Автор: Андрей
10.10.2010 00:00 - Обновлено 09.08.2014 22:15

Предисловие

Прежде чем начать свое сегодняшнее повествование, сразу хочу оговориться: в этом уроке *не* будет содержаться никаких сведений о средствах языка программирования C++ и после его прочтения вы *не* научитесь писать программы с новыми возможностями. Несмотря на это, я настоятельно рекомендую прочитать этот урок всем, кто хочет научиться писать хорошие и работоспособные программы на C++ и желает поднять свой уровень как программиста.

Почему важно придерживаться хорошего стиля в написании программ

Для написания действительно *хорошей программы* на C++ недостаточно просто владеть широким набором средств языка. Необходимо еще проявлять хорошее понимание используемых средств, а также грамотно оформлять их использование. Помимо эффективности код любой хорошей программы должен обладать такими свойствами как ясность, читабельность, логичность и недвусмысленность. Известно, что ошибкам подвержен абсолютно любой код (за исключением самых тривиальных случаев вроде "Hello, world!"), поэтому сама структура оформления вашего кода должна облегчать нахождение ошибок, исключать неправильное понимание конструкций и представлять абсолютно простую и логичную схему поведения.

Еще одно соображение в пользу ясного оформления кода состоит в том, что уже написанный код периодически приходится пересматривать и модернизировать. Это может произойти в процессе обнаружения ошибок в проектировании достаточно большой программы, в ходе естественного процесса улучшения и дополнения уже написанной программы, или же если вы хотите использовать элементы старой программы при создании новой. И это все - только с учетом возможности того, что вы сами будете читать свой собственный код через некоторое время после его написания. Вполне может случиться, что ваш код придется смотреть совершенно другому человеку и ему надо будет как-то разбираться в том, что вы написали. Поверьте, нет ничего приятного в том, чтобы пытаться вникнуть в чужой или свой собственный, созданный полгода назад, код, оформленный абы как и совершенно лишенный адекватных комментариев.

Итак, если вы не хотите даром тратить свое время на отлов многих, подчас глупых, а

подчас и очень тонких, хорошо законспирированных в непонятном коде ошибок, а также на продирание сквозь дебри запутанных и нелогичных конструкций собственного производства, то вам совершенно необходимо выработать навыки грамотного и четкого оформления вашего кода, а также некоторые другие навыки, о которых и пойдет речь ниже.

Отступы

Одним из основополагающих элементов хорошего стиля написания кода программ является согласованное использование отступов. Чтобы убедиться в необходимости применения отступов, давайте рассмотрим два варианта оформления одного и того же фрагмента кода - с отступами и без на примере функции, перемножающей две матрицы. Итак, первый вариант:

```
void mult(const double ** a, const double ** b, double ** c)
{
for (int i = 0; i for (int j = 0; j {
double t = 0;
for (int k = 0; k {
t += a[i][t] * b[t][j];
}
c[i][j] = t;
}
}
```

Несколько запутанно, не правда ли? Что за чем выполняется - примерно понятно, кое-где приходится поднапрячься, чтобы ничего не перепутать. А теперь посмотрим, как то же самое можно оформить с использованием отступов:

```
void mult(const double ** a, const double ** b, double ** c)
{
for (int i = 0; i for (int j = 0; j {
    double t = 0;
    for (int k = 0; k {
      t += a[i][t] * b[t][j];
      }
    c[i][j] = t;
    }
}
```

На мой взгляд, такой код гораздо понятнее. И, что важно, при таком оформлении, "лесенкой", очень трудно потерять открывающую или закрывающую фигурную скобку. А если компилятор все-таки обнаружит подобную ошибку, то очень легко будет проверить

парность скобок. В варианте кода без использования отступов это сделать значительно труднее!

Вообще достаточно объемный код, в котором начисто отсутствуют отступы, очень трудно читать. Если в нем присутствует описание нескольких классов и структур, достаточно большое количество функций, и они при этом не выделены такой "лесенкой", то ориентирование в таком коде можно смело приравнять к ориентированию в глухую беззвездную ночь в старом болотистом лесу без фонарика, компаса и - упаси Боже! - GPS-навигатора.

Чтобы избежать таких проблем, вы можете использовать любую разумную систему отступов, лишь бы она была вам по нраву и помогала ориентироваться в коде. Я же опишу систему, которой пользуюсь сам, и еще выскажу несколько соображений на этот счет. Итак, согласно этой системе единообразный отступ, равный трем пробелам, применяется для всех вложенных инструкций или блоков инструкций. Это относится к инструкциям, следующим за циклами for, while, do, условиями if-else, а также к инструкциям внутри switch, телам функций и определениям структур и классов. При этом если вложенным оказывается блок инструкций (это в том числе тела функций, инструкции внутри switch, определения классов), то обрамляющие этот блок фигурные скобки предваряются тем же отступом, что и сам блок, и каждая скобка располагается на отдельной строке (в некоторых системах отступов открывающая скобка может, например, находиться на той же строке, что и инструкция, к которой относится блок; я считаю такой вариант неудобочитаемым). При этом исключениями являются:

- функция main, тело которой я не отделяю отступами (это в значительной степени является моим субъективным предпочтением),
- в некоторых случаях - определения функций-членов классов, находящиеся внутри определения класса (например, состоящих всего из одной инструкции - весь блок я записываю в той же строке, что и имя функции),
- пространства имен (я не отделяю их содержимое отступами)

Скобки

Как известно, существует такое понятие, как приоритет операторов. Т.е. в конструкции, составленной из нескольких операторов, одни из них будут выполняться раньше других, причем если очень постараться, то можно точно предсказать, какие раньше, а какие - позже. К сожалению, часто так бывает, что компилятор имеет свое собственное мнение по этому поводу, несколько отличающееся от нашего. И, между нами говоря, будет он при этом прав, т.к. нормальный компилятор C++ должен строго следовать стандарту C++, а приоритет операторов в этом самом стандарте жестко фиксирован.

В голове же программиста жестко зафиксировать взаимное положение более чем 60 (прописью: шестидесяти) операторов представляется проблематичным, так что ничего удивительного, что время от времени одну и ту же конструкцию программист и компилятор могут понимать совсем по-разному. И это еще полбеды! Ведь зачастую трудно понять, какие из замысловатых нагромождений операторов наш электронный

оппонент расшифрует правильно (точнее - так, как хотелось бы нам), а какие - *прав ильно*
(так, как вообще-то надо, но только кому угодно кроме нас)! С одной стороны - сложные взаимоотношения операторов создают почву для появления ошибок, с другой - непонятно, где именно на этой почве взошли нехилые такие всходы! А каждый раз при отладке программы расшифровывать по книжке тайный код операторов сразу во многих местах, не будучи уверен, что вообще там ищешь... Удовольствие то еще.

В общем, картина устрашающая, не так ли? А ведь не допустить подобного удручающего развития событий очень просто - нужно просто явно указывать, что вы хотите сказать - при помощи скобок. Явно указывая, какие из операторов должны выполняться прежде, а какие - позже, вы не допустите разночтений в толковании конструкции вами и компилятором, и притом будете уверены в том, что в данном месте у вас нет ошибок, и не будете тратить свое драгоценное время на копание не там, где надо. Посмотрим для наглядности вот такой пример:

```
int a, b, c, d;
 a = b + c = d;
```

Как вы думаете, что хотел сказать программист? Существует как минимум две трактовки, и в обоих случаях желаемое в этом примере не осуществится.

Итак, трактовка первая: программист хотел присвоить переменной с значение переменной d, а результат сложения b и нового значения с присвоить переменной a. С чем мы его и поздравляем, поскольку компилятор раскроет это выражение не как

```
int a, b, c, d;
 a = (b + (c = d));
```

а как

```
int a, b, c, d;
 a = ((b + c) = d);
```

Ну а может быть, наш незадачливый кодер именно этого и хотел? Пардон, совсем не этого, а вот этого:

```
int a, b, c, d;
 a = ((b + c) == d);
```

Такая конструкция выглядит уже весьма осмысленно, тем более, что в языке С за неимением типа bool применялся тип int.

Как бы там ни было, а все равно такая (без скобок) конструкция будет ошибочна и скомпилирована не будет, т.к. слева от оператора = должно находиться выражение

lvalue
, т.е. нечто, ссылающееся на (неконстантный) объект. Выражение b + c - это некоторое значение типа int, хранящееся во
временной переменной
. Понятно, что такой конструкции присваивать что-либо глупо, и компилятор с нами в этом отношении полностью согласен.

Итак, еще раз повторюсь, что использование скобок делает ваш код гораздо более читабельным, а также помогает предотвратить разногласия между вами и компилятором.

Кстати, еще один совет напоследок. Часто в одной строке может быть очень большое количество скобок, и поэтому очень важно будет не запутаться и соблюсти парность открывающих и закрывающих скобок, причем сделать это не абы как, а как надо. Для этого я советую вам *сначала* напечатать обе скобки (открывающую и закрывающую), и только *потом* - их содержимое. Вы можете со мной не согласиться, но по своему опыту я говорю вам: это работает!

Осмысленные имена

Как вы думаете, что означают переменные с именами a, b и c? Стороны треугольника? Коэффициенты квадратного уравнения? Если нам повезет, то это действительно будет так. И в самом деле, обозначения довольно традиционные, и, если мы знаем, что программа или отдельная функция работает с треугольниками или решает квадратное уравнение, то разобраться в назначении этих переменных нетрудно. А если функция рассчитывает динамику облачных массивов в северном полушарии и имеет такое объявление:

```
Data* f(int a, double b, Data* c);
```

Кто-нибудь сможет сходу сказать, что это за функция и что она делает? Можете назвать меня за это плохим программистом, но я не возьмусь за это. Впрочем, хорошие программисты не дадут соврать: "в нашем деле за такое бьют подсвечниками". Просто по той причине, что иногда приходится играть в такую угадайку на пространстве тысяч строк кода. И если при этом нет никаких комментариев, то сразу понимаешь, что же чувствовал Геракл, созерцая Авгиевы конюшни.

