языки программирования будущего — анализ или пророчество

крис касперски ака мыщъх

каждое утро сэр генри просыпается, разряжает оба ствола своего дробовика в потолок и кричит: "Я не знаю, чего я хочу, но я хочу этого немедля!"

к/ф. Sir Henry at Rawlinson's End

каким будут языки программирования через несколько лет? с пятидесятилетним опытом за плечами мы можем дать вполне объективный прогноз. языки эволюционируют намного медленнее программного обеспечения, создаваемого с их помощью и на радикального изменение ситуации рассчитывать не приходится, если только не…

введение

Попытка создания принципиально нового языка каждый раз наталкивается на что-то до боли знакомое старое. Идеи закончились уже в семидесятых и все последующие языки были лишь комбинациями предыдущих. На древних машинах никакого "программирования" вообще не существовало — грохочущий агрегат проектировался под конкретную задачу и ничего другого попросту не умел. ЭВМ следующего поколения управлялись при помощи разъемов и перемычек, прямым наследником которых являются современные ПЛИС'ки (Программируемая Логическая Интегральная Схема). Программное обеспечение еще не было отделено от аппаратного и программисту приходилось совмещать в себе навыки радиоинженера с мастерством математика. Изобретение перфокарточного ввода, а вместе с ним и машинного кода произвело настоящую революцию. Теперь уже можно было говорить о *нотации* — способе выражения программисткой мысли в некоторой транспортабельной форме, которому можно было читать, размножать, передавать по наследству и т. д. В общем, возникала письменность.

Правда, работать с машинным кодом было ужасно неудобно. И вовсе не потому, что восьмеричные коды команд с трудом удерживались в голове (настоящие программисты помнили их все). Машинный код всячески препятствовал модификации программы. Команды загрузки данных, условные и безусловные переходы требовали "жесткого" (hard coded) адреса, "связывающего" программиста по рукам и ногам. Если требовалось вставить несколько инструкцией в середину программы, приходилось корректировать ссылки на все последующие адреса или цеплять так называемую "соплю" — дописывать несколько команд в конец программы, делать на них jump, а затем возврат на предыдущее место выполнение. Многие программы на 90% состояли из "соплей" и разобраться с ними было очень непросто. Появление ассемблера решило эту проблему, хотя и было встречало в штыки старым поколением программистов.

забытые идеи или динозавры в исторической ретроспективе

Ассемблер существенно упрощал программирование за счет меньшей свободы и эффективности исполнения программ. В частности самомодифицирующейся код на ассемблере невозможен, то есть возможен конечно, но отнюдь не языковыми средствами. Самомодификация на ассемблере — грязный хак, а в машинном коде это было нормой. Ассемблер в некоторой степени абстрагирует от конкретной архитектуры, однако, программы по-прежнему страдали непереносимостью, а их разработка требовала большого самообладания. Вот так, например, выглядела программа "hello, word!" для PDP-10:

```
;; Hello world in MIDAS

title   hello

start:  .open   [.uao,,'tty ? 0 ? 0]
         .lose  %lsfil
```

```
        move    1, [440700,,[asciz "Hello, world"]]
loop:   ildb    2,1
        skipn   2
         .logou 1,
        .iot    2
        jrst    loop
end     start
```

Листинг 1 программа, "hello, world", написанная на ассемблере для PDP-10

Пока программы были невелики по размеру с этим приходилось мириться, но в какой-то момент программировать на ассемблере стало просто невыносимо и тогда появился первый язык высокого уровня — Фортран, научивший компьютер человеческой форме "разговора". Большинство рутинных операций и технических деталей язык брал на себя, позволяя программисту сосредоточится непосредственно на самой решаемой задаче. Программа "hello, word", написанная на Фортране, выглядела так:

```
*
C Hello World in Fortran 77
C (lines must be 6 characters indented)
*
      PROGRAM HELLOW
      WRITE(UNIT=*, FMT=*) 'Hello World'
      END
```

Листинг 2 программа "hello, world", написанная на языке Фортран

Вслед за Фортраном появилась целая серия подражателей — Алгол, PL/I, Си… им даже название придумали — ***императивные***. Программа, записанная на императивном языке, в грубом приближении представляет последовательность инструкций, передаваемую компьютеру на выполнение. К императивным относятся практически все современные языки. Остальным завоевать популярность так и не удалось. Но они были!

