
Проблемы преподавания языка C неподготовленной аудитории

А. Г. Фенстер, <http://info.fenster.name>

18 февраля 2010 г.

1 Из двух зол

Этот текст можно в некоторой мере считать ответом на недавно выложенное в сеть эссе А. В. Столярова «[Язык Си и начальное обучение программированию](#)», в котором автор аргументированно объясняет, что обсуждаемый язык, при всей необходимости его изучения для будущего программиста, не должен быть *первым* изученным языком.

Перечислю тезисно основные моменты этой статьи.

1. Простейший пример «Hello, World!» не так-то просто объяснить студенту, не имеющему опыта программирования: в нём встречаются понятия директивы препроцессора, функции (причём возвращающей значение), символ перевода строки (неочевидная концепция для людей, не видевших печатной машинки) и прочее.
2. Ввод данных при помощи `scanf`, требующий знания указателей и операции взятия адреса.
3. Язык способствует написанию кода с «хаками» и побочными эффектами, что (независимо от пользы или вреда в промышленном программировании) не делает обучение проще.
4. Отсутствуют способы передачи параметров в подпрограммы, отличные от передачи по значению.
5. Отсутствие массивов в привычном (читай «паскалевском») понимании этого слова.

6. Отсутствие строгой типизации.

7. Отсутствие модульности.

Что ж, с этим трудно (пожалуй, бессмысленно) спорить. Ожидать, что созданный почти сорок лет назад для промышленной разработки язык будет идеально подходить для обучения, было бы по меньшей мере наивно. Можно только завидовать преподавателям ВМиК МГУ, что у них есть время начинать с Паскаля, а затем давать курс С, С++, системных вызовов UNIX и т. п. К сожалению, такая роскошь доступна не всем, а если нужно «запихать» начальный курс программирования в один семестр (после которого, скажем, на мехмате НГУ программирования практически не будет, если не считать им недо-курс по С++ из девяти практических занятий и вычислительную практику) — выбора не остаётся. Мы хотим выпустить людей, которые в состоянии читать (а в идеале — писать) код хотя бы на одном широко используемом языке программирования, и, к сожалению, Паскаль таким языком не является. Таким образом, первоначальное обучение на С мне представляется необходимым злом (если угодно, меньшим из двух).

2 Первая программа

Впрочем, так ли серьёзны перечисленные выше проблемы?

Семинары, на которых изучается язык С, я в том или ином виде веду с 2003 года (сначала в Высшем колледже информатики при НГУ, затем в самом университете на мехмате и ФИТе), и за это время места, вызывающие основные трудности у студентов, определились достаточно чётко.

Проблема, действительно, в том, что обучение языку С — как хождение по минному полю. Шаг влево, шаг вправо — скатываемся в тему, для понимания которой нужно знать немного больше, чем обычно знает первокурсник. Соответственно, изложенный далее порядок тем и примеры к ним я продумывал достаточно долго, проверил не на одной группе и могу утверждать: в принципе, это работает.

2.1 Не начинайте с «Hello, World!»

Ни разу ещё не было у меня ситуации, когда в группе находился студент, не программировавший на императивном языке никогда. Пусть кто-то даже не помнит, что такое цикл `for` (неважно, в каком языке), но понятие *переменной* обычно в долгих объяснениях не нуждается. Поэтому неоднократно в объяснениях я ссылаюсь на то, как нечто сделано в Паскале. Если же кто-то не программировал вообще никогда — мне кажется, изложенные в правильном порядке сведения могут найти свой путь даже к самому неподготовленному студенту.

Итак, начинаем с того, что пишем на доске `int a`; и говорим, что `a` — имя, связанное с определённым местом в памяти, которое может хранить значения из некоторого набора; в данном примере это целые числа, обычно от -2^{31} до $2^{31} - 1$. Это обычно вопросов не вызывает. Заодно говорим, что есть типы `float`, `char` и многие другие, о которых мы поговорим позже.

