Introduction to Programming Languages

Thanks to Jessica Bayliss

Class Goals
are on the syllabus

Some Real-Life Issues
Pragmatic programmers automate routine tasks. How will you automate routine task x?
Your boss tells you that language x is a “hot language” and the greatest thing since sliced bread. Your boss says that all future projects will be done in language x. What do you say to your boss? How do you learn language x?
It has been determined that the simulation language you used for project x is becoming a dead language. What do you do?

Comparing Languages
Design Criteria for a Language [Programming Languages, Tucker and Noonan]
Simplicity and clarity
Binding – early or late?
Orthogonality – does the language behave as expected?
Reliability of programs
Applicability – languages are usually designed for a particular domain
Abstraction – Gives you the ability to avoid reinventing the wheel
Efficient Implementation

Design Criteria [Wilson & Clark]
Expressive Power
Simplicity and orthogonality
Implementation
Error detection and correction
Correctness and standards

But remember that an early bad reputation or misunderstanding is hard to overcome
Java is slow
Lisp is interpreted
etc.

Examples
Orthogonality:
String literals and == in Java
Overloading the + operator for different reasons

Abstraction:
Java contains stacks
Functional arguments - closures
lambda is sometimes called the abstraction operator

Simplicity:
Visual Basic

Efficient Implementation
Algo 68 was an elegant language design, but its spec’s were so complex that it was nearly impossible to effectively implement.
Early versions of Ada were criticized for their inefficient run-time characteristics since Ada was designed in part to support programs that run in “real time” – critics were heard to utter, “Well, there’s real time and then there’s Ada time!”
Java has been criticized for its early run-time performance.

Programming Domains
Scientific computing
Management information systems
Artificial Intelligence
Systems
Web-centric

Programming Paradigms
Imperative programming
OO programming
Functional programming
Logic (declarative) programming
Event-driven programming
Concurrent programming

Fortran
‘The IBM Mathematical FORmula TRANslating System’
“The only solution was to design a language for scientific computations that allowed the programmer to use mathematical notation.
However, the designers of Fortran felt that this had to be done in a way that produced efficient object code, otherwise practicing programmers would reject the language if they could produce a hand coded version that ran much faster than the compiled Fortran program.”
String handling facilities were almost non-existent and the only data structure was the array.

Algol
‘ALGOrithmic Language’
“No other language has had such a profound influence on programming language design and definition as that of Algol.”

Objectives of the language:
– It should be as close as possible to standard mathematical notation and be readable without too much additional explanation.
– It should be possible to use it for the description of computing processes in publications.
– It should be mechanically translatable into machine code.
Algol was the first language to use a formal syntax definition - BNF

Why wasn’t Algol as popular as Fortran?
Algo 60 compilers came out 3 years after Fortran – Fortran was already entrenched
Since Algol 60 had more features, it was harder to learn
Although IBM initially supported Algol, they later decided to stick with Fortran
Fortran compilers were simpler and produced more efficient code.
Algol 60 had no official I/O. It was decided to leave this to the individual manufacturers so they could tailor it to their computers.
PL/1
(Originally called NPL for New Programming Language but
the UK’s National Physical Laboratory complained)

Programming Language 1
In the early 1960’s, two kinds of programmers existed:
Scientific programmer (usually used Fortran)
Commercial user (needed decimal arithmetic, efficient
searching, sorting, string manipulation)

Designed by IBM for the IBM 360
Guiding principles of design:
A programmer’s time is an important asset and should not
be wasted
There is a unity in programming which the current division
between scientific and commercial languages did not reflect
This meant that there were to be as few machine
dependencies as possible while allowing the programmer to
have full access to machine and operating system facilities
without resorting to assembly language coding.

A large language was needed…
Programmers wouldn’t need to know all the features of the
language to be able to use it efficiently.
Every attribute of a variable, every option and every
specification had a default interpretation and this was set to
be the one most likely to be required by a programmer who
does not know that alternatives exist.

PL/1 attempted to have both run-time efficiency and
flexibility, but the penalty it paid was language complexity.
It was not highly successful. The compiler was large and
slow and it never replaced either Fortran or COBOL let alone
both.

MULTICS was written in PL/1 and had a very good compiler
Cornell implemented a version known as "PL/C"

As Timesharing became more Popular…
Interactive languages became more popular

QUICKTRAN: Interactive Fortran
Basic (Beginner’s All Purpose Symbolic Instruction
Code): meant to be a student’s language

Special Purpose Languages
String manipulation languages:
SNOBOL4: did string pattern matching

List processing languages:
IPL-5
Lisp

Lisp
Based on Church’s lambda calculus
Principal features:
It performs computations with symbolic expressions
rather than numbers.
It represents symbolic expressions and other
information in the form of list structures in computer
memory
It uses a small set of constructor and selector
operations to create and extract information from lists.
These operations are expressed as functions and use
the mathematical idea of the composition of functions
as a means of constructing more complex functions.
Control is recursive rather than iterative.
Data and programs are equivalent forms. Thus,
programs can be modified as data and data structures
executed as programs.
Implemented storage management with garbage
collection

Other types of languages
Scripting languages:
Perl
Awk
Simulation languages:
Simula: the original object-oriented language
Systems languages:
C/C++

Modularity
Physical decomposition into separate data files
Logical decomposition: raising the level of abstraction
at which programmers can think
Abstract data types

Languages that grew out of this:
– Smalltalk: Everything is an object, didn’t have static type
checking
– Eiffel: Object-oriented with an emphasis on program
correctness. Pre-and post-conditions specify the meaning
of operations. Does static type checking.

Functional Languages
The problem with most languages is that they rely on
side effects
A purely functional language only involves the eval of
expressions, so it is possible to reason formally about
the effect of a functional program and to prove that it
meets its specification.

Logic Languages
Unlike procedural programs, there is very little explicit
control of how the problem is to be solved – the
solution is produced by inference from the given facts
and rules
There are also constraint logic programs that extend
this paradigm to reasoning about strings or numbers
or sets