CSCC24 – Principles of Programming Languages
Formal Language Theory

Anya Tafliovich[1]

[1]with thanks to S.McIlraigh, G.Penn, P.Ragde

who I am

- Dr. Anya Tafliovich — Anya
- 2002: Hon.BSc: specialist in CS, major in Math
- 2004: MSc in Computer Science
- 2010: PhD in Computer Science
- since 2010: Assistant Prof Teaching Stream at CMS
- since 2016: Associate Prof Teaching Stream at CMS
- since 2022: Professor Teaching Stream at CMS
- Research Interests (in no particular order):
  - Formal Methods of Software Design, Software Verification, Automated Reasoning, Quantum Computing, Programming Languages, Computer Science Education, Software Engineering Education
- Teaching: variety of courses
  - A08/A48, A20, B07, B63, C01, C24, D01, D72, D92, D94/95

who I am

- rock climbing, running, reading, learning to play piano
- should really start practising yoga again...
- two children — 15 and 12 years old
- two cats — 5 year-old

who you are

- program of study?
- year of study?
- programming languages used:
  - in university / school?
  - at work?
  - on your own?
- programming languages you want / plan to learn?

course title

- Language

  A language is an arbitrary association of a collection of forms with their meanings.
  - Syntax: specification of the forms.
  - Semantics: specification of the meanings.

  Examples:
  - in English
  - in Java

- Programming Language

  A programming language is "a set of conventions for communicating an algorithm." (Horowitz)

- Natural vs Programming Languages

on programming

The main idea is to treat a program as a piece of literature, addressed to human beings rather than to a computer. — Donald Knuth

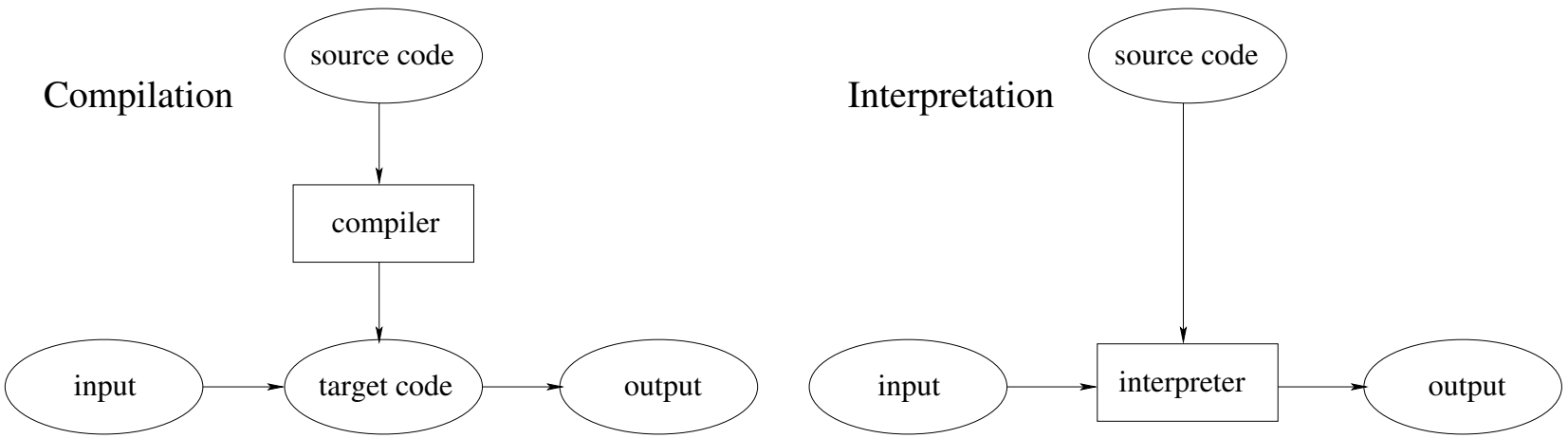
levels of programming languages

- High-level languages.
  - Which high-level languages do you know?
  - What makes them "high-level"?
- Assembly languages.
  - Do you know an assembly language?
  - Is it easy to program in?
  - Why?
- Machine languages.
  - Have you seen a machine language?
  - How would you describe its usability?

translation

The process of converting a program written in a high-level language into machine language is called translation.