Далее на доске появляется оператор присваивания. «Переменной мы можем присвоить значение: например, `a = 7`; или даже `a = a + 1`;». Я обычно произношу вслух примерно такую фразу: «В математике эта строка — уравнение, не имеющее решений, однако нужно понимать, что переменная в программировании и переменная в математике — вещи разные; в программировании переменная может неоднократно менять своё значение, например, здесь она увеличивается на единицу».

Дальше перечисляем основные операции. Нам их пока нужно немного: пять арифметических (+, -, *, /, %), три логических (&&, ||, !) и операции сравнения (>, <, ==, >=, <=, !=). Можно сказать про то, что при делении целых получается целое, а если один из аргументов — вещественное число, то получится вещественное, но это бесполезно: с первого раза это вряд ли кто-то запомнит. Ничего страшного. Конечно, нельзя не обратить внимание на отличие `==` от `=`, обычно я рисую табличку:

	Присваивание	Проверка на равенство
Pascal	<code>:=</code>	<code>=</code>
C	<code>=</code>	<code>==</code>

Естественно, бесполезно надеяться, что никто ни разу не напишет `if (a = 0)`, как и бесполезно сразу приучать студентов к `if (0 == a)` (я и сам нечасто так пишу, полагая, что не следует приносить читаемость кода в жертву и что если программист — дурак, эти трюки его не спасут).

Отличие `=` и `==` — пожалуй, единственное место в C, о котором каждый студент набьёт свой десяток шишек — но, в общем-то, проблем оно вызывает не больше, чем необходимость не ставить точку с запятой перед `else` в Паскале. Мне кажется, вполне сравнимые неудобства.

Условная инструкция обычно всем знакома. Достаточно сказать, что условие вычисляется как число, а специальный тип для логических выражений отсутствует.

Самое время перейти к функциям. Проблем с пониманием следующей реализации функции, вычисляющей модуль числа, обычно ни у кого не возникает:

```
int abs(int x)
{
    if (x < 0)
        return (-x);
    else
        return x;
}
```

Здесь и про тип результата (не «возвращаемого значения», это действительно абракадабра) поговорить можно, и про `return`, и про то, что `else` смело убирается, потому что `return` завершает вычисление функции.

Вроде бы, пока всё более-менее логично и нигде не сказано фразы «позже поймёте», которой мы так усиленно избегаем.

2.2 Зловещий printf

Я считаю, что нет ничего страшного в том, что студенты начнут пользоваться `printf`, не вполне осознавая, что представляет собой строка в двойных кавычках. В конце концов, и в Паскале дети могут и не знать, что `string` — на самом деле массив `char`'ов с длиной, хранящейся в нулевом элементе. Поэтому рассказываем про `%d` и `%f` и приводим несколько примеров. Простая фраза «`\n` переводит при печати курсор на новую строку», вопреки опасениям, проблем обычно не вызывает.

Да, возможно, объяснение (цитирую обсуждаемое эссе) «под завязку наполнено фактическими ошибками и недоговорками» (и печатается оно не на экран, а в поток стандартного вывода, и `\n` ведёт себя по-разному в разных ОС, и мало ли что ещё), но я не видел студентов, которые после

попытки объяснения функции `printf` в простом виде откидывались бы на спинки парт с видом «я ничего не понимаю».

2.3 А вот теперь `#include`

Теперь мы почти готовы к написанию программы. Сказать, что вместо «главного `begin/end`» в C пишется функция с именем `main`, возвращающая целое число, несложно. Я обычно просто говорю, что принято, чтобы программа, завершившаяся нормальным образом, возвращала 0 и что это число доступно, например, другой программе, которая запустила вашу. Желаящим потом в терминальном классе можно показать и `$?`, и условную инструкцию `if` в `bash`, но большинство вполне удовлетворится минимальным объяснением.

