таба меняется. Соответственно, и отформатированный код будет выглядеть не одинаково. Возможно они и правы, но я не встречал такого, хотя я программировал на многих IDE, табы почти везде одинаковые.
3. **Имена функций должны быть в верблюжьем регистре.**  
   Следует давать имена функциям в верблюжьем регистре, причем начинаться имя должно всегда с маленькой буквы:

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9 | // Правильно:  myFunction()  searchInFile()  // Неправильно:  SearchInFile()  search\_in\_file()  myfunction()  Myfunction()  MYFUNCTION() |

1. Вот еще несколько полезных статей: [правила именования переменных](http://cppstudio.com/spravochnik/sintaksis-yazyka-s-spravochnaya-informaciya/sintaksis-obyavleniya-peremennyx-v-s/) и [соглашения об именовании](http://cppstudio.com/uchebniki/yazyk-programmirovaniya-s/vvedenie-v-c/stil-programmirovaniya-soglasheniya-ob-imenovanii/). Обязательно прочитайте их, в дальнейшем, вам очень помогут эти знания.
2. **Оператор if-else.**

|  |  |
| --- | --- |
| 1  2  3  4  5  6 | if () {      // тут код  }  else {      // и тут код  } |

1. Ну во первых, всегда ставьте фигурные скобки, даже если в операторе if нужно выполнить всего одну строку кода. Во вторых, переносите оператор else, на новую строку, хотя, на первый взгляд это может показаться и не удобно. В чем плюс? А плюс в том, что если вдруг вам нужно будет закомментировать или удалить блок кода с оператором else, то вам это будет сделать очень просто. И поверьте, такие ситуации встречаются очень часто.  
   Вот запись, как делать не следует:

|  |  |
| --- | --- |
| 1  2  3  4  5 | if () {      // тут код  } else {      // и тут код  } |

1. Оператор else стоит в строке с закрывающейся скобкой тела оператора if. Такая запись кажется намного компактнее и красивее. Но как я уже говорил, когда вам понадобится закомментировать блок кода else, вы случайно закомментируете или удалите и закрывающую скобку предыдущего блока. В результате получите ошибку, которая отберет у вас драгоценное время.  
   Такая запись, по моему мнению, также считается не правильной:

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8 | if ()  {      // тут код  }  else  {      // и тут код  } |

1. Она получается излишне большой.
2. **Используйте оператор const как можно чаще.**  
   Это дает огромное преимущество в отлаживании работы программы. Например, вы написали программу, но она работает не правильно, то есть в код прокралась ошибка, причем эта ошибка связана с изменением значения в переменной, в которой это значение меняться не должно. Вы начинаете просматривать весь код в поисках этой ошибки, на все это уйдет не мало времени. Если бы вы сразу объявили константу а не переменную, то ошибка б ы проявилась еще на этапе компиляции и вам бы не пришлось тратить время на её поиск. Рассмотрим пример:

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12 | #include <iostream>  using namespace std;  int main()  {      const int august = 8;      int currentMonth;      cin >> currentMonth;      if (august == currentMonth) {          cout << "Current month - august";      }      return 0;  } |

1. Это корректный код, он очень простой для понимания. Пользователь вводит номер месяца, если введенный номер равен 8 — август, то выводится сообщение. Обратите внимание, что в операторе if в условии я сравниваю значение константы, ведь август — это всегда 8-й месяц, а значит значение константы  august меняться не должно. Теперь допустим одну маленькую ошибку в операторе if, вот код:

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12 | #include <iostream>  using namespace std;  int main()  {      const int august = 8;      int currentMonth;      cin >> currentMonth;      if (august = currentMonth) {          cout << "Current month - august";      }      return 0;  } |

1. Программа таrже, но уже с маленькой ошибкой в условии. Вместо оператора сравнения там стоит оператор присваивания. И тут нас спасет тот случай, что august — это константа, а константе присваивать ничего нельзя и компилятор об этом знает. То есть на этапе компиляции вылетит ошибка:
2. main.cpp:12: error: assignment of read-only variable ‘august’
3. Согласитесь, что очень удобно! Теперь сделаем august не константой, вот код:

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12 | #include <iostream>  using namespace std;  int main()  {      int august = 8;      int currentMonth;      cin >> currentMonth;      if (august = currentMonth) {          cout << "Current month - august";      }      return 0;  } |

1. august — обычная переменная. И к вашему удивлению, даже с ошибкой, программа будет собрана и запустится, вот только работать она будет не правильно и вам придется потратить некоторое время чтобы исправить её. Программисты соблюдающие код-стандарт даже допустив эту ошибку быстро её смогут исправить, а те, кто пишет небезопасный код, будут долго искать эту ошибку, особенно в том случае, если у вас сотни строк кода.
   1. **Избегайте магических чисел в программе, это кстати частный случай из предыдущего примера.**  
      Суть его вот в чем, например, в операторе if сравнение можно было бы сделать так:

|  |  |
| --- | --- |
| 1  2  3 | if (8 == currentMonth) {    } |

* 1. Но у человека, который первый раз видит ваш код сразу возникнет вопрос, что это за магическое число 8? Конечно в данном примере не сложно догадаться, что значит восемь, но даже если и так, какой в этом смысл? Другое дело, если бы код выглядел так:

|  |  |
| --- | --- |
| 1  2  3  4  5 | const int august = 8;    if ( august == currentMonth) {          cout << "Current month - august";  } |

