Advances in Programming Languages

APL1: What's so important about language?

Ian Stark and David Aspinall

School of Informatics
The University of Edinburgh

Tuesady 21 September 2010
Semester 1 Week 1

Vital Statistics

Course: Advances in Programming Languages

Lecturers: Ian Stark and David Aspinall

Level: 10-credit level 10, for undergraduate year 4 and MSc students

When: 10am–11am Tuesday & Friday

Where: William Robertson Building G.02

Web: http://www.inf.ed.ac.uk/teaching/courses/apl

Blog: http://blob.inf.ed.ac.uk/aplcourse

What it is about computers?

What it is about computers?

Scale

Nanometres, terabytes, gigahertz, megabits/second; the internet, lifebits and data smelters.

What it is about computers?

Scale

Nanometres, terabytes, gigahertz, megabits/second; the internet, lifebits and data smelters.

Digitization

Analogue paper, images, film, music, sound; printers, cameras, telephones, copiers; all now just bits.

What it is about computers?

Scale

Nanometres, terabytes, gigahertz, megabits/second; the internet, lifebits and data smelters.

Digitization

Analogue paper, images, film, music, sound; printers, cameras, telephones, copiers; all now just bits.

Programmability

The computer is *protean*, capable of assuming many forms.

All three are significant, but are mutually dependent for their effectiveness.

Easy Exercises

1. Write down three programming languages.
2. Write down three language paradigms or characteristics.
3. Write down three reasons to choose a particular language.

What matters in a programming language?

We might like a language that is:

- Easy to learn, quick to write, expressive, concise, powerful, supported, well-provided with libraries, cheap, popular, . . .

It might help us to write programs that are:

- Readable, correct, fast, reliable, predictable, maintainable, secure, robust, portable, testable, verifiable, composable, . . .

It might help us address challenges in:

- Multicore architectures, distributed computing, warehouse-scale computation, programming the web, quantum computing, . . .

Shaping the conceivable

Languages frame the way we think, and the programs we can imagine.

Sapir-Whorf Hypothesis

We dissect nature along lines laid down by our native language

This claim is not without controversy; both in its original domain of linguistics, and as more recently applied to programming languages.

Shaping the conceivable

Languages frame the way we think, and the programs we can imagine.

Sapir-Whorf Hypothesis

We dissect nature along lines laid down by our native language

Boole: Language is an instrument of human reason, not merely a medium for the expression of thought [An Investigation of the Laws of Thought, 1854]

Shaping the conceivable

Languages frame the way we think, and the programs we can imagine.

Sapir-Whorf Hypothesis

We dissect nature along lines laid down by our native language

Boole: Language is an instrument of human reason, not merely a medium for the expression of thought [An Investigation of the Laws of Thought, 1854]

Wittgenstein: The limits of my language mean the limits of my world [Tractatus Logico-Philosophicus, 1922]

Shaping the conceivable

Languages frame the way we think, and the programs we can imagine.

Sapir-Whorf Hypothesis

We dissect nature along lines laid down by our native language

Boole: Language is an instrument of human reason, not merely a medium for the expression of thought [An Investigation of the Laws of Thought, 1854]

Wittgenstein: The limits of my language mean the limits of my world [Tractatus Logico-Philosophicus, 1922]

Orwell: The purpose of Newspeak was not only to provide a medium of expression for the world-view and mental habits proper to the devotees of Ingsoc, but to make all other modes of thought impossible [*1984*, 1949]

Shaping the conceivable

Languages frame the way we think, and the programs we can imagine.

Sapir-Whorf Hypothesis

We dissect nature along lines laid down by our native language

Boole: Language is an instrument of human reason, not merely a medium for the expression of thought [An Investigation of the Laws of Thought, 1854]

Wittgenstein: The limits of my language mean the limits of my world [Tractatus Logico-Philosophicus, 1922]

Orwell: The purpose of Newspeak was not only to provide a medium of expression for the world-view and mental habits proper to the devotees of Ingsoc, but to make all other modes of thought impossible [*1984*, 1949]

Perlis: A language that doesn't affect the way you think about programming, is not worth knowing [Epigrams on Programming, 1982]

That's a bit philosophical

Does this really happen? Can programming languages help us write new kinds of program? Or even manage to stop us from writing bad ones?

