языки, которые мы потеряли

крис касперски, a.к.a мыщъх, a.k.a nezumi, a.k.a elraton, a.k.a souriz, no-email

> **позади нас осталось целое кладбище языков. не прижившихся парадигм, вымерших концепций. идей, опередивших свое время. для будущего не осталось ничего. все, что только было можно придумать — уже придумано, реализовано, опробовано и… выброшено на свалку за ненадобностью. эпоха великих открытый давно прошла и нам осталось лишь обезьянничать, двигаясь от золотого века в деградирующую тьму чудовищных лингвистических решений. назад возврата нет. а все потому, что…**

введение

…язык определяет мышление, а мышление формирует язык, в результате чего язык начинает стремительно развиваться, приобретая новые, ранее не свойственные ему черты, не только позволяющие программисту выражать инженерную мысль во всей ее полноте, но и порождающие совершенно новые мысли, вытекающие из свойств языка.

Появление абстрактных типов переменных (не имеющих прямого машинного воплощения) — довольно наглядный тому пример. Программисты каменной эры, высекающие программы долотом (долотом в прямом смысле — на перфокартах) мыслили в конкретной плоскости — включить мотор дисковода, прочитать сектор, положить данные в ячейку размеров в N-слов, расположенную в памяти по адресу P… Современные языки отошли от железа и мыслят категориями абстрактных концепций, позволяя сравнивать переменные A и B, одна из которых представляет файл на диске, а другая — сетевой локатор.

Хорошо это или плохо?! Все намного хуже, чем вы предполагаете. Абстрактное мышление смертоносно, ибо провоцирует программиста на решение задачи в общем виде на сто тысяч строк исходного кода, в то время как частное решение (отвечающее ТЗ) уложилось бы и в десяток. Но это еще не самое страшное, истина в том, что…

явление приплюсного си народу

…язык Си++ (а вместе с ним и его многочисленные "преемники") уже давно не является объектно-ориентированным языком и доэволюционировал до метапрограммирования, более известного как программирование с использованием шаблонов (хотя шаблоны всего лишь одно из средств его реализации). Конечная цель метопрограммирования — создание программ, создающих другие программы как результат своей работы, то есть, грубо говоря, язык престает быть сущностью, полностью подчиненной авторскому замыслу, и приобретает определенную самостоятельность, существенно упрощающую решение поставленной задачи, но вместе с тем — вносящую большую долю неопределенности и неуправляемости.

Например, вместо того, чтобы писать десяток функций, сравнивающих переменные различных типов, достаточно запрограммировать один-единственный шаблон. Правда, тут же возникает вопрос — а как он себе поведет, если ему "скормить" нечисловые переменные? Ответ — шаблон вообще ничего не знает ни о типах, ни о размерностях. Он просто обращается к методам соответствующих классов, которые могут быть реализованы как угодно, либо же не реализованы вовсе и тогда программа даст сбой. До тех пор, пока использовались только статические абстрактные типы, "перекладываемые" на машинное представление еще на стадии компиляции, транслятор легко отлавливал подобные ошибки, но с появлением динамических типов, обрабатываемых в реальном времени, язык стал вообще неконтролируемым и программы начали падать в случайное время.

И ведь никто не виноват!!! Программист, реализующий шаблон, тут, естественно, не причем, ведь он написал, что-то вроде: IF (a>b) THEN RETURN A; ELSE RETURN B, переложив реализацию процедуры сравнения на разработчиков классов, которым, возможно и в голову не могло прийти, что их классы кому-то понадобиться сравнивать. Взять хотя бы класс shape, реализующий различные геометрические фигуры: отрезки, круги, треугольники… Можно ли сравнивать два экземпляра класса share с друг другом? А почему бы и нет! Это может соответствовать длинам отрезков и площадям фигур. Но ведь… может и не соответствовать!!!

