Язык Си и начальное обучение программированию

А. В. Столяров

30 января 2010 г.

Аннотация

Недавно коллеги задали мне вопрос, как я отношусь к идее замены языка Паскаль на первом курсе языком Си. Когда я категорично ответил «ни в коем случае», меня спросили «а почему?» Предлагаемый текст — мой ответ на этот вопрос.

1 Предисловие

Язык Си, созданный в первой половине 1970-х годов, стал, как известно, подлинным прорывом в области программирования. Внезапно программисты обнаружили, что в некоторых областях, считавшихся до той поры сугубо «ассемблерными», таких как ядра операционных систем, прошивки ЭВМ специального назначения и т. п., автокоды и языки ассемблеров вдруг утратили монопольное положение, а сами операционные системы стало возможно создавать переносимыми с одной аппаратной архитектуры на другую. В настоящее время язык Си, история которого перевалила за 35-летний рубеж, используется для создания широчайшего спектра программ, от прошивок микроконтроллеров до офисных пакетов; его компиляторы доступны для практически всех существующих архитектур процессоров; объем программного кода, написанного на Си за долгую историю его существования, составляет заметную долю всех текстов программ, написанных человечеством. Язык прочно удерживает первое место по популярности, и в особенности это заметно в мире свободно распространяемых программ. Знание языка Си неявно, но от этого не менее категорично считается обязательным для всех профессиональных программистов вне зависимости от их специализации; представляется сомнительным, чтобы программист, не знающий Си, смог вообще найти работу по специальности. Пожалуй, в этом Си уникален: ни про какой иной язык программирования такого сказать нельзя.

Обучение студентов программистских специальностей языку Си, вне всякого сомнения, представляется мне обязательным; не обсуждая этот вопрос (в силу очевидности ответа на него), я хотел бы в предлагаемом эссе уделить внимание вопросу о возможности **начинать** обучение студентов именно с этого языка.

2 На кого ориентироваться

Одной из основных особенностей аудитории, если говорить о первокурсниках, является практически полное отсутствие базовой подготовки по программированию. Информатика в школах чаще всего не преподается вообще (по рассказам многих родителей нынешних школьников, чаще всего «уроки» информатики в школе представляют собой 45 минут компьютерных игр), а если преподается, то на столь низком уровне, что чудес из этой области можно не ждать. Введение ЕГЭ по информатике проблемы, скорее всего, не решит. Во-первых, преподавать информатику в школах просто некому: люди, одновременно умеющие и программировать, и учить, вообще встречаются чрезвычайно редко, а среди российских школьных учителей не встречаются никогда. Во-вторых, для школьной информатики начисто отсутствует методическая база. Мне пришлось за последние несколько лет рецензировать рукописи, предлагавшиеся к изданию в качестве школьных учебников информатики; судя по этим рукописям, ситуация в этой области плачевная, если не сказать катастрофическая. Относительно **всех** без исключения текстов, попавших ко мне на рецензию, я могу заявить совершенно определенно, что их авторы о предмете программирования не имеют ни малейшего представления. Единственный учебник, который я смог одобрить, отличался от остальных тем, что его автор даже не пытался излагать программирование; текст был посвящен общим вопросам информатики, истории информации, письменности, арифметики, уделялось внимание началам теории кодирования, логики и т. п. но не программированию. Наконец, даже если удастся заставить (например, с помощью ЕГЭ по информатике) школьных учителей исполнять свои обязанности на сколь бы то ни было должном уровне, не следует забывать, например, что динамические структуры данных (и вообще понятие указателя) в школьный курс информатики не входит, считаясь неким высшим пилотажем.

Несомненно, среди студентов, поступающих на первый курс, имеется небольшой, но достаточно стабильный процент тех, кто, невзирая на все вышеизложенное, программировать уже умеет. Я сам, поступив на факультет в 1992 году, имел к тому времени опыт получения денег за написанные мной программы, и в те времена это не было редкостью: только в одной со мной группе училось не менее четырех человек, ни в чем не уступавших мне в области программирования. Сейчас экономическая ситуация не столь экстремальна, как в начале девяностых, так что количество состоявшихся профессионалов среди поступающих на факультет стало меньше, но они, тем не менее, есть. Однако следует ли ориентироваться на них при составлении программ программистских курсов? Прежде чем дать ответ на этот вопрос, отмечу одно обстоятельство, неизменно ускользающее от внимания публики. До поступления на факультет этих людей учить программированию было некому, и, следовательно, *если вчерашний абитуриент умеет программировать — это прежде всего означает, что программированию он может учиться самостоятельно, без всякой помощи со стороны преподавателей.* Иначе говоря, такой студент не нуждается в том, чтобы его «учили программировать» — рассказывали синтаксис языков программирования, показывали всяческие алгоритмы, приемы и техники. Как ни

странно, чаще такие студенты нуждаются в совершенно ином, а именно — чтобы кто-нибудь убедил их, что их без пяти минут профессиональные навыки — это еще не все, что есть в мире, и что умение писать программы как таковое еще не делает человека программистом. Однако мир, к сожалению, не идеален, и у некоторых преподавателей попросту не хватает квалификации (не преподавательской, а профессионально-программистской), чтобы поддерживать на должном уровне свой авторитет среди студентов рассматриваемой категории. Лучшее, что можно сделать в таком случае — это не мешать студенту учиться самостоятельно и не прививать отвращение к предмету.