That’s a bit philosophical

Does this really happen? Maybe.

- LISP S-expressions, metaprogramming, treating code as data.
- Higher-order functions. For example, *parser combinators*:

  ```
  expr = (expr ‘then‘ opn ‘then‘ expr) ‘or‘ term
  opn  = (char ’+’) ‘or‘ (char ’-’)
  term = ...
  ```

- Objects: packaging private state with methods to act on it.
- Laziness for infinite datastructures:

  ```
  odds = 3 : map (+2) odds
  fibs = 1 : 1 : [ a+b | (a,b) <- zip fibs (tail fibs) ]
  ```

- [Your suggestion here. . .]

Properties

One of the defining feature of computers is that they are *programmable*.

Programmability means that computers can always do more. Best of all, you can program new ways to program.

Properties

One of the defining feature of computers is that they are *programmable*.

Programmability means that computers can always do more. Best of all, you can program new ways to program.

Turing writing about the Automatic Computing Engine ACE:

> Instruction tables will have to be made up by mathematicians with computing experience and perhaps a certain puzzle-solving ability. There will probably be a good deal of work of this kind to be done, ...

Properties

One of the defining feature of computers is that they are *programmable*.

Programmability means that computers can always do more. Best of all, you can program new ways to program.

Turing writing about the Automatic Computing Engine ACE:

> Instruction tables will have to be made up by mathematicians with computing experience and perhaps a certain puzzle-solving ability. There will probably be a good deal of work of this kind to be done, ...
>
> This process of constructing instruction tables should be very fascinating. There need be no real danger of it ever becoming a drudge, for any processes that are quite mechanical may be turned over to the machine itself.

[Proposed Electronic Calculator, 1945]

Properties

One of the defining feature of computers is that they are *programmable*.

Programmability means that computers can always do more. Best of all, you can program new ways to program.

Turing writing about the Automatic Computing Engine ACE:

> Instruction tables will have to be made up by mathematicians with computing experience and perhaps a certain puzzle-solving ability. There will probably be a good deal of work of this kind to be done, ...
>
> This process of constructing instruction tables should be very fascinating. There need be no real danger of it ever becoming a drudge, for any processes that are quite mechanical may be turned over to the machine itself.

[Proposed Electronic Calculator, 1945]

That is:

> If you don't like the computer you have, you can create a better one

[Miller, LtU, 2009-05-11]

Abstraction

The concept of *abstraction* adds significant power to programmability.

Abstractions build upon each other: bytes, strings, arrays, matrices

Abstraction

The concept of *abstraction* adds significant power to programmability.

Abstractions build upon each other: bytes, strings, arrays, matrices, lists, maps, trees

Abstraction

The concept of *abstraction* adds significant power to programmability.

Abstractions build upon each other: bytes, strings, arrays, matrices, lists, maps, trees, pointers, files, sockets, objects, databases

Abstraction

The concept of *abstraction* adds significant power to programmability.

Abstractions build upon each other: bytes, strings, arrays, matrices, lists, maps, trees, pointers, files, sockets, objects, databases, instructions, procedures, functions, threads, agents, behaviours, . . .

Abstraction

The concept of *abstraction* adds significant power to programmability.

Abstractions build upon each other: bytes, strings, arrays, matrices, lists, maps, trees, pointers, files, sockets, objects, databases, instructions, procedures, functions, threads, agents, behaviours, . . .

Abstraction frees up you to think about other things, and you should. Let the machine get on with its job.

Abstraction

The concept of *abstraction* adds significant power to programmability.

Abstractions build upon each other: bytes, strings, arrays, matrices, lists, maps, trees, pointers, files, sockets, objects, databases, instructions, procedures, functions, threads, agents, behaviours, . . .

Abstraction frees up you to think about other things, and you should. Let the machine get on with its job.

Whitehead: Civilization advances by extending the number of important operations which we can perform without thinking about them.

Abstraction

The concept of *abstraction* adds significant power to programmability.

Abstractions build upon each other: bytes, strings, arrays, matrices, lists, maps, trees, pointers, files, sockets, objects, databases, instructions, procedures, functions, threads, agents, behaviours, . . .

Abstraction frees up you to think about other things, and you should. Let the machine get on with its job.