Проблема объектного- и мета- программирования в том, что абстрагируясь от "ненужных" технических подробностей, они скрывают информацию о своем поведении от программиста. Программист каменной эры, глядя на запись A = A + B, мог _однозначно_

сказать, что она выдаст при любых значениях A и B (даже с учетом переполнения переменных). А сейчас? Возьмем две переменные типа "символ" и попытаемся их сложить. Вопрос: как поведет себя реализующий их класс? Будет ли он добавлять код одного символа к другому, оперируя с ними как с целочисленными переменными (поведение по умолчанию) или скомбинирует их в двухсимвольную строку? А если взять экземпляры класса share, то это вообще конец определенности. Математически можно складывать только отрезки (да и то с той лишь оговоркой, что задано их направление, т. е. мы оперируем с отрезкой как с вектором), но сложение квадрата с треугольником — бессмысленно. Зато в графическом аспекте сложение может быть уподоблено наложению, что, кстати говоря, тут же нарушит коммутативное свойство, т. е. A+B совсем не тоже самое, что B+A! Вот и попробуй после этого вносить в программу изменения…

У программистов каменной эпохи подобных проблем просто не возникало, поскольку язык не давал возможности оперировать абстрактными концепциями и в каждой точке программы приходилось выражать законченную техническую мысль. Конечно, это приводило к дублированию кода, снижало наглядность листинга, но зато исключало неоднозначности его интерпретации.

Абстрагируясь от базовых концепций, мы усиливаем лексическую мощь языка (там где раньше приходилось писать тысячу команд, сейчас достаточно одной), но вместе с нею увеличиваем количество взаимодействий между различными компонентами, которые как-то нужно учитывать… В результате за кажущуюся легкость программирования приходится расплачиваться многократно возросшей сложностью проектирования.

Язык невозможно осваивать последовательно, шаг за шагом, как это было раньше. Если на Бейсике первая программа состояла всего из одной строки "PRINT 'hello'", то теперь даже минимально работающая программа (с шаблонами, ес-но) этих строк насчитывает десятки! Создавая новый экземпляр класса, мы должны обработать ситуацию с нехваткой памяти, установить обработчики исключений (ну или хотя бы "заглушки") и заблаговременно предусмотреть реакцию программы на ситуации, которые в рамках древнего процедурного программирования просто не возникают.

Кстати, тот, кто считает, мета-программирование достижением последних десятилетий — жестоко ошибается. Да, в языке Си++ оно появилось совсем недавно и в полном объеме (описанном в последних редакциях Стандарта) не реализовано ни в одном реально существующем компиляторе, а Nemerle и R# (языки программирования для платформы .Net со встроенной поддержкой метапрограммирования) — вообще младенцы, но… на самом деле концепция метапрограммирования возникла еще во времена палеолита.

Lisp, появившийся в далеком 1958 году, — хороший пример языка, естественным образом поддерживающий метапрограммирование, одной из задачей которого является создание программы, выводящей точную копию своего собственного исходного текста — так называемый куин (англ. quine). На Lisp'е он записывается так:

```
(funcall (lambda (x)
               (append x (list (list 'quote x))))
             '(funcall (lambda (x)
                         (append x (list (list 'quote x))))))
```

Листинг 1 программа на Lisp'е, распечатывающая сама себя

На Си так:

```
#include<stdio.h>
 char*i="\\#include<stdio.h>",n='\n',q='"',*p=
 "%s%cchar*i=%c%c%s%c,n='%cn',q='%c',*p=%c%c%s%c,*m=%c%c%s%c%c;%s%c",*m=
 "int main(){return!printf(p,i+1,n,q,*i,i,q,*i,q,n,q,p,q,n,q,m,q,n,m,n);}"
 ;int main(){return!printf(p,i+1,n,q,*i,i,q,*i,q,n,q,p,q,n,q,m,q,n,m,n);}
```

Листинг 2 программа на Си, распечатывающая сама себя

А теперь попробуйте реализовать тоже самое на Си++ с использованием шаблонов и посмотрите насколько сильно они вам "помогут". Парадигма метопрограммирования, красиво реализованная в Lisp'е, была заброшена на полвека именно из-за потери управляемости языком, но затем восстала из пепла в ужасной реикцианации, притянутой за уши, уходящие своими корнями в директивы препроцессора языка Си… Но это уже совсем другая история.