* 1. По имени переменной понятно, что 8 — это порядковый номер месяца, все предельно ясно и просто.
  2. **В условиях сравнивайте константное значение со значением переменной, а не наоборот.**  
     Данная рекомендация немного усложняет понимание кода, но зато предотвращает появление ошибок времени компиляции. Вернемся все к тому же примеру из пункта 5. Вот фрагмент кода:

|  |  |
| --- | --- |
| 1  2 | if (august == currentMonth) {  } |

* 1. В условии я сравниваю значение константы со значением переменной, если бы было наоборот, то ошибка в пункте 5 не была бы поймана. Обратите на это внимание и стройте условие по принципу сравнивания константного значения с переменной, а не переменную с константным значением, хотя это и логичнее. Вот так делать не надо:

|  |  |
| --- | --- |
| 1  2 | if (currentMonth == august) {  } |

* 1. **Всегда выполняйте начальную инициализацию переменных.**  
     Соблюдение этой рекомендации поможет сэкономить вам кучу времени. Пример из жизни:

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11 | #include <iostream>  using namespace std;  int main()  {      int sum = 0;      for(int i = 0; i < 5; i++) {          sum++;      }      cout << sum;      return 0;  } |

* 1. В переменной sum в цикле накапливается сумма. Программа хорошо работает, проверьте. Заметьте, что переменной sum при объявлении был присвоен нуль, это и есть начальная инициализация. Тот же пример но без начальной инициализации:

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11 | #include <iostream>  using namespace std;  int main()  {      int sum;      for(int i = 0; i < 5; i++) {          sum++;      }      cout << sum;      return 0;  } |

* 1. Результат будет неправильный, так как небыло начальной инициализации, соответственно и сумма будет неправильной.
  2. **Старайтесь разбивать большие вычисления на отдельные функции.**  
     Код состоящий из нескольких вызовов функций будет намного понятнее, чем код построенный без использования функций и с большим количеством вложенностей. Тут примера приводить не буду, подробнее об этом можно почитать в статье о [процедурном программировании](http://cppstudio.com/uchebniki/yazyk-programmirovaniya-s/funkcii-v-s/).

1. **Старайтесь объявлять все свои переменные как можно более локальнее.**  
   Эта рекомендация поможет вам более эффективнее использовать память компьютера. Кроме того, вероятность появления ошибки конфликта имен будет минимальной. Давайте рассмотрим пример:

|  |  |
| --- | --- |
| 1  2  3  4 | // фрагмент кода  for(int i = 0; i < 5; i++) {      sum++;  } |

1. Управляющая переменная i объявлена внутри цикла, это значит то, что область её видимости ограничивается областью видимости цикла. С точки зрения физических процессов в памяти, это значит, что когда цикл завершится, память отводимая под переменную i — освободится, а сама переменная i будет удалена. Это процесс полностью обоснован, ведь цикл закончился и переменная уже не нужна, так зачем же она будет занимать память. Тот же пример, но в нем переменная i будет объявлена вне цикла. Этот пример я не придумал, я его часто встречаю в жизни, почему-то некоторым программистам нравится делать так:

|  |  |
| --- | --- |
| 1  2  3  4 | int i;  for(i = 0; i < 5; i++) {      sum++;  } |

1. или

|  |  |
| --- | --- |
| 1  2  3  4 | int i = 0;  for(; i < 5; i++) {      sum++;  } |

1. Обе эти записи абсолютно неприемлемы, это неправильно. Кроме того эти записи выглядят не красиво. Но кроме косметических недостатков, есть другой — намного более важный. Он связан с тем, что при такой организации кода, даже когда цикл завершит свою работу, переменная i не будет уничтожена, а значит будет занимать лишнюю память. Кроме всех этих недостатков возможна ситуация возникновения конфликта имен, вот пример.

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9 | int i = 0;  for(; i < 5; i++) {      sum++;  }    int i = 0;  for(; i < 5; i++) {      sub--;  } |

1. Обычная ситуации, мы в коде воспользовались двумя циклами, причем не важно, что конкретно каждый из них делает. Важно то, что они используют одну и ту же управляющую  переменную. Отсюда возникнет такая ошибка:
2. main.cpp:14: error: redeclaration of ‘int i’
3. Но если бы циклы были построены так, то ошибок бы не было:

|  |  |
| --- | --- |
| 1  2  3  4  5  6 | for(int i = 0; i < 5; i++) {      sum++;  }  for(int i = 0; i < 5; i++) {      sub--;  } |

1. И память высвобождается в таком случае правильно.

Если вы до конца внимательно прочитали эту статью, можно считать, что качество вашего кода отныне стало лучше, хотя бы, на чуть чуть. И теперь, когда вы приступите к написанию своей очередной программы, не забудьте применять на практике все те приемы, о которых вы прочитали в этой и [предыдущей](http://cppstudio.com/uchebniki/rekomendacii-po-napisaniyu-koda-v-s/rekomendacii-po-napisaniyu-koda-v-c-chast-1/) статьях. Если вы хотите изучить более подробно стандартизации в С++, прочитайте книгу «Стандарты программирования на С++ Герб саттер и Андрей Александреску».