Whitehead: Civilization advances by extending the number of important operations which we can perform without thinking about them. Operations of thought are like cavalry charges in a battle — they are strictly limited in number, they require fresh horses, and must only be made at decisive moments. [Introduction to Mathematics, 1911]

What's in the course?

The lectures will cover five sample areas of "advances in programming languages":

- Programming for concurrent code
- Types and Classes in Haskell
- LINQ and cross-language integration in .NET
- Augmented languages for correctness and certification
- Bidirectional programming

Lectures also specify reading and exercises on the topics covered. This homework is not assessed, but it is essential in order to fully participate in the course.

There is substantial piece of written coursework which contributes 20% of students' course grade. This requires investigation of a topic in programming languages and writing a 10-page report with example code.

Time plan

Week 1	Tuesday 21 September	Friday 24 September
Week 2	Tuesday 28 September	Friday 1 October
Week 3	Tuesday 5 October	Friday 8 October
Week 4	Tuesday 12 October	Friday 15 October
Week 5	Tuesday 19 October	Friday 22 October
Week 6	Tuesday 26 October	Friday 29 October
Week 7	Tuesday 2 November	Friday 5 November
Week 8	Tuesday 9 November	Friday 12 November
Week 9	Tuesday 16 November	Friday 19 November
Week 10	Tuesday 23 November	Friday 26 November
Week 11	Tuesday 30 November	Friday 3 December

This gives slots.

Time plan

Week 1	Tuesday 21 September	Friday 24 September
Week 2	Tuesday 28 September	Friday 1 October
Week 3	Tuesday 5 October	Friday 8 October
Week 4	Tuesday 12 October	Friday 15 October
Week 5	Tuesday 19 October	Friday 22 October
Week 6	Tuesday 26 October	Friday 29 October
Week 7	Tuesday 2 November	Friday 5 November
Week 8	Tuesday 9 November	Friday 12 November
Week 9	Tuesday 16 November	Friday 19 November
Week 10	Tuesday 23 November	Friday 26 November
Week 11	Tuesday 30 November	Friday 3 December

This gives 22 slots.

Time plan

Week 1	Tuesday 21 September	Friday 24 September
Week 2	Tuesday 28 September	Friday 1 October
Week 3	Tuesday 5 October	Friday 8 October
Week 4	Tuesday 12 October	Friday 15 October
Week 5	Tuesday 19 October	Friday 22 October
Week 6	Tuesday 26 October	Friday 29 October
Week 7	Tuesday 2 November	Friday 5 November
Week 8	Tuesday 9 November	Friday 12 November
Week 9	Tuesday 16 November	Friday 19 November
Week 10	Tuesday 23 November	Friday 26 November
Week 11	Tuesday 30 November	Friday 3 December

This gives 20 slots.

Time plan

Week 1	Tuesday 21 September	Friday 24 September
Week 2	Tuesday 28 September	Friday 1 October
Week 3	Tuesday 5 October	Friday 8 October
Week 4	Tuesday 12 October	Friday 15 October
Week 5	Tuesday 19 October	Friday 22 October
Week 6	Tuesday 26 October	Friday 29 October
Week 7	Tuesday 2 November	Friday 5 November
Week 8	Tuesday 9 November	Friday 12 November
Week 9	Tuesday 16 November	Friday 19 November
Week 10	Tuesday 23 November	Friday 26 November
Week 11	Tuesday 30 November	Friday 3 December

This gives 18 slots, with a coursework week.

Time plan

Week 1	Tuesday 21 September	Friday 24 September
Week 2	Tuesday 28 September	Friday 1 October
Week 3	Tuesday 5 October	Friday 8 October
Week 4	Tuesday 12 October	Friday 15 October
Week 5	Tuesday 19 October	Friday 22 October
Week 6	Tuesday 26 October	Friday 29 October
Week 7	Tuesday 2 November	Friday 5 November
Week 8	Tuesday 9 November	Friday 12 November
Week 9	Tuesday 16 November	Friday 19 November
Week 10	Tuesday 23 November	Friday 26 November
Week 11	Tuesday 30 November	Friday 3 December

This gives 18 slots, with a coursework week.

The lecture on Friday week will be about the coursework investigation: by then you should have read about the topics available.