Впрочем, попугал я вас - и будет. На самом деле, все, что я хочу вам сказать - "используйте такие названия переменных, функций, классов, пространств имен и т.п., которые отражали бы смысл поименованных сущностей". Это нетрудно, однако приносит ощутимые результаты.

Есть и тут, правда, небольшая проблема - компилятор C++ воспринимает только латинские символы, а наш живой великорусский ему чужд. Поэтому при подборе названия различных сущностей у вас есть два варианта - называть их по-английски (по-немецки, по-итальянски, по латыни в конце концов) или использовать транслит. Лично я предпочитаю английские названия. Другие языки, во-первых, не знаю я сам,

во-вторых, знает меньшее число людей, чем английский (про китайский мы промолчим). Транслит же на мой взгляд уродлив и зачастую малопонятен. Впрочем, и здесь решать вам, я лишь привел аргументы в пользу своего варианта.

И еще одно замечание по поводу поименования переменных и т.п. Что делать, если название должно содержать несколько слов? Например, "матрица смежности", "максимальный элемент", "цвет шрифта", "имя файла". Писать все в одно слово:

```
int** adjacencymatrix;
 int maximumelement;
 int fontcolor;
 char* filename;
```

неудобно и трудночитаемо. Несколько слов в пределах одного имени необходимо отделять друг от друга, и тут вы можете использовать либо знак подчеркивания '_' между двумя словами, либо начинать каждое новое слово с большой буквы. Я предпочитаю второй вариант:

```
int** AdjacencyMatrix;
 int MaximumElement;
 int FontColor;
 char* FileName;
```

Итак, повторюсь - не забывайте именовать смыслонесущие переменные, классы, функции, пространства имен и т.д. в соответствии с их назначением, желательно оформляя эти названия как можно более читабельно.

Комментарии

Немало об этом уже сказано, и будет сказано еще больше, однако обойти эту тему было бы неправильно.

Поговорим все же о комментариях. Что бы вы о них не думали, комментарии - это один из важнейших инструментов программиста, который не хочет, чтобы его код оставался актуальным только здесь и сейчас, а по прошествии небольшого времени превратился бы в непонятное нечто, лишенное малейшего шанса на развитие. На это существует самое надежное доказательство, и вам оно сейчас будет предъявлено.

Дело в том, что в реальных ситуациях программный код имеет немалые размеры. В самых простых случаях это несколько сотен строк, в более реальных - не меньше тысячи. Программы в десятки и даже сотни тысяч строк - тоже отнюдь не редкость. Естественно, что такие программы за один день не пишутся, да и вообще невозможно держать в памяти полное понимание такого объема кода. Так что, сами понимаете, без комментариев не обойтись никак.

Но! Не всякий комментарий хорош, уместен, безвреден, актуален, понятен и т.д. Вот вам парочка аксиом, касающихся грамотных комментариев:

- Комментарий должен:

1. Помогать в понимании кода программы
2. Относиться непосредственно к коду
3. Быть информативным и понятным
4. Соответствовать истине и быть актуальным
5. Быть там, и только там, где это действительно необходимо

6. Комментарий НЕ должен:

1. Дублировать то, что выражено в коде простыми и ясными конструкциями
2. Быть единственным залогом того, что части программ будут использованы правильно (инициализация, очистка памяти, переменные и функции "только для внутреннего использования" должны поддерживаться самим кодом, а не полагаться на сознательность пользователя, читающего комментарии и документации от А до Я)

3. Написание хороших комментариев - искусство, овладеть которым (как и познать дзен) невозможно в полной мере, но можно и нужно к этому стремиться.
4. Хорошие комментарии помогают понять код, плохие - загромождают его и запутывают читающего.
5. Без хороших комментариев крайне затруднительно написать хорошее приложение.

Немного поясню насчет второй аксиомы, в частности, первого ее подпункта. Что значит "дублировать то, что выражено в коде простыми и ясными инструкциями"? А значит это то, что в C++, как и в любом другом языке программирования, да и в программировании вообще, существует множество шаблонов написания тех или иных логических конструкций (не путать с шаблонами типа template и паттернами программирования!). Это, в частности, перебор массивов, использование счетчиков, выделение памяти и т.п. Чаще всего такие участки кода вообще пишутся на автомате и при чтении являются абсолютно понятны, потому комментировать их смысла не имеет. Также не имеет смысла делать комментарии системы "перевод с C++ на русский". Небольшой пример того, что я имею в виду:

```
int array[10];   //Массив из 10 целых чисел
 for (int i = 0; i                  std::cin >> array[i];   //Вводим их с клавиатуры
 int sum = 0;   //Сумма элементов, начальное значение - ноль
 for (int i = 0; i                  sum += array[i];   //Увеличиваем сумму элементов массива на array[i]
 std::cout
```