Итак, среди первокурсников есть прослойка тех, кто умеет программировать (в той или иной степени), но эта прослойка, по-первых, малочисленна, и, во-вторых, как раз эти студенты в услугах преподавателей (по программистским предметам) нуждаются меньше всего. Если в своей работе мы будем ориентироваться на них, то от нашей деятельности окажется на удивление мало пользы: *эти* студенты без нашей заботы вполне обошлись бы, а остальные на занятиях ничего не поймут (то есть вообще) и нам придется смириться с тем, что подавляющему большинству студентов по программистским предметам ставятся безобразно натянутые «тройки», просто чтобы не выгонять почти весь курс. В этот производственный брак попадут и те студенты, кто потенциально мог бы прекрасно программировать, просто им не повезло со школой и окружением.

Приходится сделать неизбежный, хотя, возможно, и не очень приятный вывод: расчитывать следует на *средний* уровень студентов, а это, в частности, означает, что программировать наша целевая аудитория не умеет вообще ни в каком виде и ни до какой степени, так что начинать обучение программированию приходится **действительно с нуля**, и эта ситуация в обозримом будущем никак не изменится.

3 «Hello, world», или барьер, который возьмут не все

Как дорога начинается с первого шага, так и обучение начинается с первого занятия. Если начинать рассказ о языке программирования с долгих перечислений операторов, операций, типаов, переменных и прочих составляющих языка, слушатели эту информацию не воспримут, поскольку не будут понимать, о чем вообще идет речь. Чтобы это стало понятно, систематическое изложение необходимо предварить неким введением в предмет, в ходе которого показать, как вообще выглядят программы на том языке программирования, который мы начинаем рассматривать.

По традиции, введенной авторами Си Брайаном Керниганом и Денисом Ритчи, обучение этому языку начинают с программы «Hello, world». Вот ее текст[1]:

[1]На всякий случай отмечу, что, согласно современным представлениям, функция `main` **не может** иметь никакой тип, кроме `int`, в том числе не может она и быть `void`'овой, ну а в функции, возвращающей значение, присутствие оператора `return` обязательно. Программа, в которой опущено слово `int` перед `main` и оператор `return`, откомпилируется с двумя предупреждениями, что, конечно же, недопустимо в учебных примерах.

```
#include <stdio.h>
int main() {
    printf("Hello, world\n");
    return 0;
}
```

Написать этот текст на доске не сложно, сложности начинаются в тот момент, когда нарисована последняя фигурная скобка и необходимо переходить к пояснениям. Что это там в первой строке? Правильный ответ звучит примерно так: это директива макропроцессора, которая включает в нашу трансляцию специальный «заголовочный» файл, содержащий объявления библиотечных функций. А теперь давайте вспомним, что мы проводим *первое* занятие для абсолютно неподготовленной аудитории. О смысле слова «директива» наши слушатели еще могут догадаться из общих соображений и окажутся недалеки от истины, но остаток фразы — макропроцессор, функции (к тому же еще «библиотечные»), объявления... все эти слова не оставляют слушателям ни единого шанса на понимание. Если в этот момент пуститься в пространные объяснения, мы можем где-то через полчаса с удивлением обнаружить, что увлеченно рассказываем, например, о конвенциях вызовов функций, в то время как наши слушатели читают книжки, играют в тетрис на мобильных телефонах, рисуют в тетрадках цветочки и занимаются другими делами, имеющими столь же прямое отношение к программированию. Реальность такова, что объяснить неподготовленному слушателю смысл директивы `include` **невозможно**, нравится это нам или нет. Поэтому приходится произнести сакраментальное «так надо, а зачем — мы узнаем позже». Заметим, мы еще ничего не объяснили, а ссылка вперед нам уже потребовалась.

Дальше — больше. Что такое `int`? Что такое `main`? Правильный ответ таков: `main` — это имя главной функции нашей программы, а `int` — тип значения, которое она возвращает. Конечно же, это очень просто, но только не для человека, который вообще не умеет программировать. Слово «функция» он раньше встречал только в математическом смысле, ну и еще в общезначимом (функция утюга — гладить); к нашему случаю не подходит ни то, ни другое. Термин «возврат значения» слушателю и вовсе представляется странно звучащей бессмыслицей, а «тип значения» окончательно добьет тех, кому до сей поры еще казалось, что они что-то понимают. Все, 90% аудитории потеряно, и с нами остались только те, кто уже программировал (хотя бы на каком-нибудь языке, где были типы и функции).