Вот, например, язык ***логического*** программирования Пролог. Это неимперативный язык, в котором программа представляет собой совокупность утверждений, таких например, как: sin(x) = 0.69; print(x). Вот так, например, на Прологе вычисляется факториал.

```
fak(0,1).
fak(N,R) :-  N > 0,  M is N - 1,  fak(M,R1),  R is N * R1.

?- fak(6,N).
```

Листинг 3 вычисление факториала на Прологе

Сморите, сначала программист говорит: fak(M, R1), затем определяет R1 как "R1 это N *R1". Все! Логическое соотношение между R и R1 задано. Обо всем остальном позаботиться транслятор. Ну разве не красота?

Функциональные языки. О! Это совсем другой зверь. Вместо последовательности инструкций типа: "отрой кран, наполни чайник водой…" они говорят, что есть такая штука как "кран" и есть такая штука как "чайник". Что произойдет при встрече чайника с краном — решает сам транслятор. Программист лишь описывает "характера" чайника с краном. Функциональные языки либо совсем не содержат переменных (а, значит, и оператора присвоения), либо не очень-то поощряют его использование. Тем не менее, программировать на них вполне возможно:

```
sub fac 1,
  if == 1 @[0]
    1,
  ( * @[0] &fac -- @[0] ),
```

Листинг 4 вычисление факториала на языке LIRL

Отсутствие переменных предполагает отсутствие типов данных. LISP — язык ориентированный на работу со списками — это бестиповой язык, в каноническом варианте которого, нет ничего кроме списков. Даже чисел нет. Число n представляется списков из n элементов. Звучит, пугающее, не правда ли? На самом деле, от чисел и переменных полностью никто не отказывается (в противном случае, это чрезвычайно снизило эффективность) и типичная LIST-программа выглядит так:

```
(defun fact (n)
```

```
  (loop for i from 2 to n
        for fact = 1 then (* fact i)
        finally (return fact)))
(format t "Factorial of 6 is: ~A~%" (fact 6))
```

Листинг 5 вычисление факториала на LISP'е

Объективное ориентированные языки (а это практически все современные языки), наоборот, исповедуют диаметрально противоположный подход, навязывая жесткую типизацию даже там, где она на хрен не нужна и только мешает. Каждый Windows-программист знает что такое кастинг (явное преобразование типов данных), и как он раздражает.

Короче говоря, за что боролись за то и напоролись. Языки в процессе своей эволюции выродились в настоящих монстров, ужасающих своей сложностью. Насколько же проще было выучить пару сотен машинных команд, чем освоить Си++! Вместо того, чтобы заниматься решением задачи, программист вынужден воевать с транслятором, пытаясь заставить его сделать то, что он должен.

Как все начиналось? Программисту предоставляется набор хорошо продуманных примитив из которых, как из кирпичиков, и собирается программа. Сейчас же кирпичики сменились готовыми бетонными блоками, к каждому из которых прилагается увесистая спецификация. Программист больше не возводит стены, не устанавливает окна, унитаз и двери — он листает спецификации в поисках модулей, которые лучше всех должны подойти. Программирование медленно, но неуклонно выражается в проектирование. Это в императивных языках можно приступать к программированию даже не имея никакого проекта, со смутными представлениями в голове. Жестких иерархий нет. Исходный код гибок и подвижен. Отрезали отсюда, приклеили сюда. Объективно-ориентированный подход заковывает программу в неподвижный скелет. Неудачно спроектированную систему классов на поздних стадиях разработки уже не исправить и ее придется тащить за собой как асфальтовый каток.

Хуже всего, что языки становятся неотделимыми от среды разработки и библиотек. Можно ли представить себе Си++ без RTL? Что же тогда от него останется?! Не будут работать ни исключения, ни оператор new. А какой же без этого ООП? Тоже самое и со средой. В Microsoft Visual C++ еще можно отказаться от мастеров, но ряд языков (DELPHI, Флора) без нее уже немыслим. Чем плохи мастера? Претензий к ним много, вот только несколько из них: мастера генерируют ужасно запутанный код. Лучше удавиться, чем в таком ковыряться (а без ковыряний дело все равно не обходится, особенно когда в программе требуется что-то доработать или подкрутить). Мастера провоцируют на дурной стиль программирования, перемешивающий интерфейс с реализаций. Мастера делают процесс обучения непоследовательным, нелинейным и трудновоспроизводимым. Вместо исходного текста программы, который можно *начертать* и *прочитать* новичку предлагается последовательность нажатий клавиш, каждая которая бессмысленна сама по себе…