Сложнее с `#include <stdio.h>`. Работая с `gcc`, можно в принципе ничего такого и не писать — ни ошибок, ни предупреждений не будет (если не использовать `-Wall`), но это как-то не совсем честно. А вот сказать, что эта строка вставляет в вашу программу заголовок функции `printf` и других функций, чтобы компилятор о них знал, вполне честно и, мне кажется, достаточно понятно. На ФИТе, где студенты обычно более заинтересованы, попозже можно (и нужно) рассказать подробно о заголовочных файлах, `include guards` и прочем (они все будут писать проект из нескольких файлов и им это пригодится), мехмату же это ни к чему: кому интересно — спросит или прочитает сам.

Кстати, обратите внимание на расшифровку `stdio`. Если не проговорить вслух «standard input/output», количество людей, пишущих в первый раз по ошибке «`studio.h`», будет больше.

Итак, мы получили работающую программу, которую не стыдно привести здесь полностью:

```
#include <stdio.h>

int abs(int x)
{
    if (x < 0)
        return (-x);
    return x;
}
```

```
int main()
{
    int x;
    printf("Печать целых чисел: %d, %d\n", abs(5), abs(-7));
    x = abs(11);
    printf("Значение переменной x: %d\n", x);

    return 0;
}
```

После этого можно уже и квадратное уравнение решить у доски. Главное — не надо пока вводить данные из стандартного ввода. Задайте коэффициенты прямо в коде: `float a = 1.0, b = -3.0, c = 4.0;` — это не самое главное и, поверьте, не нужно рассказывать про `scanf` и оператор взятия адреса прямо на первом же занятии.

Объяснение прочих вещей, достойных упоминания на первом семинаре (а это все циклы, операторы инкремента-декремента и, возможно, что-то ещё) я здесь опущу. Я уже несколько лет заканчиваю первый семинар функцией `is_prime`, проверяющей число на простоту и возвращающей (не печатающей) 0 или 1, и печатью таблицы простых чисел с использованием этой функции. Можно поговорить про рекурсию, написать пример хвостовой рекурсии и даже поговорить про стек вызовов — по ситуации, в зависимости от уровня подготовки группы. А вот на втором семинаре придётся начать говорить про следующую серьёзную тему.

3 Указатели

Трудно не согласиться с Joel'ом, который утверждает, что указатели и рекурсия — две наиболее тяжело идущих темы у студентов. Единственный, как мне кажется, способ объяснить указатели так, чтобы люди без большого опыта программирования это поняли — рисовать, рисовать и рисовать квадратики, состояние памяти, выдумывать «номера ячеек памяти», которые вскоре заменятся на более наглядные стрелки:

```
int a = 7;    int *p = &a;
```



Далее, когда операции взятия адреса и разыменования станут более или менее понятными, говорим про передачу параметров в функцию. Здесь опять придётся порисовать, чтобы объяснить, что происходит, если функция `f`:

```
void f(int a)
{
    a++;
}
```

вызывается с константой (`f(7)`) или с переменной (`f(x)`) в качестве параметра. Обычно этот разговор я заканчиваю функцией `void swap(int *p, int *q)`, меняющей местами значения переменных, адреса которых были переданы в функцию. Обычно я так же рисую картинку со стрелочками, чтобы показать, что значения фактических параметров функции не изменились, что не помешало функции изменить значения клеток памяти, адреса которых в этих параметрах записаны. Не было ещё группы, которая на вопрос «И как же теперь функцию `swap` правильно вызвать?» не нашла бы правильного ответа `swap(&a, &b)`.

Далее по желанию можно рассказать про массивы, связь массивов с указателями, передачу массивов в функцию, а можно оставить это на следующее занятие, а оставшуюся часть второго семинара посвятить следующей большой теме — вводу данных в программу.

4 Ввод

Объяснить работу функции `scanf` — занятие действительно не из простых, и мне кажется, что лучше всего постараться сделать так, чтобы студенты сами догадались, как такая функция должна быть устроена (речь сейчас, естественно, о передаче адресов в функцию, а не о разборе форматной строки или `stdarg.h` — об этом следует поговорить, но только в конце семестра или даже в следующем).

