Course aims

Two main aims:
• Broaden your view of programming languages. Java is not the only way.
• Give an understanding of the general principles common to all programming languages.

Side effect:
• improve your understanding of Java by seeing why things are done in a certain way.
Course aims

Hundreds of different programming languages exist.

There are different kinds/families of language (paradigms).

Programming in a logic language, for example, is very different from programming in Java.

BUT the similarities still outweigh the differences.

Once you know how to program in one language, it is straightforward to learn other related languages.

The emphasis in the course is on showing the similarity and the connection between apparently dissimilar language features.

Organisation

Schedule: see online
Lecture attendance - a good idea!

Textbook (essential reading): Comparative Programming Languages (3rd ed.), R.G.Clark, Publisher: Addison-Wesley
Software Development Process
Programming languages used as part of the software life cycle.
We have the following phases:
• requirements capture,
• problem specification,
• design,
• implementation,
• validation and verification,
• maintenance.

Different life cycle models, e.g. waterfall model and spiral model.
Note that:
• Phases of the software life-cycle are not independent
• The process is iterative
• Knowledge of the implementation language influences the design

The design of modern programming languages has been greatly influenced by the needs of the software development process.
Language Design

The primary purpose of a programming language is to support the construction of **reliable software**. How can the **design of a language** support software reliability?

**Expressive power**

Allow solutions to be expressed in terms of the problem being solved rather than in terms of computer instructions.

---

**Language Design**

**Orthogonality**

Small number of basic constructs which can be combined systematically to give more complex structures. Few, if any, special cases.

*Pragmatic* considerations?

*Law of Minimum Surprise*
Language Design

Error Detection and Correction

Testing.
Errors should be shown up at *compile time*. Why?
Language design example: require the type of all variables to be declared. How will this help?

This leads to the notion of

*strong typing*

– all expressions have a well defined type.

*static typing*

– the type of all variables is fixed at compile time.

• Hence *static strong typing* is where the types of all variables are well defined at compile time.

Fortran Example

Consider the following two statements:

```
DO 20 I = 1,30
DO 20 I = 1.30
```

When a simple typing error is made, it should result in a syntax error, not in a statement with a different meaning.

An American space probe missed the moon due to the above error. Resulted in them giving up Fortran.
Language Design

Correctness

Not just testing to show it works - *prove* it works.
More common with *declarative languages*.

Language Design

Standards

– ISO and ANSI
– Standard definitions give the *syntax* (structure) and *semantics* (meaning) of each language construct.
– Problem of *dialects* (subsets or supersets).
Language Design

Size
Small and (Beautiful? Constrained? Simple?)
Large and (Multipurpose? Bloated? Too Complex?)

Some successful languages were explicitly designed to be small:
Pascal much smaller than Algol 68; Java removed many features from C++.

Is Java large or small?

Language Design

Language Implementation
Imperative programs (e.g. in Pascal, Ada, C, C++) are usually translated (compiled) into an equivalent machine code program which is then executed.

Does this translation affect the acceptability of a language?
Language Design

Language Implementation (continued)
Or, use an interpreter. An interpreter can directly execute a source program. Or (nowadays)

source program $\Rightarrow$ program in a virtual machine language

And the interpreter executes the virtual machine language.

Language Design

Language Implementation (continued)
Java is interpreted for portability reasons. Browsers such as Netscape and Internet Explorer include implementations of the Java virtual machine (JVM).

Otherwise Java has the features of a conventional compiled imperative language.
Brief history of languages

Early machines were very small (1K store, no decent peripherals).
Binary code.

Next step: a symbolic form of machine code (assembly language)
LDA 0
BNE LABEL

Autocodes developed adding expressions and arrays.

Imperative Languages

Observation:
– Computers have individually addressable store locations which can contain instructions or data.
– Machine code programs achieve their effect by changing the contents of store locations.

• There is a lose connection between machine code and imperative languages
• This means efficient implementations.
Imperative Languages

Recall von Neumann architecture: an addressable store, programs as data, fetch-execute cycle.

How does this relate to Imperative languages?
A variable or object corresponds to one or more store locations.
Programs achieve their effect by changing the values of variables or objects.
Instructions are executed in sequence with jump instructions altering the flow.

Fortran

first manual released: 1956; first full implementation: 1958
successful beyond the expectations of its inventors, and influenced all later imperative languages.
aimed at scientific computation
allowed programmers to express solutions in mathematical notation
major selling point: machine independence

A Fortran program consists of a main program, subroutines (like procedures) and functions.
Still widely used; latest version is Fortran 90.
Algod 60

The next major procedural language was Algod 60, designed during 1958-1963.
- The original block-structured language
- Used recursion
Its syntax was defined formally using Backus Naur Form (BNF).

Algod 60 did not replace Fortran, but languages such as Pascal are its direct descendents.
BNF is still the normal way in which the syntax of programming languages is defined.

Later imperative languages

Cobol
Universal languages?
- PL/I was a large language developed by IBM in the mid-sixties.
- Algod 68 was designed as the replacement for Algod 60, but was large, too complex and was never in widespread use.

Smaller languages?
- Pascal was developed around 1970. It cleaned up and extended Algod 60.
- C also developed at that time and took features from Algod and Fortran. Main use was as an efficient systems programming language. It eventually killed off assemblers.
Language Categories

Two kinds of imperative language: *procedural* and *object-oriented*.
Until late 1990s, procedural languages were dominant.
Many procedural languages now have new versions that are object-oriented:
C++ from C, Delphi from Pascal, Ada 95 from Ada 83, Object Cobol from Cobol.
Hybrid rather than pure OO languages.

Modules

Modula 2, Modula 3 and Ada developed from Pascal in late 70s and early 80s.

Their main addition is the *module*.

Ada developed by *US Department of Defense* to be the standard language used on *all* defence contracts.
Modules

Emphasis shift from algorithms to data structures.

Many good features:

– Encapsulation
– Information hiding
– Data abstraction
– Decomposition
– Localising changes
– Component re-use

Modules

In Ada, the module construct is called a package.
Packages can be used to group together the definition of a type and its associated operations.

In C++ and Java, the module is a class; it defines a new type.

Similar but different:

Packages are primarily a program structuring device while classes define a type.
Object-oriented languages

First OO language was Simula 67.
Developed from Algol 60 with the addition of classes.
Defined as a simulation language, but found to be a useful systems language.
Smalltalk developed in 1970s. Its development environment introduced concepts of window, mouse, menu and icon, i.e. the GUI.
Weakly typed and can be used to rapidly implement prototypes. In Smalltalk, everything is an object.

Object-oriented languages

C++ developed in 1980s, by adding classes to C. Hybrid language.
Eiffel developed from Pascal in late 1980s to introduce good software engineering principles to OO programming.
Delphi is an OO extension of Pascal.
Java introduced in mid 1990s as a simplification of C++.
C# developed as Microsoft's answer to Java.
Declarative languages

Functional and logic languages are *declarative languages*.

Declarative languages concentrate on stating *what* is to be done rather than on *how* it is to be done.

**Pro:**
- shorter and more reliable programs
- easier to reason about programs (provably correct programs)

**Con:**
- less efficient

Overview

The course examines different kinds of programming language:

- **Object-oriented languages**, e.g. Java, Smalltalk;
- **Procedural languages**, e.g. Pascal, C, Fortran, Cobol;
- **Logic languages**, e.g. Prolog;
- **Functional languages**, e.g. ML;
- **Scripting Languages**, e.g. Perl, Javascript, PHP

C++ and Ada 95 can be considered to be hybrid object-oriented/procedural languages.