В общем языки программирования вырождаются в жестикуляцию. Но по другому сегодня уже нельзя. Возросший оверхид (от английского overhead – накладные расходы) в прямом смысле слова разувает муху до размеров слона. Представляет интерес проследить эволюцию программы "hello, world! (http://www.sources.ru/news/20010609.shtml). Насколько же проще было программировать на ассемблере под PDP-10, чем на современных "высокоуровневых" языках под OLE/ActiveX! Что ждет нас в будущем? Бесконечный мусор многолетних наслоений или необычайно удобный и элегантный язык?

Прогнозировать трудно. Языки всегда были консервативны. Мы до сих пор находится под влиянием синтаксиса Си и парадигмы объективно-ореентированного программирования. Язык, каким бы замечательным он ни был, находится под властью требований обратной совместимости миллионов исходных строк, которые никто не будет переписывать с нуля. Необходима обширная инфраструктура — компиляторы/интерпретаторы, библиотеки, отладочные средства, учебные пособия… Взять хотя бы Haskell, магическое очарование которого влюбляет в себя с первого взгляда. Многие были бы не прочь писать на нем и только на нем, но… "Я пробовал изучать Haskell, и был впечатлен его элегантностью и тем, как он позволяет мне писать код, который работает с первой попытки (или со второй). Однако, я не исследователь. Я занимаюсь коммерческой разработкой программного обеспечения и мне требуется документация и стабильность" — Alexander Jacobson.

```
fac n
  | n == 0    = 1
  | otherwise = n * fac (n-1)
```

Листинг 6 вычисление факториала на языке Haskell

дележ неубитого медведя или мысли о новом языке

Чего нам не хватает в приплюснутом Си, Даже в прочих языках? Простоты. Именно той простоты и элегантности, на которой сделали карьеру UNIX и классический Си. Кстати, UNIX до сих пор остается простой и дружелюбно настроенной к программисту средой, предлагающей элегантный интерфейс с минимальным оверхидом. Свой первый драйвер под Windows я писал несколько дней, и всю последующую неделю ходил как пришибленный с чумной голой, а под LINUX загружаемый модуль ядра я написал через несколько минут (!) после того, как открыл документацию. "Как просто… Невероятно… Так не бывает…" носилось у меня в голове.

Гениальность создателей UNIX'а в том, что они предоставили унифицированный интерфейс ко всем компонентам системы. Одна и так же функция открывает и файл, и устройство, и даже оперативную память! Microsoft же на все заводит свой API. Отсюда и неподъемная сложность. Библиотеки (типа MFC) ничего не меняют. Вместо того, чтобы учить API-операционной системы, мы вынуждены учить и API, и разлапистую иерархию библиотек. Тот же самый пресловутый ActiveX/OLE можно было сделать намного, намного проще если подойти к делу с головой.

Итак, новый язык прежде всего должен быть прост. Платформа .NET на эту роль явно не тянет. Да, она похоже на нее и не претендует. Это узкоспециализированный набор языков, ориентированный на базы данных и клиент серверные приложения, но ни словом ни делом не поддерживающий ни векторное, ни параллельное программирование. А зря! Тактовая частота не резиновая и быстродействию процессоров очень скоро наступит предел. Многопроцессорные системы уже не за горами (достаточно вспомнить хотя бы тот же Hyper Threading чего стоит). Существует множество диалектов приплюсного Си, поддерживающих параллельное программирование, существуют библиотеки, поддерживающие векторные команды (они же "мультимедийные"), но все это жалкие костыли. Неудобные, ни с чем не совместимые да к тому же недостаточно эффектные. Это как морская свинка. Уже не морская, но уже не свинка. Векторно-параллельное программирование требует совсем другого подхода и типа мышления, а значит и принципиально нового языка.

В новом языке (если такой вдруг появится) скорее всего будет встроен автоматический сборщик мусора (эта такая штука, которая освобождает неиспользуемую память) и поддержка динамических массивов с контролем границ (чтобы хакеры не могли переполнить буфер и заслать агрессивно настроенного червя). Стек, динамическая и глобальная память наверняка атрофируются, хорошо есть сам термин "память" продолжит существовать. Программирование без переменных не только возможно, но и удобно! Синтаксис станет более свободным демократичным, а контроль ошибок — ужесточится до пределе. В идеале все синтаксические ошибки транслятор будет исправлять сам. Наследие логических языков (типа Пролога) также найдет себе применение. Программист говорит, что он хочет, а компилятор/интерпретатор решает как это реализовать.

