**Прикладной или системный?**

Евгений Каратаев

В разработке программы встречаются неясные вопросы относительно того, как спланировать тот или иной код, куда его отнести и как классифицировать. Код может быть классифицирован по самым различным классификациям - на код верхнего и нижнего уровня, на специфичный и библиотечный, и другим.

Правильная классификация зачастую на первых же шагах разработки (как проектирования, так и реализации) может помочь в выявлении ошибок. Если читаешь схему данных или код, предназначенный для выполнения определенной задачи, которая классифицируется определенным образом и при этом код имеет признаки классификации не соответствующей задаче, то у меня срабатывает рефлекс и этому коду (схеме, диаграмме) я сразу приписываю проблемный крестик. Независимо от текущего положения дел расхождение классификационных признаков задачи и реализации есть индикатор по крайней мере потенциальной ошибки или мины, которая рано или поздно сработает.

Одним из предметов споров иногда может выступать деление как программистов, так и кода на прикладных и системных. К системным программистам традиционно относят разработчиков операционок, драйверов, инструмента, а остальных - к прикладным. Здесь я опишу свое видение классификации кода по признаку - прикладной или системный.

Сразу можно помолчать, выслушать кучу вопросов и попыток уточнения вроде "А смотря что понимать под прикладным". Вот в таком виде, чтобы было понятно без подобных уточнений, и поведем дальше речь.

Одна из основных аксиом программирования гласит, что программа есть виртуальная машина для преобразования или передачи информации, сама по себе являющаяся информационной сущностью. Программа по отношению к одному слою является переработчиком информации, а по отношению к другому слою (операционная система) - перерабатываемая информация. Таким образом, рассматривая основные способы построения программ как сущностей в контексте операционной системы, можно утверждать, что существует объект, который обрабатывает программу и поддерживает ее функционирование.

Связывание же двух противоположных сущностей - алгоритмов как неизменяемых и исполняемых информационных объектов и данных как изменяемых и неисполняемых объектов производится через определенные соглашения о вызове. Эти соглашения называются декларацией типа и описывают, каким именно образом алгоритм должен получать доступ к обрабатываемой памяти. Прекрасной иллюстрацией может служить использование в языках высокого уровня различения между знаковым и беззнаковым целым. Программист для обоих типов может использовать одинаковые операции, например, сложение. Но при этом транслятор, имея сведения о типе целого (знаковый или беззнаковый), генерирует различный машинный код.

Дальнейшим развитием понятия типа является понятие класса в объектно - ориентированных языках. Если говорить о С++, то на нем программист может переопределить в том числе и встроенные операции языка, например, определить операцию сложения для строк. Синтаксически использоваться же будет по-прежнему оператор сложения, но транслятор, имея информацию о типе, будет генерировать код, описанный программистом в перегруженном операторе.

Несложно видеть, что реализация типизированного доступа в обоих случаях опирается на "используя информацию о типе". Независимо от того, как именно реализуется поддержка типов - на этапе генерации кода, с применением таблиц виртуальных функций или присоединением теговых данных, где-то все равно должна существовать соответствующая информация. Именно эта информация и определяет поведение исполнительного механизма доступа. Таким образом, обрабатываемая информация описывается как собственно ее содержанием, так и ее форматом доступа.

Довольно общеизвестные сведения, приведенные выше, нужно рассматривать как освежение знаний и погружение в контекст вопроса. Теперь о сути темы. Мое мнение по вопросу разделения кода на системный или прикладной состоит в правиле - если исполнение кода и его течение определяется форматом данных, то это системный код, а если код зависит от содержания данных, то это прикладной код. Правило очень простое, хотя и, может быть, спорное. Тем не менее мне оно помогает в планировании модулей и в проектировании.

Как различить код, зависящий от значений данных и не зависящий от значений данных. Просто надо посмотреть точки ветвлений программы (функции), условных переходов и циклов и определить, от чего зависит выполнение условия. Если от значения данных, к тому же обрабатываемых, то код должен быть отнесен к прикладному. Если только от неких констант, типов, данных описывающих формат других данных, то это системный код. Более того - такая оценка может выявить очевидные логические ошибки. Например, если в коде написано, что если параметр беззнаковый, то делать одно, а если больше 12, то другое. Приведем несколько примеров, демонстрирующих различение прикладного и системного контекстов.

string func( string str)

{

    int len = strlen( str.c\_str());

    for( int i = 0; i < len; i++)

        str[i] = WinToKoi( str[i]);

    return str;

}

В этом примере код не зависит от значения символов в строке. Какие бы они ни были, программа будет исполняться точно так же. Таким образом, этот код является системным.

string func( int sum)

{

    if( sum < 10)

        return "мало";

    if( sum < 100)

        return "нормально";

    if( sum < 1000)

        return "много";

    return "очень много";

}

В этом коде выполнение алгоритма явно зависит от значений данных. Поэтому такой код я классифицирую как прикладной.

При разработке программы я бы счел хорошим стилем размещение приведенных функций в разных модулях, с тем, чтобы существовали модули, отвечающие за прикладную часть и за системную часть. Впрочем, для небольшой программы размещение таких функций в одном модуле может быть вполне оправдано.

Кроме приведенного правила классификации кода на системный и прикладной следует обращать внимание на смешанный случай, существующий в явном виде при составлении функций, оперирующих визуальным представлением данных. При визуализации данных следует учитывать существование национальных стандартов представления данных, которые могут различаться так же и форматом представления. Например, при визуализации времени по российскому стандарту следует указывать час, минуту и секунду, а при визуализации по английскому стандарту следует дописывать еще и символы представления "AM/PM". Таким образом, код независящий от значения данных, начинает зависеть от контекста его работы, который есть данные. Еще более сложный случай - формирование визуализации строкового представления с применением особенностей языка, например "сумма прописью". К какой категории отнести такую функцию? Думаю, что к прикладной.

Приведенное правило классификации, конечно, не есть догма. Но лично я стараюсь придерживаться таких простых правил в работе и они мне помогают в проектировании и отыскании ошибок.