There are two general methods.



Compilation: the whole program is translated before execution.
Interpretation: translate and execute, one statement at a time.

translation

Compilation:

Interpretation:

translation

Compilation:

- Can execute translated program many times because the entire translation is produced.
- Program execution is faster because:
  - compilation time is "overhead": done once;
  - the translator can do optimisation;
  - can get rid of lots of information no longer needed for run-time.
- Harder to provide useful feedback when debugging because executing the target code.
- Not easily portable (e.g., to a different OS / architecture / etc.).
- What compiled languages do you know?

translation

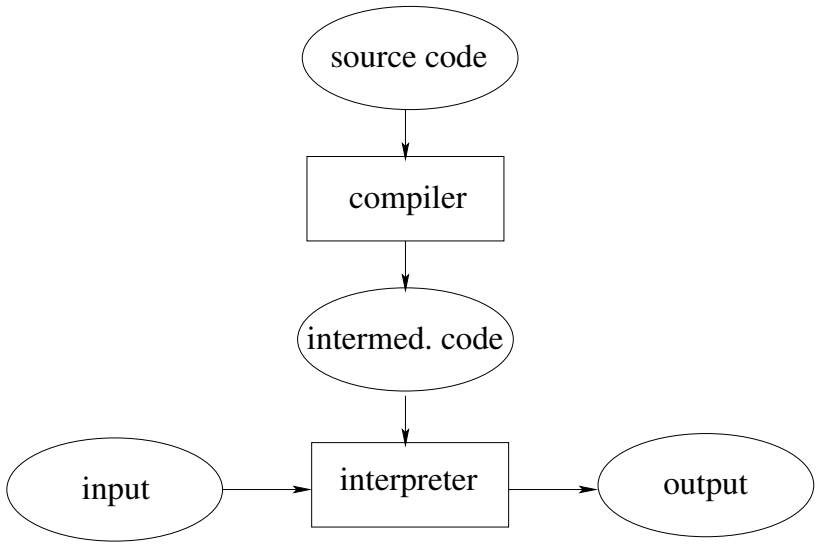
Interpretation:

- Must re-translate for every execution.
- Program execution is slower.
- More space needed at execution time.
- Easier to provide useful feedback when debugging.
- Flexibility supports rapid prototyping.
- Portable!
- What interpreted languages do you know?

Why are so many modern languages interpreted?

translation — pseudo-compilation

A hybrid of compilation and interpretation.



- A compiler translates the whole program before execution, but only into intermediate code.
- An interpreter translates and executes the intermediate code one statement at a time.
- The intermediate code can be executed on any machine that has an interpreter for the intermediate code.

Example: Java's intermediate code is called bytecode.

translation — Just-In-Time (JIT) compilation

- Run in the interpreter.
- Compile at run-time. (Huh??)
- Compile on an "as needed" basis.
- Run-time monitoring:
  - If something runs a lot, compile it.
  - If something runs a whole lot, compile it with optimisations.

thoughts on translation

- We often say that a language is compiled or interpreted.
- Technically, this is incorrect. A language itself is neither. A particular implementation of a language can be compiled or interpreted.
- Many think that the future lies with hybrid implementations, especially JIT compilation and optimisation.
- Bottom line is always: given the task at hand, make an informed decision about which language(s), and which implementation of the language(s) to use.

on programming

The main idea is to treat a program as a piece of literature, addressed to human beings rather than to a computer. — Donald Knuth

Want to write code in a high-level language.

translation

The process of converting a program written in a high-level language into machine language is called translation.
How is this done?