Частично спасти положение можно еще одной сакраментальной фразой: «если это непонятно, ничего страшного, скоро все станет понятно». Внимание некоторой части аудитории это нам вернет. Правда, ненадолго, но пояснить следующую строку (к счастью, это единственная более-менее понятная строка в программе) мы успеем. Итак, следующая строка печатает фразу «Hello, world» на экране. У этого утверждения есть одно достоинство: оно понятно. Вместе с тем, оно под завязку наполнено фактическими ошибками и недоговорками, если же попытаться их устранить, у нас опять получится целый ряд ссылок вперед. Чего стоит одна только комбинация «`\n`», обозначающая символ перевода строки. Большинство нынешних студентов никогда не видели пишушей машинки, во

всяком случае, в действии, да и матричных принтеров тоже уже не застали, а с компьютером работали только в графическом режиме, так что вот это вот понятие «символ перевода строки» им придется долго разъяснять. Ну то есть что такое «перевод строки», наверное, поймут все, но как ЭТО может быть символом?! Конечно, все это станет понятно после первых же двух-трех часов работы с эмулятором алфавитно-цифрового терминала, но не забывайте, что сейчас мы проводим **первое** занятие, и знакомство с терминалом у студентов еще впереди.

Разъяснив с грехом пополам строчку, в которой вызывается `printf`, мы будем вынуждены перейти к следующей строке, в которой опять возникнет пресловутый «возврат значения из функции». Вне всякого сомнения, это очень простое и очень важное для программирования понятие, но объяснять его на **этом** примере — идея вряд ли удачная. Даже если мы как-то объясним, что же такое «функция» в терминах языка Си, слушатель вряд ли поймет, какое это имеет отношение к программе, печатающей строку; ну а загадочный ноль в качестве параметра потребует экскурса в управление процессами, чтобы объяснить, что это такое за загадочный зверь по имени «код завершения задачи».

Посмотрим теперь, как все то же самое выглядит на Паскале. Текст программы будет таким:

```
program hello;
begin
    WriteLn('Hello, world');
end.
```

При ее объяснении не потребуется никаких особо сложных концепций, никаких ссылок вперед. Слова «начало» и «конец» достаточно перевести с английского и сказать слушателям, что таковы правила языка Паскаль; мы нимало не погрешим здесь против истины, поскольку и заголовок программы, и то, что все ее операторы следует поставить между словами `begin` и `end` — это именно что синтаксические соглашения, ничем особенным не обусловленные и не являющиеся частным случаем каких-либо более общих (и, значит, более сложных) правил. Можно даже не уточнять, что `begin` и `end` — это пресловутые «операторные скобки», потому что в данном случае они выступают не в этой роли. Более того, процедура `WriteLn` переходит на печати на следующую строку после распечатки своих аргументов (это слушатели поймут), и страшноватая для неподготовленного уха концепция *символа* перевода строки оказывается за кадром — до наступления удобного момента.

Но как же быть с Си, ведь все его «сложности» никуда не денутся, а рассказывать его студентам рано или поздно придется? Так-то оно так, но если ту же самую программу «Hello, world» показать и объяснить студенту, уже занимавшемуся программированием (на том же Паскале или на чем-то еще), мы сможем избежать практически всех трудностей, перечисленных выше. С понятием «макропроцессор» студент уже знаком, а если не знаком — то можно ему сказать (не упоминая макропроцессор), что директива `#include` подставляет вместо себя содержимое файла, в данном случае это файл `stdio.h`, прилагающийся к стандартной библиотеке, он тоже написан на Си и содержит информацию для

компилятора о том, какие библиотечные функции есть в этой библиотеке. Понятие библиотеки студенту уже известно, понятие функции в программистском смысле — тоже (попутно уточняем, что в Си, в отличие от Паскаля, есть только функции, а процедур нет — и нас понимают). Слово `int` обозначает целочисленный тип выражений — и нас снова понимают, потому что со всеми этими терминами уже не раз встречались. Слово `main` — это имя функции, попутно уточняем, что главной программы, в отличие от Паскаля, здесь нет, программа вся состоит из функций, просто есть соглашение, что одна из них — а именно вот эта вот `main` — вызывается при старте программы. Никаких проблем не вызывает и `return`, и замечание, что `printf` представляет собой библиотечную функцию (попутно уточняем, что в Паскале `WriteLn` есть часть языка, тогда как здесь мы можем `printf` написать сами). Да и с символом перевода строки наш подготовленный слушатель, вне всякого сомнения, уже сталкивался. Совсем, как говорится, другой коленкор — а все потому, что на этот раз Си для наших слушателей не первый в жизни язык программирования.

4 Второй пример программы: барьер, который не возьмет никто

Обычно программы не ограничиваются выводом информации: в общем случае, да и в большинстве частных, алгоритм (и программа, как запись алгоритма) представляет собой преобразователь исходных данных в данные, составляющие результат. Вполне естественно поэтому, что уже вторая программа, предъявляемая слушателям в качестве примера, будет содержать ввод данных извне.

Совершенно не сговариваясь между собой, многие авторы пособий в качестве такого второго примера используют решение квадратного уравнения. Пример действительно удачный, он позволяет продемонстрировать переменные разных типов, арифметику, ветвление; вычисление дискриминанта можно вынести в отдельную функцию, заодно показав, как на Си пишутся функции и как из них возвращать значение (после этого слушателям становится проще понять, что собой в действительности представляет `main`). Наконец, необходимость использования функции `sqrt` заставляет нас сначала подключить второй заголовочный файл (в этот раз `math.h`), пояснив, что прототип функции `sqrt` находится именно в нем; затем приходится отметить, что функция сама по себе не входит в стандартную библиотеку, так что при сборке программы необходимо эту библиотеку подключить, используя параметры командной строки компилятора. Впрочем, многие из этих пояснений можно и опустить без особого ущерба для изложения; речь же сейчас пойдет о другом.