научился курить сам — научи товарища!

Язык Си++ оказал огромное влияние как на мышление программистов, так и на развитие всех последующих языков, став стандартом де-факто. Никто, естественно, не говорит, что ООП- и мета-программирование — это плохо. Почему обязательно плохо?! Очень даже хорошо! Местами… Но вот мысль о том, что ООП — единственно правильный подход — ужасна. Самолеты и космические корабли мирно сосуществуют с велосипедами и автомобилями. Никому ведь и в голову не придет летать за сигаретами на ракете, особенно, если сигареты продаются в киоске на соседнем углу.

Но ведь это же неправильно!!! Появление ракет _должно_ перевернуть наше мышление!!! И ведь еще как перевернет! Строим мега-киоск за орбитой Плутона и каждому даем по ракете, чтобы туда летать, а горючее покупаем за деньги, вырученные от строительства космодромов и продаж ракет. Кто не может строить ракеты — пусть учит других как на них летать. Вы только прикиньте сколько создается новых рабочих мест и главное, что все в бизнесе. Вот тут уж действительно, возврата в прошлое нет… Сигареты стоят миллиарды долларов и деньги в индустрии вращаются просто огромные. Кто же захочет от них отказываться?! Напротив, ракеты будут стремительно "совершенствоваться", чтобы за сигаретами можно было летать аж на Альфу-Центавра.

Говорите, что это нелогично и невозможно? Но ведь именно такая ситуация сложилась с Си++. Судите сами — реализация компиляторов языка Си++ очень сложная и дорогостоящая задача, а сам язык настолько обширен и объемен, что требует для изучения невероятных усилий. Чтобы окупить средства, вложенные в разработку компиляторов, фирмы вынуждены "подсаживать" на него миллионы программистов, которые в свою очередь, пройдя длительный (и ужасно мучительный) путь обучения Си++, просто не могут признаться себе в том, что напрасно убили десять лет своей жизни (данной человеку лишь однажды!) и что стоящие перед ними задачами с ничуть не меньшей эффективностью реализуются на чистом Си и других процедурных языках, легко осваиваемых на ходу без отрыва от жены, рыбалки и производства. Вот они и начинают убеждать остальных, что процедурные языки давно мертвы. Сложность ради сложности. Ничем не оправданная. Мрак.

Впрочем, среди этого мрака есть и светлые пятна. При приеме в нормальную фирму за попытку "выпендиться" и решить задачу с применением шаблонов, там где они не требуются, по головке не погладят и программу скорее всего не зачнут. Один из известных "подколов" — написать программу, выводящую сумму первых 100 натуральных чисел на экран. int a, b=0; for(a=1;a<101;a++) b+=a; printf("%d\n",b); – незачет, printf("5050\n") – зачет. Ключевое слово здесь — "выводящую", а не "вычисляющую". Чуть более жестокий вариант — вывод первой сотни простых чисел. Ясен пень, что сумму членов арифметической прогрессии можно вычислить и в уме, а найти простые числа без компьютера не то, чтобы нереально, просто… зачем мучаться, когда компьютер под рукой? Однако, программист — это прежде всего инженер, а инженер должен не только внимательно читать ТЗ, но и выбирать адекватные средства для решения задачи. В данном случае — написать программу, генерирующую программу, выводящую простые числа на экран, т.е., что-то вроде: printf("3, 5, 7...\n"), после чего первую программу можно смело стереть. Действительно, какой смысл обсчитывать одни и те же данные при каждом запуске программы, когда это достаточно сделать всего один раз?!

Важно понять, что язык — это всего лишь средство выражения мыслей. Не стоит фетишизировать его. Умные мысли можно выразить и в машинном коде (там, где это оправдано), если же мыслей нет — не поможет никакой язык. Проблема, однако, в том, что программирование машины (т. е. задание последовательности операций, которая она должна выполнять) все больше превращается в диалог в ней. Программисты перестали обдумывать решения поставленных задач вдали от машины. Компьютер превратился в калькулятор, а программы — в "записки охотника". Именно потому, когда современный программист слышит "сумма ряда", он рефлекторно представляет себе цикл, и машинально тянется к клавиатуре… Он утратил понимание того, что язык (машинный) тут и рядом не валялся. Это чисто математическая задача, а от машины требуются лишь средства ввода/вывода для печати результата.