Time plan

Week 1	Tuesday 21 September	Friday 24 September
Week 2	Tuesday 28 September	Friday 1 October
Week 3	Tuesday 5 October	Friday 8 October
Week 4	Tuesday 12 October	Friday 15 October
Week 5	Tuesday 19 October	Friday 22 October
Week 6	Tuesday 26 October	Friday 29 October
Week 7	Tuesday 2 November	Friday 5 November
Week 8	Tuesday 9 November	Friday 12 November
Week 9	Tuesday 16 November	Friday 19 November
Week 10	Tuesday 23 November	Friday 26 November
Week 11	Tuesday 30 November	Friday 3 December

This gives 18 slots, with a coursework week.

The lecture on Friday week will be about the coursework investigation: by then you should have read about the topics available.

You must choose a topic by the end of Week 3,

Time plan

Week 1	Tuesday 21 September	Friday 24 September
Week 2	Tuesday 28 September	Friday 1 October
Week 3	Tuesday 5 October	Friday 8 October
Week 4	Tuesday 12 October	Friday 15 October
Week 5	Tuesday 19 October	Friday 22 October
Week 6	Tuesday 26 October	Friday 29 October
Week 7	Tuesday 2 November	Friday 5 November
Week 8	Tuesday 9 November	Friday 12 November
Week 9	Tuesday 16 November	Friday 19 November
Week 10	Tuesday 23 November	Friday 26 November
Week 11	Tuesday 30 November	Friday 3 December

This gives 18 slots, with a coursework week.

The lecture on Friday week will be about the coursework investigation: by then you should have read about the topics available.

You must choose a topic by the end of Week 3, with the final report by the end of Week 8.

Communication

Web

http://www.inf.ed.ac.uk/teaching/courses/apl/

The course web page gives basic information, and through the semester will carry lecture slides, details of coursework and exams.

Lecturers

The most effective way to contact either lecturer is by personal email, from your University email address. However, many questions are even better posed through comments on the course blog.

The mailing list apl-students@inf.ed.ac.uk reaches all APL students and staff. Check http://lists.inf.ed.ac.uk/ to see that you are listed correctly.

Blog

http://blob.inf.ed.ac.uk/aplcourse/

You should read the course blog. It carries the lecture log, slides, and information about homework exercises.

You can add comments, and respond to the questions of others. Please do.

Crystal ball gazing

Some areas to watch, and possible drivers of future language design:

- Multicore
- Weak memory models
- General-purpose computing on GPUs, FPGAs
- Warehouse-scale computing and upwards
- {Cloud,mobile,web} computing
- Dynamic languages
- Certified compilation
- Quantum computing

Don't take this too seriously: some of these have been on the "soon to be hot" list for decades. What would you put on your list? What's next?

See Nature 429:423–429; and Venter's *Synthia*

Crystal ball gazing

Some areas to watch, and possible drivers of future language design:

- Multicore
- Weak memory models
- General-purpose computing on GPUs, FPGAs
- Warehouse-scale computing and upwards
- {Cloud,mobile,web} computing
- Dynamic languages
- Certified compilation
- Quantum computing

Don't take this too seriously: some of these have been on the "soon to be hot" list for decades. What would you put on your list? What's next?

See Nature 429:423–429; and Venter's *Synthia*

Homework

The next lecture is at 10am on Friday. It's about programming for concurrency. Before then:

1. Read the Wikipedia article on *History of programming languages*. (If you find it's missing something, fix that.)
2. Pick a programming language, and find out what support (if any) it offers for concurrency.

   Then post a brief comment on the blog entry for this lecture describing what you have found out.

   Try to avoid duplication — and no more than one language each, leave some for others.
3. Find out about the *Blub Paradox*.

The Secret Agenda of the Functional Illuminati

All advances in the design of mainstream programming languages shall arise from existing functional languages.

Everything necessary can be found by contemplation of ML or Haskell.

The exceptionally adept may already discern all these in LISP.

- ✓ Automatic memory management (everywhere these days)
- ✓ Exceptions (ditto)
- ✓ Parametric polymorphism (see Java/C# generics)
- ✓ Implicit pointers (any OO language)
- ✓ First-class functions (C# delegates)
- ✓ Immutable values (see Java `string`)
- ✓ Closures (lambdas in C#, Visual Basic 9, maybe C, Java 7?)
- ? Algebraic datatypes (still trying, but see Scala)
- ? First-class continuations (. . .)