CSCI-GA-2130: Compiler Construction

Instructor

Joseph Tassarotti

Office: 60 FA 401

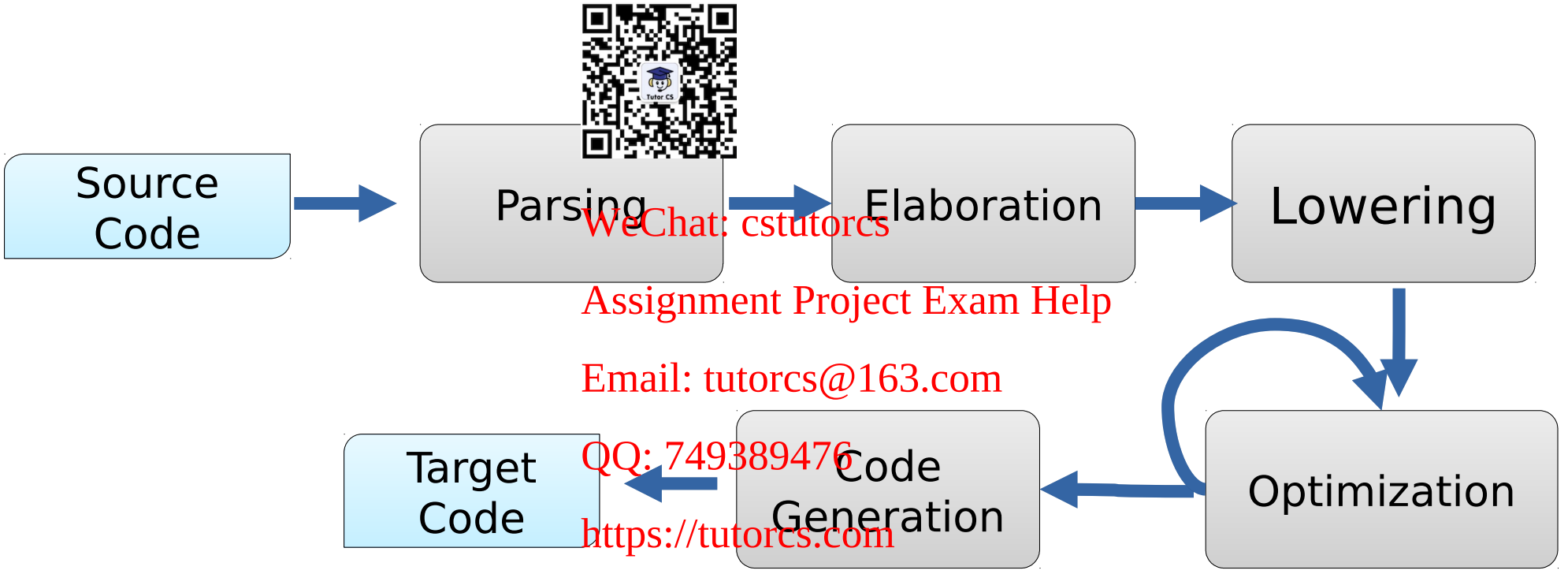
Email: jt4767@nyu.edu

What is this about?

- Compilers translate programming languages to machine code.

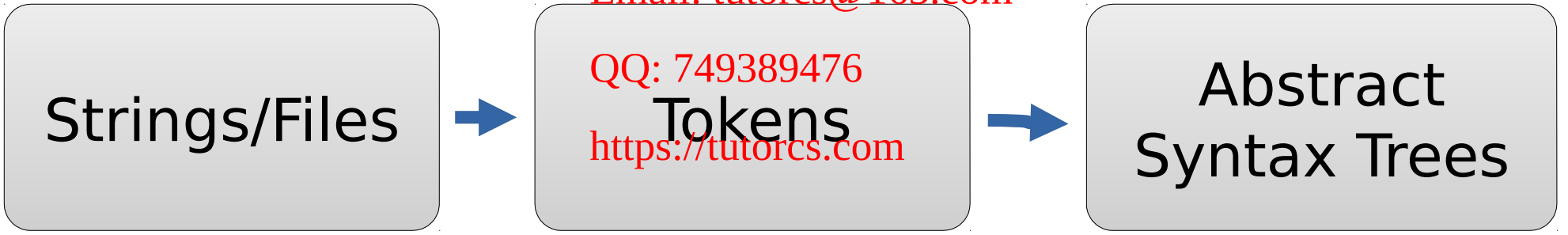
- How do we structure a compiler to be:
  - Correct
  - Maintainable
  - Extensible