1. Lexical analysis converts source code into sequence of tokens.
2. Syntactic analysis structures tokens into initial parse tree.
3. Semantic analysis annotates parse tree with semantic actions.
4. Code generation produces final machine code.

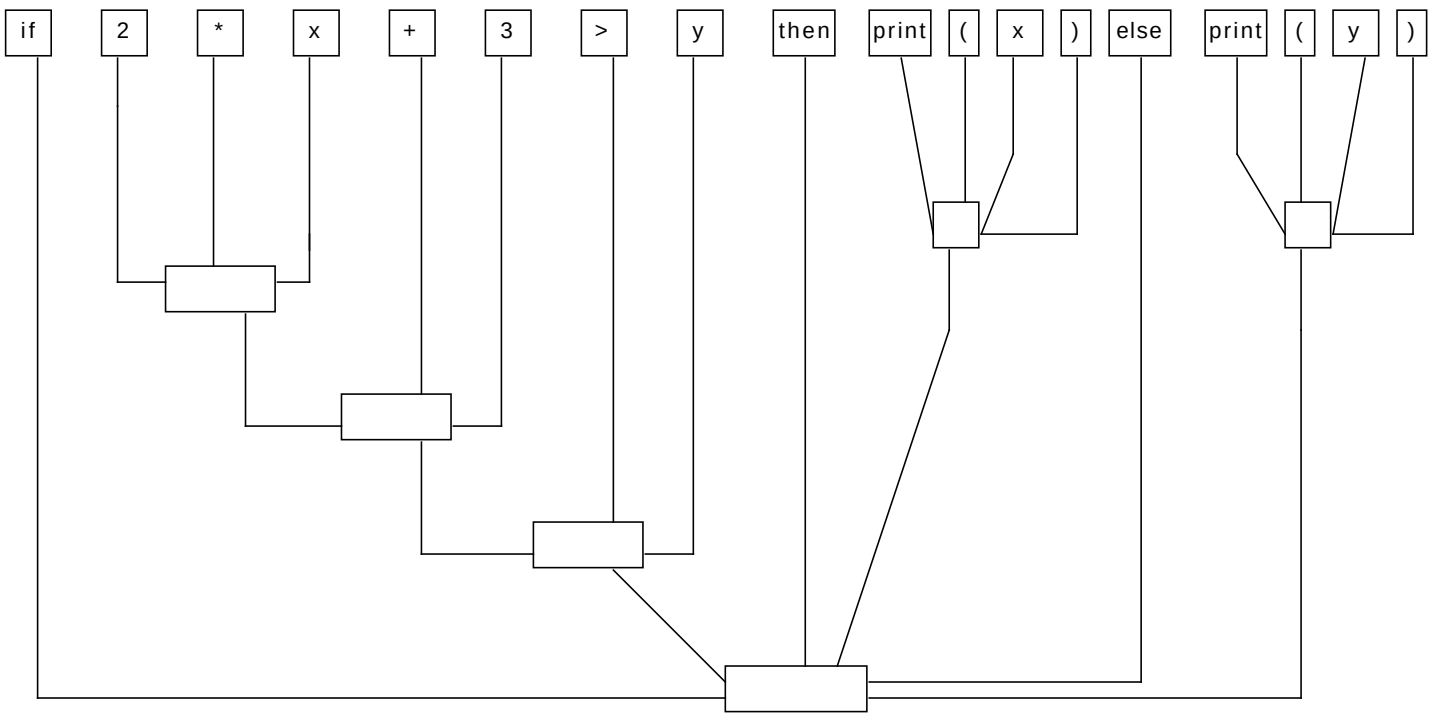
translation — example

```
if 2 * x + 3 > y then print(x) else print(y)
```

Tokens:

```
if 2 * x + 3 > y then print ( x ) else print ( y )
```

Parse tree:



translation — prereq(?) knowledge

- For Lexical analysis:
  Regular expressions, finite state automata.

- For Syntactic analysis:
  Context free grammars, push-down automata.