Сколько это будет жрать ресурсов, лучше даже не представлять, а то ведь и свихнуться можно. Любые траты, направленные на упрощение программирования, по определению оправданы. Ведь это компьютер создавался, чтобы служить человеку, а не наоборот. Трагедия в том, что по-настоящему удобный язык никогда не будет коммерчески успешным. Такой язык не требует ни полноводного потока учебной литературы, ни кусов повышения квалификации, ни армии консультантов… К тому же язык программирования нельзя сеть и выдумать просто так. Настоящие языки возникают лишь естественным путем.

заключение или пара пинков на последок

Каким бы плохим/хорошим ни был приплюснутый Си, закидывать его камнями — это не хакерский путь. А почему бы не попробовать создать свой собственный язык? Запрограммировать на нем пару-тройку программ, показать друзьям, выложить в сеть… Не стоит ждать пока кто-то это сделает за нас. Не стоит хвататься за уже существующие языки только потому, что работодатели их требуют. Работодатели у нас довольно демократичные и зачастую им самим приходится объяснить, что они требуют, а что нет. Конечно, подавляющее большинство отечественных фирм не могут позволить себе финансировать разработку нового языка и уж тем более не в состоянии продвинуть его на рынок, но… практически каждый второй студен вынашивает идею собственного языка и каждый десятый создает минимально работающий интерпретатор, а в некоторых случаях и компилятор, в ходе процессе написания

которого решаются сложные технические задачи и рождаются интересные идеи. Может быть, прямо здесь, в этот день, в этот час, в эту секунду, кто-то пишет язык, которому предстоит стать основным языком программирования на последующие полсотни лет.

>>> врезка классификация языков программирования

- **Императивные** — языки, представляющие собой последовательность команд, в основном оперируют ветвлениями и операциями присвоения;
- **Декларативные** — языки, описывающие отношения между данными. Алгоритм скрывается семантикой языка;
- **Функциональные** — языки, ориентированные на операции с функциями. Переменные и ветвления в них либо вообще отсутствуют, либо практически не используются;
- **Логические** — языки, представляющие собой совокупность предикатов и отношений типа p(x, Y) Программы на языках логического программирования выражены как формулы математической логики, а компилятор пытается получить следствия из них;
- **Объектно-ориентированные** — языки, оперирующие объектами, представляющих "вещь в себе" и обменивающимися "сообщениями";

>>> врезка интересные ссылки

- **языки программирования через сто лет**
  - интересная статья Пола Грема (автор классических учебников On Lisp и ANSI Common Lisp), в которой он делится своими пророческими соображениями по поводу будущих языков (на русском языке) http://www.computerra.ru/hitech/35042/;
- **433 Examples in 132 (or 162*) programming languages**
  - 433 примера программ на 132 языках — чрезвычайно увлекательное и познавательное сравнение различных идей и парадигм программирования (на немецком языке) http://www.ntecs.de/old-hp/uu9r/lang/html/lang.en.html
- **"Hello World" in 65 verschiedenen Sprachen**
  - тоже самое, что и предыдущее, только малость победнее, да поскучнее, тем не менее все-таки представляет определенный исследовательских интерес (на немецком языке) http://www.ntecs.de/old-hp/s-direktnet/sprachen.htm
- **Programming Language Popularity**
  - сравнение различных языков программирования по популярности, периодически обновляемое (на английском языке) http://www.dedasys.com/articles/language_popularity.html
- **The Language List**
  - энциклопедия по языкам программирования настоящего и будущего – краткая сводная информация более, чем по 2.500 языков (на английском языке) http://people.ku.edu/~nkinners/LangList/Extras/langlist.htm
- **Computer Languages History**
  - хронология языков программирования с отслеживанием родственных связей между ними (Исаак родил Абрама, Абрам родил Фортран…) — 10 листов формата A4 в печатном виде. Однозначный must have (на английском языке) http://www.levenez.com/lang/
- **Про компиляторы**
  - сравнительная характеристика различных языков программирования (на русском языке) http://aplprogs.altnet.ru/books/language.htm