Чтобы *получить исходные данные* (в рассматриваемом случае — коэффициенты уравнения) извне, нам в языке Си придется воспользоваться функцией `scanf`. Квадратное уравнение тут ничем особенным не примечательно, ввод неких числовых данных «с клавиатуры» (то есть из стандартного потока ввода) предполагается большинством учебных заданий для начинающих. И здесь мы сталкиваемся с отсутствием в Си передачи параметров иначе как по значению.

Описываем переменные (тут все в порядке), выводим приглашение к вводу (тут тоже все понятно, с `printf`'ом студенты уже знакомы), начинаем писать следующую строку... и в процессе ее написания понимаем, что теперь нам предстоит объяснить аудитории нечто, для понимания чего слушателям не просто не хватает знаний — им не хватает примерно целого семестра этих знаний. А строка выглядит так:

```
scanf("%lf %lf %lf", &a, &b, &c);
```

Объяснить концепцию «форматной строки» еще возможно, хотя нельзя не отметить, что даже на втором курсе далеко не все студенты с первого раза понимают, о чем идет речь. Но вот зачем перед именами переменных стоит знак «`&`», неподготовленный слушатель не поймет, что бы вы ни делали и к каким хитростям ни прибегали. Концепция «указателей» и адресных выражений кажется простой лишь тем, кто с ней давно и прочно знаком. Каждый, кто хотя бы раз объяснял работу с динамической памятью людям, имеющим нулевой уровень подготовки, знает, что это на самом деле сложнейшая педагогическая задача — как всегда бывает в случаях, когда необходимо сформировать в мышлении ученика новый уровень абстракции, которого там раньше не было. Можно, конечно, произнести несколько раз что-то вроде «у каждой переменной есть то место в памяти, где она располагается, и это место обозначается адресом; чтобы функция `scanf` знала, куда занести значения, которые она прочитает, мы передаем ей адреса наших переменных, и вот знак амперсанда как раз и есть операция взятия адреса» и тешить себя надеждой, что мы все понятно объяснили, да только это будет не более чем самообман. Понять нас в данном случае есть шанс лишь у тех, кто уже сталкивался с указателями, причем даже среди этих (весьма немногочисленных) первокурсников нас поймут не все.

Остается лишь поднять белый флаг и сказать уже порядком поднадоевшее и нам, и студентам «так надо, а почему — поймете потом». Студенты, разумеется, просто *запомнят* (читай — зазубрят), что перед переменными в `scanf` надо ставить знак «`&`», и будут очень удивлены, обнаружив, что это правило почему-то не работает при вводе текстовых строк с использованием `%s`. О массивах и адресной арифметике разговор еще впереди, пока же отметим, что Паскаль позволяет не упоминать о явной работе с адресами до тех пор, пока мы не решим (сами, в силу поставленной перед собой учебной задачи) перейти к изложению динамических структур данных, что, в свою очередь, делается ближе к концу первого семестра, когда студенты в основной своей массе к этому уже более-менее готовы. С другой стороны, при изложении языка Си студентам второго курса можно опираться на уже имеющуюся у них понятийную базу, припомнить `var`-параметры Паскаля, пояснить, что здесь их нет и приходится поэтому прибегать к передаче адресов. Поскольку и с указателями, и с возвратом значений через параметры ко второму курсу студенты уже знакомы, проблемы с пониманием если и возникают, то не столь фатальные.

5 Хак как стиль мышления

Возможно, кто-то из моих коллег, которым адресовано это эссе, сочтет, что сложности первого занятия так или иначе можно преодолеть; я соглашусь, что это действительно так, ведь существуют даже школы с углубленным изучением информатики, где каким-то образом ухитряются вбить Си в головы ученикам старших классов. Конечно, барьер минимального понимания тут оказывается очень высок, потому что для изучающего программирование на примере Си не понять указатели означает не понять ничего (то есть вообще ничего), а указатели понимает, увы, даже не всякий среди поступивших на ВМК; такие студенты часто становятся жертвами зачетных комиссий третьего и четвертого семестра, но некоторые все же выживают и продолжают «обучение», чтобы потом никогда даже не помышлять о работе по специальности. В этом плане основы программирования, излагаемые в рамках школьной информатики, играют не столь критическую роль: далеко не каждый школьник собирается становиться программистом, и не понять ничего в отдельно взятом школьном предмете — это не так уж страшно; в профильном ВУЗе ситуация совершенно иная.

Так или иначе, допустим, что трудности, описанные в двух предыдущих параграфах, благополучно преодолены. Это никак не отменяет того факта, что уровень подготовки бóльшей части аудитории остался нулевым и, следовательно, именно материал первого семестра закладывает у студентов основы стиля программирования и программистского мышления. И в этом плане язык Си предлагает широчайший ассортимент средств, прекрасно пригодных, чтобы мышление ученика раз и навсегда искалечить.

