Advances in Programming Languages
What’s So Important About Language?

Ian Stark

School of Informatics
The University of Edinburgh

Tuesday 20 September 2016
Semester 1 Week 1
Key Information

Course: Advances in Programming Languages

Lecturer: Ian Stark

Level: Undergraduate year 4, year 5 and MSc students
(10 credit points at Level 11)

When: 1510–1600 Tuesday & Friday

Where: George Square Lecture Theatre (or possibly elsewhere...)

Web: https://blog.inf.ed.ac.uk/apl16
Research Mission

More and more people are programming: not only those with traditional Computer Science backgrounds, but also physical scientists, journalists, and more. We have an unprecedented opportunity to shape how people reason about software — and the systems that people do not yet call software. The goal of my research is to make provable guarantees ubiquitous. I am interested in designing programming models, language implementation strategies, and programming tools to make it easier for people to create the software they intended. Application domains of interest include security, privacy, and biological modeling.
What it is about computers?

Scale

Nanometres, exabytes, gigahertz, megabits/second; the internet, genomes, 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.
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, ...
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.

**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]
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 write us to new kinds of program? Or even stop us from writing bad ones? Maybe.

- LISP S-expressions, metaprogramming, treating code as data.
- Higher-order functions. For example, *parser combinators*:
  
  \[
  \text{expr} = (\text{expr} \text{ 'then' \ opn \ 'then' \ \text{expr}}) \text{ 'or' \ \text{term}} \\
  \text{opn} = (\text{char '+')} \text{ 'or' (char '-')} \\
  \text{term} = \text{variable 'or' constant 'or' ...}
  \]

- Objects: packaging private state with methods to act on it.

- Laziness for infinite datastructures:
  
  \[
  \text{odds} = 3 : \text{map (+2) odds} \\
  \text{fibs} = 1 : 1 : [ a+b \mid (a,b) \leftarrow \text{zip fibs (tail fibs)} ]
  \]

  [Your suggestion here...]
Programming languages may shape the way we think about problems, and about solutions.

What can we do to shape the programming languages?

Well, because computers are programmable, we can use them.
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, 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 four sample areas of “advances in programming languages”:

- Types: Parameterized, Polymorphic, Dependent, Refined
- Programming for Concurrency
- Augmented Languages for Correctness and Certification
- Programming for Memory Safety

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 your course grade. This requires investigation of a topic in programming languages and writing a 10-page report with example code.
Communication

Web

The APL web site gives detailed information about the course. Through the semester it will carry copies of the lecture slides, course blog, coursework assignments and past exam papers.

Lecturer

The most effective way to contact the lecturer is by personal email, from your University email address. However, many questions are even better asked by posting to the course mailing list.

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

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

- Manycore, multicore
- General-purpose computing on GPUs, FPGAs
- Warehouse-scale computing and upwards
- {Cloud, mobile, web} computing
- Metaprogramming
- Trustworthy code
- 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?

e.g., The *Edinburgh Genome Foundry* will create DNA strands of up to $10^6$ base pairs to “equip cells or whole organisms with new or improved functionality”. In what languages do you program that functionality?
Summary

- **Abstraction**: Lift the level of operations you can describe
- **Programmability**: Build a new computer from the one you have
- **Expression**: Broaden your thoughts and the programs you can imagine

“To me programming is more than an important practical art. It is also a gigantic undertaking in the foundations of knowledge.”

David Sayre (one of the creators of FORTRAN) in conversation with Grace Hopper (one of the key advocates for COBOL), 1962
Homework

The next lecture will be at 1510 on Friday. It’s about the use of types in programming languages.

Check the blog to find out where.

1. Read the Wikipedia article on *History of programming languages*. (If you find it’s missing something, fix that.)

2. Pick a programming language you don’t already know, and find out the following.
   - Does it assign types to distinguish between things like numbers, strings, or functions?
   - Does it check these are used correctly?
   - How does it do that? When does it do that?

Bring your answers along to the lecture.