вавилонское столпотворение

Всякий раз, когда появляется очередной новый язык, о котором говорят, как об "окончательном" и "безальтернативном", предрекая скорую смерть всех остальных, мне становится смешно.

Сам по себе язык в отрыве от среды программирования — малоинтересен, да и все языки крутятся вокруг одного и того же набора концепций и парадигм, просто разные языки смешивают этот коктейль в различных комбинациях. Популярность Си++ отчасти вызвана тем, что он вобрал в себя _все_ существующие парадигмы, допускающие эффективную реализацию. Остальные языки либо неэффективны, либо представляют собой подмножество Си++.

Да, есть совершенно иные классы языков, например, логические языки (ярким примером которых является Prolog), основанные на выводе новых фактов из заложенных в них данных согласно установленной системе логических правил, работающей в замкнутом мире фактов. Грубо говоря, если определить объекты "вода", "чайник", "газ", "огонь", то Prolog'у достаточно дать задание "вскипятить воду", чтобы он поставил чайник на огонь. Но… насколько же медленно он это сделает!!! И сколько времени уйдет на формирование "базы знаний" об объектах окружающего мира.

В отличии от Си++, придерживающихся парадигм декларативного программирования, описывающих процесс вычисления в виде инструкций, изменяющих состояние программы, функциональные языки (самым популярным из которых является Haskell), не представляют собой последовательность инструкций и не имеют глобального состояния, вместо этого они определяют _что_ нужно вычислять, а _как_ это делать – забота транслятора. Вот только эффективных трансляторов декларативных языков упорно не наблюдается. Увы, машина не может мыслить, а ответ на вопрос _как_ это сделать предполагает активную творческую деятельность.

Таким образом, благополучию Си++ ничего не угрожает. Сколько бы языков ни создавалось и какие бы усилия ни предпринимались бы для продвижения их на рынок, мы получим либо урезанный вариант Си++, либо страшный тормоз. Ни то, ни другое программистам даром не нужно. Разумеется, это не означает, что Си++ венец эволюции. Скорее это свалка всех идей, демонстрирующих хорошую эрудицию его создателей, но нарушающих целостность языка, благодаря чему, собственно, с завидной регулярностью появляются "претенденты на престол", пытающие отобрать лучшие из конструкций Си++, скомпоновав их в гармоничный "букет", но… новые языки приходят и уходят, а Си++ вбирает в себя все удачные решения своих конкурентов, становясь целым миром. И наряд ли найдется хоть один вменяемый программист, который осмелится утверждать, что он владеет этим миром во всей его полноте. Максимум на что он может претендовать — на ту малую часть, которую вмещает ум отдельно взятого индивидуума.

заключение или есть ли свет в конце тоннеля?

Первые языки программирования были крайне простыми, учились быстро и легко, а потому при обучении основное внимание уделялось концептуальным понятиям — архитектуре, алгоритмам и т. д., однако, языковые средства того времени были недостаточно выразительными для записи мыслей в наглядной удобочитаемой форме и код быстро превращался в "спагетти", которое было невозможно ни отлаживать, ни сопровождать, ни развивать. При достижении определенных размеров программы буквально "рушились" под собственной тяжестью и проще было переписать их заново, чем добавить пару строк в уже существующий код, что, естественно, не устраивало ни пользователей, ни программистов.

Языки последующих поколений совершили качественный рывок вперед, избавившись от множества недостатков своих предшественников, но… при этом они так усложнились, что язык из инструмента для решения проблем сам по себе превратился в проблему, образовав предметную область шириной во всю жизнь. Алгоритмы оттеснились на задний план и ВУЗы стали выпускать молодых людей с кашей в голове и программирующих на Си++ еще хуже, чем на Бейсике — с кучей глобальных переменных и десятками классов, там где и трех функций хватило бы с головой… Программирование усложнилось настолько, что стало уделом избранных. Появились консультанты по языку (не умеющих программировать вообще, но знающих Стандарт как отче наш, это что-то вроде искусствоведов, не нарисовавших ни одной картины, но с умным видом рассуждающих о правилах композиции, восходящих и нисходящих мажорных и минорных линиях и т. д.)