В первой половине девяностых появился шуточный текст, написанный якобы от лица Кена Томпсона, в котором говорится, что Си и Unix — это просто первоапрельская шутка, и что их создатели (Брайан Керниган, Деннис Ритчи и Кен Томпсон) никак не ожидали, что им удастся столь качественно разыграть все мировое программистское сообщество. В частности, в этом тексте есть такие строки:

> Когда мы обнаружили, что другие действительно пытаются писать программы на A, мы быстро добавили еще парочку хитрых примочек, создав B, BCPL, и, наконец, Си. Мы остановились, добившись успешной компиляции следующего:
>
> ```
> for(;P("\n"),R-;P("""))for(e=C;e-;P("_"+(*u++/8)%2))P("" "+(*u/4)%2);
> ```
>
> Мы не могли даже представить, что современные программисты будут пытаться использовать язык, допускающий подобный оператор!

Конечно, это всего лишь шутка; известно, что на самом деле язык B был наследником языка BCPL, а не наоборот, и именно из него получился Си, а язык A в этой истории вообще никак не замешан. Однако, как говорится, в каждой шутке есть доля шутки, а все остальное — правда. Так, например, многие приверженцы языка Си убеждены, что конструкция

```
while(*p++=*q++);
```

представляет собой образец подлинного изящества. В условное выражение цикла `while` здесь вставлено три побочных эффекта (сдвиг одного указателя, сдвиг второго указателя, присваивание), тело цикла в результате оказалось пустым, поскольку вся функциональность помещена в условное выражение, ну а сама по себе эта конструкция предназначена для копирования строки.

В качестве другого образчика программистского мастерства приведу следующее:

```
while(~wait(NULL));
```

Поясню, что системный вызов `wait`, предназначенный, чтобы дождаться завершения порожденного процесса, возвращает (как и практически все системные вызовы ОС Unix) значение `-1` как индикатор ошибки. Двоичное представление числа `-1` представляет собой цепочку из двоичных единиц во всех разрядах регистра или переменной; вспомнив, что в Си числом `0` обозначается логическая ложь, а любое другое число считается обозначением логической истины, мы сможем догадаться, что условное выражение, использующее побитовую инверсию (символ «~»), будет ложным с случае, если вызов вернет ошибку, и истинным во всех остальных случаях. Приведенная конструкция, таким образом, означает «выполнять `wait`, пока он не вернет ошибку», и предназначена, чтобы дождаться завершения всех имеющихся в настоящий момент порожденных процессов. Аналогичным образом часто поступают и с другими вызовами, используя побитовую инверсию, чтобы `-1` превратить в логическую ложь; конструкция вроде

```
while(wait(NULL) != -1) {}
```

делающая абсолютно то же самое, видимо, кажется истинным почитателям Си слишком банальной.

Конечно, никто не заставляет учить студентов таким выкрутасам; более того, студентов можно и нужно предостеречь от использования подобных трюков. Давайте, однако, рассмотрим другой случай. Существует целый класс программ, называемых «фильтрами»; эти программы принимают некий текст из стандартного потока ввода, производят над ним те или иные преобразования и полученный результат выдают в стандартный поток вывода; как правило, они продолжают работать, пока в стандартном потоке ввода не возникнет ситуация «конец файла». Поскольку многие задачи, связанные с разбором и преобразованием текста, можно решить с помощью регулярных автоматов или вариаций на тему оных, чаще всего такая программа на верхнем уровне управления представляет собой цикл посимвольного чтения, выполняемого до возникновения ситуации «конец файла». Записывается такой цикл обычно следующим образом:

```
while((c = getchar()) != EOF) {
    /* тело цикла */
}
```

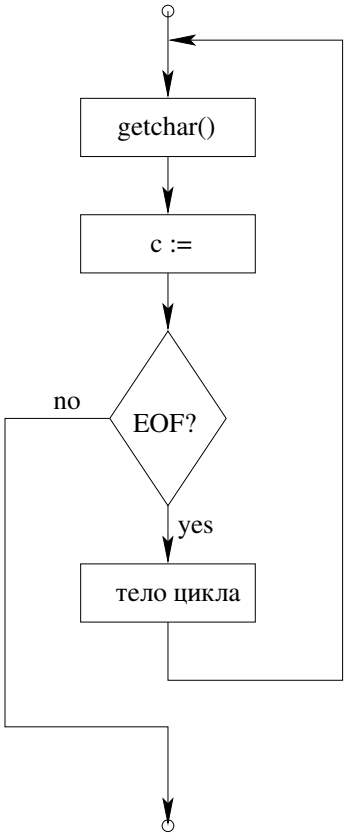

Рис. 1: Блоксхема традиционного цикла с `getchar`

Заголовок цикла `while`, построенный таким способом, можно встретить практически в любом учебнике по Си. Профессиональным программистам эта конструкция знакома и привычна, и никто даже не замечает, что она тоже представляет собою ни что иное, как хак. Чтобы убедиться в этом, рассмотрим блок-схему получившегося цикла (см. рис. 1). Что это, простите? Цикл с предусловием или с постусловием? Или структурное программирование уже отменили?

