COMPSCI 630 Systems **Spring 2022**

Lecture 1

Lecturer: Emery Berger *Scribe: Juan Pizzorno, Matthew Jamieson*

1.1 Early Computers and Languages: COBOL, FORTRAN

- The Education of a Computer - Grace Murray Hopper
- The FORTRAN Automatic Coding System - J.W. Backus et al
- The Night Watch - James Mickens

The first two papers predate the theory of compilers: parsing, grammars, compilers, the notion of scope, of program analysis, of data types, and even of floating point numbers. These theories and techniques were invented over time to improve upon the the ad-hoc manner in which these first compilers were created.

The early computers themselves were also very different from today's: they were very slow, had very small in memory and storage and were, in fact, many times less capable than people's phones of today. They used vacuum tubes rather than chips, and initially didn't even have screens, but used printers for output.

Programming was done by expert programmers and in assembly language. The instructions were typically specific to the computer being programmed, as each new computer brought its own instruction set.

The instructions were typically simple, so that, for example, a multiplication might have to be expressed in terms of additions in a loop. On a stack-based system, the code would need to load all operation arguments onto the stack and then invoke the appropriate operation. For example, "if a^2 equals 42, then ..." might be expressed as

```
PUSH A
PUSH A
MUL
PUSH 42
CMP
JMPZ 12
```

The original intent and vision behind COBOL and FORTRAN was enabling people other than expert programmers, such as managers and scientists, to write their own programs and have these automatically translated (in today's terms, compiled) into executable code. COBOL was designed primarily with ease-of-use in mind, whereas FORTRAN was designed for high performance.

The divide between these audiences still basically exists, in that some languages emphasise high performance computing needed by scientists and the like (Java, C, C++, Rust, etc) where others emphasise making it easy to read and write programs (Python, Perl, Ruby etc).

COBOL stands for "Common Business Oriented Language" and was English-like, using phrases like

```
MOVE 1 TO I
ADD 2 TO I
```

to express `i = 1` and `i = i + 2`.

FORTRAN stands for ”Formula Translation”; it was intended to allow scientists and engineers to write code in more mathematical terms. Certain variable names, such as `I`, `J` and `K` were always integer-valued (since they are symbols common used as indices in math) while `X` and `Y` were always real-valued (given their use as unknowns in math).

Initial versions of FORTRAN could be very error prone. If the comma in the `DO` loop

```
... DO 10 I = 1,100
    ...
10  CONTINUE
```

was accidentally exchanged for a dot, FORTRAN would interpret that as a `DO10I=1.1` variable assignment. There is an apocryphal story that this exact bug caused the destruction of the Mariner 1 space probe.

Of note: FORTRAN precedes just about all modern notions of programming language design and theory, including: compilers, parsers, types, objects, and so on. The original version of FORTRAN has no notion of scope. For performance reasons, rather than using a stack based approach, FORTRAN uses the concept of a ”common block” to store variables needed for functions. This means that instead of pushing parameters individually to a stack, only a pointer to a contiguous block of memory is passed. The problem with this is that FORTRAN has no system to make sure calls to functions are using the correct arguments and can result in unexpected behaviour.

Early languages were highly influential on future ones (`https://en.wikipedia.org/wiki/Generational_list_of_programming_languages`)

Programming languages came a long way, but they still sport a lot of unexpected behavior, as we saw in the “WAT” lightning talk by Gary Bernhardt (`https://www.destroyallsoftware.com/talks/wat`).

In today’s terms, a variables may have static or dynamic types. A statically typed variable has a name, a data type and a value; anything that goes into that variable needs to have that type. A dynamically typed variable, in contrast, has a name and a value, but the type of its value is contained within the value itself, and depends what the variable was assigned.

Some languages (eg. MyPy, TypeScript) also permit “gradual typing”, in which otherwise dynamically typed variables can receive type declarations, allowing programs to move gradually from dynamic to static typing.

FORTRAN was the first optimizing compiler. By “optimizing” we don’t mean it will necessarily generate optimal code, but that it’ll replace code as written with something equivalent that hopefully runs faster. Compilers use static analysis (which means looking at the source code, rather than *dynamic analysis*, which means looking at the state of the running program) for that. Some of the things optimizing compilers do are:

- constant propagation. They can simplify sequences such as `x = 3; y = x + 12` to `y = 15` so that the value of `y` needn’t be computed at runtime;
- strength reduction. They can substitute operations such as the exponentiation in `x = y^2` for the multiplication `x = y * y`, as that is a faster operation;
- inlining. Certain parts of a program are best expressed as a function, for example, to indicate what they do. Function calls incur in a certain overhead, but a compiler can substitute the function call for the function code directly, as though it had been coded directly in;