При этом ни Си++, ни его последователи не реши поставленных перед ними проблем. Напротив, они открыли множество новых возможностей испортить дизайн программы так, что потом его никакими средствами уже не исправить. Если программу, написанную на процедурном языке можно переписывать по частям, исправляя ее структуру путем декомпозиции, то с Си++ этот номер так просто не пройдет. Иерархия классов жестко задается на этапе проектирования и закладывается в программу точно железобетонный каркас,

расширяемый только в одном направлении. В направлении наслаивания нового кода. В результате чего нас окружают программы-монстры, а программисты утрачивают возможность понимать друг друга. Изобилие языковых средств приводит к тому, что использование _всех_ конструкций языка одновременно становится затруднительно и неоправданно. Даешь каждому программисту по парадигме! Ну и что с того, что остальные ни строчки не понимают! Никто же ведь и не обещал, что программировать — легко!

А почему, собственно, программировать должно быть тяжело?! Почему мы ссылаемся на авторитеты всякий раз, когда чувствуем себя недостаточно компетентными?! Откуда вообще взялась слепая вера в то, что профессиональный программист _обязан_ идти в ногу с прогрессом, осваивая новые библиотеки, языки, etc?! И уже совсем не понятно, почему программирование превращается в соревнование кто напишет самую непонятную программу, с использованием новейших языковых средств.

Программирование идет по пути непрерывного наращивания сложности и эта гонка "вооружений" ничего хорошего в себе не несет. В мире есть только одна причина, способная поддерживать движение этой машины — деньги. Программное обеспечение невероятно дорого и разработчикам хочется, чтобы оно было еще дороже. Программировать быстро, красиво и эффективно становится просто не выгодно, вот индустрия и движется к собственному краху огромными прыжками.

Остановить этот суицид очень просто — достаточно на законодательном уровне потребовать от производителей отвечать за свой продукт, выплачивая неустойку за каждую критическую ошибку. Программисты мгновенно одумаются и выкинут из программы все лишние узлы, а оставшиеся перепишут с использованием "легких" языковых средств, выбросив "заумные" и "тяжелые" в топку.

Впрочем, процесс упрощения языков пошел уже и без того. При всей моей нелюбви к C#, я все-таки вынужден признать, что это большой шаг вперед, поскольку из него выкинули кучу Си++ конструкций, которые были слишком сложны для рядовых программистов, в результате чего в C# намного меньше шансов написать уродливую программу и это вселяет надежду, что свет в конце тоннеля все-таки есть. Или это всего лишь встречный? Вопрос риторический, а, значит, безответный.

>>> врезка полезные ссылки по теме

- **Why Pascal is Not My Favorite Programming Language**:
  - аналитическая статья Брайна Керигана — одного из создателей языка Си, — критикующего недостатки Паскаля, устраненные в Си, но заново возрожденные в Си++ и его потомках (на английском языке): http://www.lysator.liu.se/c/bwk-on-pascal.html;
- **433 Examples in 132 (or 162*) programming languages:**
  - исходные коды 433 простых программ ("hello, world", факториал, поиск максимума, пузырьковая сортировка, etc) на 162 различных языках программирования, как существующих, так и давно умерших — очень полезная штука для расширения кругозора (на английском языке): http://www.ntecs.de/old-hp/uu9r/lang/html/lang.en.html;
- **List of hello world programs:**
  - исходные коды "hello, world" программ на 190 языках программирования: http://en.wikibooks.org/wiki/Transwiki:List_of_hello_world_programs;
- эволюция программистов и языков программирования (шутка):
  - http://www.ariel.com.au/jokes/The_Evolution_of_a_Programmer.html;