Итак, мы имеем еще один хак. Однако, в отличие от двух предыдущих примеров, представляющих собой не более чем бессмысленное лихачество, в этом случае не все так просто. Чтобы привести этот цикл к каноническому виду, нам придется применить в тексте два совершенно одинаковых оператора:

```
c = getchar();
while(c != EOF) {
    /* тело цикла */
    c = getchar();
}
```

С концептуальной точки зрения этот текст, конечно, чище, но любого профессионального программиста он заставит недовольно скривиться, и есть от чего: ведь в Си присваивание сделали операцией, а не оператором, специально для таких случаев, а практическое программирование, как это часто бывает, диктует свои условия и не желает приносить лаконичность кода в жертву пусть

и красивым, но все-таки теоретическим канонам. Надо отметить, что я и сам использую в своих программах побочные эффекты в заголовках операторов, и в примерах, приводимых мной на лекциях и семинарах, встречаются такие приемы. Более того, еще несколько лет назад я вообще не обращал на это никакого внимания, воспринимая побочные эффекты в заголовках как нечто само собой разумеющееся, пока не заметил, что некоторые студенты из числа слабых не могут понять объяснения, например, управления процессами в ОС Unix именно потому, что в примерах используется что-то вроде

```
if(fork() == 0) {
    /* ... */
```

Применение вместо этого более подробного фрагмента, а именно

```
int pid;
pid = fork();
if(pid == 0) {
    /* ... */
```

к моему немалому удивлению **уменьшает в группе долю студентов, не понявших, что такое** `fork()`. А натолкнуло меня на мысль о трудностях понимания `if(fork()==0)` то обстоятельство, что на зачетных комиссиях и экзаменах в третьем семестре я несколько раз обнаруживал, например, такое:

```
fork();
if(fork() == 0) {
    /* ... */
```

Очевидно, что ошибка такого рода свидетельствует о том, что студент просто не владеет концепцией побочного эффекта.

Сделав для себя определенные методологические выводы, я теперь при объяснениях нового материала никогда не применяю побочные эффекты в заголовках `if`, `while` и других операторов; тем не менее, в дальнейших примерах я возвращаюсь к использованию побочных эффектов, делая на первых порах замечание, что это просто сокращенная и более элегантная запись того же самого. Полностью отказаться от использования этой техники мне представляется неправильным, поскольку это противоречит сложившейся культуре программирования на языке Си и может сформировать неправильное отношение к возможностям этого языка. Более того, я убежден, что пытаться оградить студентов от «вредных» знаний — занятие, заведомо обреченное на неудачу, ведь соответствующие хаки встретятся им в литературе, в Интернете, в примерах, приводимых другими преподавателями, а иногда и другими студентами. Применение трюков в некоторых случаях есть часть, как я уже сказал, культуры, сложившейся вокруг языка Си, и при изучении этого языка отречься от всей этой культуры все равно не получится.

Однако не следует забывать, что это делается уже на втором курсе, когда за плечами у аудитории уже имеется Паскаль и язык ассемблера. Это означает, во-первых, что большинство студентов к этому времени все-таки чувствует

на интуитивном уровне, что такое «побочный эффект» и как с этим работать. Во-вторых, и это даже важнее, **благодаря имеющемуся опыту программирования на Паскале студенты имеют возможность отличить трюк грязный и ненужный**, вроде вышеописанной побитовой инверсии, **от трюка, применение которого оправдано**. Чтобы уметь различать, что хорошо, а что плохо, нужно уже уметь программировать.

6 Инструменты, которых нет

В контексте фундаментального университетского образования овладение конкретным языком программирования не следует рассматривать как основную цель обучения; языки и другие инструменты меняются со временем, устаревают, к тому же далеко не все студенты, обучающиеся на «программистских» специальностях, в будущем становятся именно программистами, большинство же тех, кто становится, способны (как это уже отмечалось выше) освоить практически любой язык программирования самостятельно. Целью университетских курсов должно быть овладение общими принципами, лежащими в основе дисциплины, и в этом плане конкретный язык программирования, конкретная операционная среда и т. п. — не более чем иллюстративный материал.

Основной лекционный программистский курс третьего семестра ВМК посвящен операционным системам, а на семинарах в его поддержку рассматривается программирование на уровне системных вызовов; язык Си представляется вполне подходящим иллюстративным примером в этой ситуации. Однако подходит ли Си в качестве учебного пособия для *начального* обучения? Выше уже приведен ряд причин, почему на этот вопрос следует ответить отрицательно. Добавим к ним еще одну: в Си попросту *отсутствуют* некоторые базовые концепции, которые явно заслуживают того, чтобы быть проиллюстрированными.

Начнем с того, что в Си присутствует один, и только один способ передачи параметров в подпрограммы — а именно, передача по значению. Передачу по ссылке приходится имитировать с помощью явной передачи адресов. Проблема тут не только в том, что это сложно для понимания неподготовленной аудиторией, но и в том, что у студента сложится впечатление (неправильное), что в языках программирования только и бывает передача по значению. Паскаль в этом плане подходит гораздо лучше, ведь в нем есть и `var`-параметры. Конечно, всего многообразия способов передачи параметров это не исчерпывает, ведь бывает еще и передача по имени, и ленивые вычисления, предполагающие передачу невычисленного выражения. Но наличие хотя бы двух, а не одного способа передачи параметров демонстрирует обучаемому очень важный факт: **существует больше одного способа передачи параметров в подпрограммы**. Зная два таких способа, человек, как правило, на уровне подсознания оказывается лучше подготовлен к восприятию третьего и последующих способов.

