Синтаксис: в начале перед объявлением класса напишем слово `template` и укажем параметры в угловых скобках. В нашем примере:

```
template < int ArrayLength, typename SomeValueType >
class SomeClass{
        SomeValueType SomeValue;
        SomeValueType SomeArray[ ArrayLength ];
        ...
};
```

Тогда для первой модели пишем:

```
SomeClass < 20, int > SomeVariable;
```

для второй:

```
SomeClass < 30, double > SomeVariable2;
```

Хотя шаблоны предоставляют краткую форму записи участка кода, на самом деле их использование не сокращает исполняемый код, так как для каждого набора параметров компилятор создаёт отдельный экземпляр функции или класса. Как следствие, исчезает возможность совместного использования скомпилированного кода в рамках разделяемых библиотек.

УДК 371

Войткевич И.С.

КАЧЕСТВО ПРОГРАММНОГО КОДА

БНТУ, Минск
Научный руководитель: Дробыш А.А.

При создании любой программы разработчики беспокоятся прежде всего о ее работоспособности, ведь если приложение не удовлетворяет требованиям заказчика, то вас вряд ли спасут рассказы об изящной внутренней архитектуре, эффективных алгоритмах и других достоинствах вашего детища.

Однако выпуском работающей версии жизненный цикл ПО не ограничивается. В период эксплуатации приложения в нем обнаруживаются ошибки, которые необходимо исправлять, а у пользователей могут появляться новые требования, так что программу приходится дорабатывать и выпускать новые версии.

Тут и встает вопрос о том, насколько трудоемко внесение изменений в уже написанный код, который никто не трогал месяцы, а то и годы. Одно дело – исправлять программу «по горячим следам», пока все участники разработки хорошо представляют ее структуру и схему работы, и совсем другое – модифицировать незнакомый код, в котором еще надо разобраться. Практика показывает, что нередко программисты с трудом продираются сквозь дебри даже собственных творений, созданных несколько лет назад. Что уж говорить о ситуации, когда приходится править чужие программы.

Совокупность характеристик кода, влияющих на трудоемкость его восприятия и внесения в него изменений, принято называть качеством кода. Чем выше качество кода приложения, тем проще его поддерживать и обновлять. Как следствие, разработчики могут сосредоточиться на реализации действительно нового функционала, а не заниматься созданием нового велосипеда только потому, что не смогли починить педаль у старого.

Как же обезопасить себя и коллег от кошмара поддержки нечитаемого кода? За десятилетия развития программной индустрии специалисты набили немало шишек, разбираясь в этом вопросе.

В этом докладе мы постараемся рассмотреть основные факторы, влияющие на качество кода, а также обобщить рекомендации по его улучшению, применяющиеся в различных проектах и компаниях.

Форматирование и оформление кода

Одним из главных аспектов качества кода является его визуальное оформление, за счет которого можно существенно повысить читаемость программы и уже тем самым серьезно облегчить ее поддержку и доработку.

Первым моментом, оговариваемым при согласовании правил оформления кода, является стиль отступов для обособления структурных блоков программы – тел функций, циклов и тому подобного. Всегда необходимо уточнять, используются для таких отступов символы табуляции или пробела, и если используются пробелы, то сколько (как правило, выбор ведется между значениями 2, 4 и 8).

Казалось бы, необходимость использования единообразного стиля отступов по крайней мере в рамках одного файла очевидна, но в реальной жизни часто встречаются смешения стилей, сильно затрудняющие читаемость программы.

Правила форматирования кода включают и ряд других аспектов, единообразие которых влияет на восприятие программы:

Правила именования

Помимо собственно форматирования, простота восприятия кода во многом зависит от способа именования переменных, функций, членов классов и прочих сущностей.

Использование констант

Помимо переменных, в большинстве программ используются и константные значения, неизменяемые в ходе работы. Однако использовать их в коде напрямую следует с осторожностью. Вместо этого следует объявить константу с осмысленным именем (если язык программирования не позволяет определять константы явно, то можно обойтись и простой переменной) и использовать это имя.

Ведь у человека, просматривающего код, может возникнуть вопрос: почему используются именно это число, строка или другое выражение и откуда оно взялось? Конечно, можно это

указать в комментарии, но если фиксированные значения мелькают во многих местах, то комментировать их все вряд ли разумно.

Размер структурных блоков

Разработчики давно пришли к выводу, что для лучшей управляемости и понятности код программы лучше разбивать на относительно независимые блоки – например, на процедуры и функции в процедурной парадигме программирования или на классы и их методы в ООП. Но насколько мелкими или крупными должны быть тела функций и методов?

Документированность кода

Широко известно следующее полушуточное утверждение, что хорошо написанный код в комментариях не нуждается – он очевиден сам по себе. Конечно, к такому положению вещей стоит стремиться, но далеко не всегда это получается, и без текстовых подсказок на «человеческом» языке не обойтись.

Впрочем, избыточное комментирование также не является хорошей практикой – согласитесь, не очень удобно изучать код, каждые две строки которого предваряются десятком строк комментариев.

Инструментальная поддержка

Один из главных шагов на пути к высокому качеству кода – это настройка используемой IDE или текстового редактора. Современные среды разработки и продвинутые редакторы содержат множество опций, влияющих на форматирование и оформление текста программы.

Запуск всевозможных анализаторов – дело полезное, но само по себе рутинное. Не удивительно, что в больших проектах оно осуществляется автоматически. В частности, инструменты непрерывной интеграции (например, Hudson и Jenkins) предоставляют плагины для запуска различных утилит анализа кода и удобного представления их результатов.

Впрочем, автоматизация – это хорошо, но многие огрехи никакой инструмент исправить не в состоянии, и лучше изначально следовать правилам оформления кода, чем потом работать над его приведением в порядок.

Итак, хорошо оформленный и читаемый код – необходимый фактор простоты поддержки программного продукта, существенно облегчающий жизнь разработчиков. Ведь программист – достаточно мобильная профессия, представителям которой приходится часто переходить из одной команды в другую и переключаться с проекта на проект. Соответственно один и тот же человек за свою профессиональную карьеру может поучаствовать во множестве проектов, а над одним и тем же кодом за время его жизненного цикла может поработать большое количество программистов. Следование стандартам существенно облегчает эти процессы, позволяя людям сразу концентрироваться непосредственно на работе, а не на изучении доставшегося в наследство кода и притирке к стилю предшественников и товарищей по команде.

При работе в команде необходимо придерживаться единого стиля, даже в ущерб собственным вкусам. Если вы приходите в проект или компанию с богатой историей, то выбирать скорее всего не придется – вас просто поставят перед фактом, какой именно набор правил следует использовать. А вот в учебных командных проектах (если таковые предусмотрены в вашем учебном заведении) такие правила необходимо выработать самостоятельно перед началом написания кода. Однако брать эти требования с потолка не стоит – для всех популярных языков программирования уже имеются более-менее стандартные правила, на них и надо основываться. Отлично сформулировали разработчики Google в своем C++ Style Guide – «придерживайтесь здравого смысла и будьте последовательны».