Compiler Architecture



Front End

- Convert a string to a data structure:
  - Lexing: Convert raw characters into list of **tokens**
  - Parsing: List of tokens into abstract syntax tree



Elaboration

- Type-checking
- Resolve variables/modules/names
- Check other “well-formedness” properties

Lowering

- Translate high-level features into lower-level constructs:

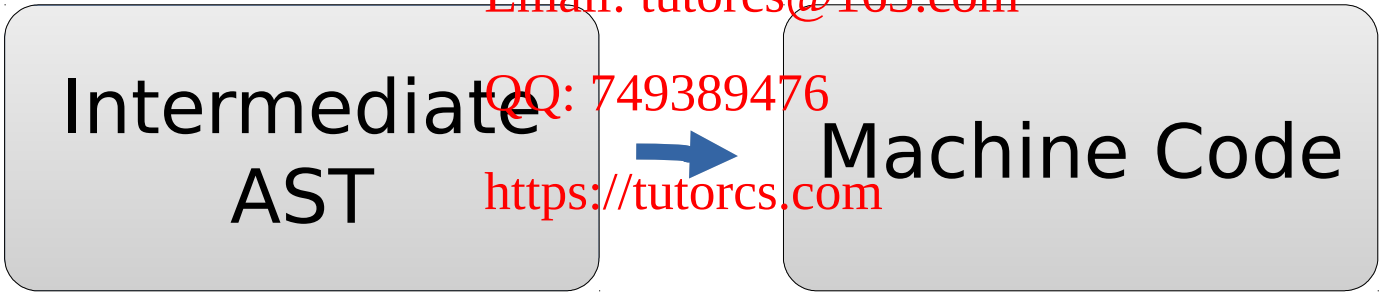
  E.g. while/for replaced by “goto”

Optimization

- Make code more efficient:
  - Remove “dead” code
  - Simplify computations (replace 3+6 with 9)
  - Reuse repeated computations

Code Generation

- Convert Interme code to target:
  - Register Allocation
  - Instruction Selection



Required Background

- Officially:
  - CSCI-GA 1170, CSCI-GA 2110, and CSCI-GA 2250.
- Unofficially:
  - Passing familiarity with basic computer systems/architecture
  - Willingness to work hard to fill gaps

Administrative Details

- Website: BrightSpace*
- Assignment submission: GradeScope

- ~9 assignments determine the entirety grade.
  - 3 late days total throughout semester
  - See collaboration policy on website

Programming Environment

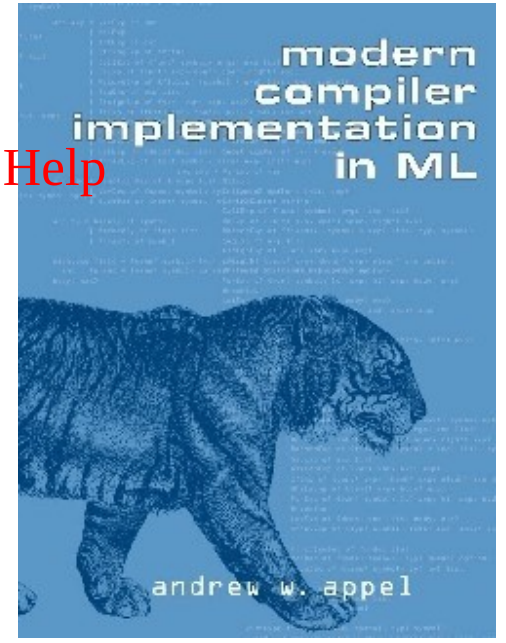
- OCaml
  - Functional programming language, very good for writing compilers
  - Intensive introduction in first week
- SPIM
  - Simulator for MIPS assembly language

Resources

- No official book, but there are many references about OCaml and compilers

- Closest to what we're doing:

  Modern Compiler Implementation in ML by Andrew Appel



Assignments (Tentative)

0. Ocaml warm-up
1. MIPS Simulator
2. Fortran-ish → MIPS
3. C-ish → MIPS
4. Scheme-ish → C-ish
5. ML-ish → Scheme-ish
6. Optimization
7. Control-Flow Graphs
8. Register Allocation

Learning Outcomes

- Understand how compilers work:
  - What can they do?
  - What can they not do?
- Lots of functional programming experience
- Thinking of languages as an extensible programming interface