Возражение, что-де `var`-параметры Паскаля все равно реализуются именно через передачу адреса, тут несколько неуместны: речь ведь идет об изобразительных средствах языков программирования, а не о том, как эти средства реализуются. Программисты, использующие языки очень высокого уровня, та-

кие как Лисп, Пролог, Haskell и тому подобные, часто вовсе не задумываются о том, как же этот вычислитель реализован.

В качестве второго важного объекта, отсутствующего в Си, я назову обыкновенные массивы. Утверждение, что **в языке Си нет массивов**, часто вызывает удивление — но ведь при описании массива у нас не возникает никакого имени, которое представляло бы собою весь этот массив целиком, как единый объект; имя массива в действительности представляет собой константу адресного типа, указывающую на первый элемент массива, а вовсе не массив как таковой. Над массивами (как таковыми) в Си нет никаких операций — собственно говоря, потому, что такие операции было бы не над чем проводить. Пикантность ситуации еще и в том, что в Си присутствуют не только средства описания массива, но даже тип «указатель на массив»[2], однако самих массивов как не было, так и нет. Более того, операция индексирования в Си представляет собой не более чем арифметическую операцию над адресом и числом, и для ее существования сами массивы оказываются не обязательны.

Здесь можно возразить, что используемая в Си для работы с массивами адресная арифметика, вообще говоря, мощнее, чем массивы того же Паскаля; так, в Си можно создавать массивы динамически изменяемого размера, к тому же с любой частью любого массива можно работать как с массивом (например, если есть массив из 20 элементов, то можно рассматривать его элементы с 3-го по 12-й как массив из 10 элементов, причем для этого достаточно одной операции сложения). Все это так, но я ведь и не утверждаю, что Си хуже Паскаля в качестве профессионального инструмента, я и сам часто пишу на Си, тогда как на Паскале последний раз всерьез что-то делал лет пятнадцать назад. Речь в настоящем эссе идет не о том, какой из языков лучше использовать для написания программ, и даже не о том, надо или не надо изучать Си в рамках основных курсов (я сразу отметил, что для меня необходимость этого очевидна), а о том, пригоден ли Си для начального обучения.

Массив не просто как область памяти, в которой располагаются подряд N переменных одного типа, но как отдельный полноценный *тип переменной* — это концепция, несомненно достойная того, чтобы донести ее до студентов. Чтобы развеять сомнения в этом, я хотел бы подчеркнуть, что отсутствие массива как полноценного типа переменной нарушает в Си концептуальную целостность системы типов, ведь *указатель на массив* (именно на массив, а не на его элемент) в языке в качестве отдельной сущности присутствует. Скажу больше, объяснить, что за странная штука «указатель на массив», проще человеку, знакомому с Паскалем, нежели тем, кто начал сразу с Си, потому что у них оборот «`p` указывает на массив» прочно ассоциируется с указанием именно что на первый элемент массива; дело осложняется еще и тем, что адрес-то в обоих случаях используется один и тот же, иначе говоря, численное значение указателя на массив (например, на строку двумерной матрицы) такое же, как и указателя на ее же первый элемент. Таким образом, отсутствие в Си массивов как полноценного типа переменных осложняет объяснение даже самого языка Си, не говоря о других языках; у тех учащихся, для кого странные массивы Си стали первы-

[2]Выражения такого типа возникают при описании многомерных массивов.

ми массивами в жизни, позже часто наблюдаются трудности с пониманием, что представляет собой объект-вектор, который можно ввести в Си++.

Мне как-то раз довелось общаться с профессиональным программистом (то есть человеком, работающим по этой специальности и получающим за это деньги), который всерьез утверждал, что понятия «массив» и «адрес массива» тождественны. Его не могли убедить в обратном никакие доводы, включая и тривиальное соображение о том, что адрес массива занимает 4 байта, тогда как сам массив — в общем случае гораздо больше. Понятие «указатель на массив» он, как и следовало ожидать, не понимал, а на вопрос, какой же тип, по его мнению, будет иметь имя, описанное как двумерный массив, он ответить не смог, но в своей уверенности так и не поколебался.

Продолжая тему типизации, отмечу, что отсутствие строгой типизации в Си тоже не слишком способствует пониманию общезначимых программистских концепций. Все-таки символ и его ASCII-код — *это не одно и то же*, хотя бы потому, что операция сложения двух однобайтовых чисел вполне осмысленна, тогда как операция сложения двух букв (если, конечно, это не конкатенация в строковом смысле, но в Си такого нет) — бессмысленна. Да и перечислимый тип изначально все-таки не то же самое, что набор целочисленных констант, ну а отсутствие отдельного булевского типа оказалось, как мы знаем, настолько неудобно, что в Си++ тип `bool` ввели, хотя и не сразу. По правде говоря, Паскаль тоже не умеет запрещать складывать килограммы с километрами, но имеющиеся в нем зачатки строгой типизации подготавливают слушателя к встрече с действительно строгими типами в Аде и, например, доменами в базах данных.