Начнём с функций `getchar` и `putchar` — самых простых функций ввода/вывода, возвращающих/принимающих одно значение типа... вовсе не `char`, а `int`, потому что ещё бывает EOF, и на объяснение этого обычно уходит минимум минут пять. Зато после объяснения можно разобрать первый пример рекурсивной функции, не сводящейся к циклу (если не использовать дополнительной памяти): функцию разворота введённой

строки (также здесь возникают функции с типом результата `void`, т. е. процедуры в терминах Паскаля).

Когда с функцией `getchar` студенты освоятся, разбираем, как узнать числовое значение цифры (`c - '0'`), и мы готовы к тому, чтобы написать функцию `int read_integer(void)`. А теперь предлагаем задуматься, как будет выглядеть эта функция, если мы хотим, чтобы она не возвращала значение, а изменяла переменную, данную ей в качестве параметра (для паскалистов можно сказать «как `read` в Паскале»). Функция `void read_integer1(int *p)`, а вскоре и её вызов `read_integer1(&a)` появятся на доске без больших проблем, а перейти от этого к `scanf` — вообще говоря, дело техники.

5 Дальнейшие действия

Честное слово: после того, как объяснены указатели и `scanf`, изучение С мало чем будет отличаться от изучения любого другого императивного языка того времени. Конечно, есть специфика: один семинар нужно посвятить строкам (ничего лучше реализации на доске функций `strlen`, `strcpy` и прочих по этому поводу я не придумал), функции работы с файлами тоже лучше разобрать подробно, но ничего нереального в этом нет. Да, побочные эффекты и `while ((c = getchar()) != EOF)`. И что с того? Студент, способный на математике понять, что такое $f(g(x))$, не должен чувствовать проблем с разбором подобных конструкций.

Само собой, дальше пойдёт динамическая память (разговор о которой начинается с вопроса «как сделать функцию, результат которой — массив или строка»), списки и прочее, а параллельно — поиск, сортировка, деревья, графы, то есть весь джентльменский набор, который даётся совершенно независимо от используемого языка программирования.

В этом разборе я ещё не затронул важной темы, о которой говорится в последних четырёх пунктах приведённого в самом начале «списка недостатков»: отсутствие некоторых вещей, важных для современных языков программирования, таких, как различные способы передачи параметров, «настоящие» массивы, модульность, строгая типизация. Но мне не кажется, что за семестр (или даже за год) можно сформировать у начинающих понимание важности модульного подхода, объяснить, зачем нужна передача параметров по имени или строгая типизация. Для желающих существуют спецкурсы (те же «Парадигмы программирова-

ния»), где времени достаточно для разговора о том, какие вообще бывают языки. К начальному курсу программирования все эти вопросы имеют весьма малое отношение.

Основной идеей текста, ссылка на который дана в начале, была простая мысль: нельзя учить С первым языком. Утверждаю, что учить С первым языком можно. Мой опыт показывает, что не стоит делать этого в школе (разве что на факультативе для действительно интересующихся), потому что средний школьник и с Паскалем общается весьма на «вы». А первый курс — самое то, и не нужно бояться объяснять студентам сложные вещи. В конце концов, матанализ определённно сложнее.

6 Постскрипtum

Полностью согласен с идеей перевода обучения на UNIX-подобные системы, выдвинутой в конце статьи А. В. Столярова. Не дожидаясь команды «сверху» (будет ли она вообще когда-нибудь дана?), я пару лет тому назад своими силами перевёл студентов своих групп на Linux и никаких действительно серьёзных минусов этого до сих пор не заметил (зато плюсы, в общем, очевидны). Благодаря [Евгению Балдину](#) об этом переходе вышла [статья в Linux Format](#), в которой описывается, что было необходимо сделать для отказа от Windows / MSVS в учебном процессе. В общем-то, сделать нужно было не так уж и много.

Любые комментарии по поводу этого текста принимаются по электронной почте `fenster@fenster.name`.