Наконец, в Си нет полноценной модульности, а то, что есть вместо нее, требует использования нетривиальной техники, основанной на директивах условной компиляции[3]. Здесь можно возразить, что в «стандартном Паскале» модульности тоже нет, но ведь в реальной жизни практически все реализации Паскаля поддерживают вполне полноценные модули. Отсутствие знакомства с правильной системой модульного программирования заставляет студентов поверить, что техника раздельной компиляции исходников на языке Си, включающая заголовочные файлы и буквально *состоящая* из хаков — это и есть модульность, но ведь на самом деле это не так.

7 Заключение

Итак, заменять Паскаль на Си в первом семестре, с моей точки зрения, недопустимо; позволю себе в заключение высказать ряд соображений относительно того, какие изменения в программистских предметах на первом курсе не только допустимы, но и желательны.

Во-первых, следует, как мне кажется, отказаться от давно и прочно мертвой платформы MSDOS и Турбо-Паскаля, который уже больше десятка лет не поддерживается производителем. Отмечу, что переход на более современные средства может изрядно упростить изложение иллюстративного материала, и в особенности это касается курса «Архитектура ЭВМ и язык ассемблера». Пере-

[3]Имеются в виду защитные макросы заголовочных файлов.

ход от мертвой системы команд 16-битного 8086 к 32-битным программам для i386 в «плоской» модели памяти позволяет сэкономить массу учебного времени, которое сейчас тратится на объяснение нелепых странностей старой 16-битной системы команд. Следует, в то же время, понимать, что писать на языке ассемблера под Windows — чрезмерно сложно и потребует глубокого проникновения в WinAPI, что, пожалуй, еще хуже, чем изучение бессмысленных странностей 8086; с другой стороны, программирование на языке ассемблера под операционные системы семейства Unix (Linux и FreeBSD) ни малейших сложностей не представляет.

Вообще, начинать обучение программированию с рисования окошек, естественно, недопустимо, так что при работе под Windows мы вынуждены заставлять студентов писать консольные приложения, которые в этой системе выглядят неполноценными, если угодно, ненастоящими. Конечно, мы и не ставим перед собой цели прямо на первом курсе сделать из студентов профессиональных програмистов, так что, кажется, «неполноценность» консольных приложений под Windows, а равно и «мертвость» MSDOS не должны быть препятствием учебному процессу. К сожалению, утверждая так, мы не учитываем, что студенты — живые люди и их личное отношение к предмету (на эмоциональном уровне) способно весьма и весьма существенно повлиять на усвоение материала и успеваемость. Использование устаревших или в том или ином смысле «ненастоящих» инструментов, нравится нам это или нет, расхолаживает студентов и формирует отношение к программистским предметам как к бесполезной трате времени.

В этой связи представляется перспективной идея перевода практикума первого курса в операционные среды Unix, которые основаны именно на культуре консольных приложений. В ОС Unix консольной является практически любая программа, включая и те, которые снабжены графическим пользовательским интерфейсом (наличие окошек не избавляет программу от наличия стандартных потоков ввода и вывода). Под ОС Unix доступны реализации практически всех живых языков программирования, включая, разумеется, и несколько конкурирующих реализаций Паскаля (прежде всего это Free Pascal и GNU Pascal, обе реализации имеют вполне профессиональное качество), и по меньшей мере два ассемблера, подходящие для учебных целей — NASM и FASM. Не могу не отметить, что в качестве управляющих систем суперкомпьютеров, как правило, тоже используются различные версии Unix. Компьютерные классы под управлением Unix работают существенно надежнее, и, более того, в настоящее время возможность загрузки Unix присутствует во **всех** компьютерных классах на факультете, в том числе и там, где на машинах инсталлирована ОС Windows (при работе с ОС Unix локальный жесткий диск не используется, операционная система загружается по сети с сервера). Использование дорогостоящей (прежде всего, в эксплуатации) системы Windows на первом курсе не оправдано еще и потому, что возможности самой системы, вообще говоря, на первом курсе не задействуются, ведь она используется просто как пускач для эмуляции MSDOS, а эмуляторы MSDOS в крайнем случае есть и под Unix.

Кроме того, если ставить целью ускоренную подготовку хотя бы некоторого количества студентов для работы с суперкомпьютерами, следует, видимо,

учесть, что далеко не все студенты вообще способны эту дисциплину освоить. В этой связи, возможно, было бы осмысленно наряду с тестированием по английскому языку перед распределением по группам проводить также тестирование по программированию; на основе такого тестирования можно было бы сформировать одну-две группы, в которых все студенты уже умели бы программировать, что позволило бы расширить их подготовку на первом курсе — например, прочитать им основные курсы в сокращенном варианте, предложить сдать по этим курсам досрочный экзамен, а затем специально для этих студентов прочитать один или два спецкурса, посвященных профессиональному программированию.

8 Благодарности

Автор считает своим приятным долгом выразить признательность А. Чернову и А. Сальникову за конструктивные дискуссии, а также Е. А. Бордаченковой за проявленную инициативу. Если бы не они, это эссе вряд ли когда-нибудь было бы написано.