南方科技大学
SOUTHERN UNIVERSITY OF SCIENCE AND TECHNOLOGY

CS323 Lab 1

Yepang Liu

liuyp1@sustech.edu.cn

Agenda

- Introduction to labs & course project
- Set up tools and environment
- Find teammates for the course project

What will we do in labs?

- Demos and exercises to help you understand the core concepts learnt in lectures
- Tutorials to help you build a working compiler
- Cover some detailed content that cannot be covered in lectures, e.g., complex algorithms, etc.
- Q&A (we won't have much time for questions during lectures)
- …

Lab Attendance & Exercises

- We will check attendance and exercises every week. Lab attendance will account for ~5% of the overall grade of this course.

- Some lab exercises may be difficult. It is fine if you cannot finish all lab exercises during class. You can continue to do the exercises after each class (we will not check).

Course Project (Research + Practice)

- **Team size:** 2-3 students (there is no additional bonus for small-sized or single-person teams)

- **Task 1: Research on language implementation and compilation techniques**
  - Report submission (week 12)
    - Grading: Excellent (90-100), Good (80-89), Fair (70-79), Poor (< 70)
  - Invited presentations (week 15 during lecture)
    - Only for "excellent" ones

Hints for the Research Task

- **Category 1: Lexer and parser technologies**
  - Error recovery: How do modern parsers handle syntax errors and attempt to recover to find more than one error per compile?
- **Category 2: Intermediate Representations (IR) and Optimization**
  - LLVM IR: Design principles, advantages and weaknesses, and applications
  - SSA form: The history of SSA and how does it enable effective optimizations? Compare its implementation in different compilers (LLVM, GCC)
- **Category 3: Runtime Systems and Memory Management**
  - JIT compilation techniques in modern runtimes (e.g., HotSpot JVM, V8 JavaScript engine)
  - Garbage collection algorithms: Mark and sweep, generational, copying, reference counting
- **Category 4: Type systems and semantic analysis**
  - How do modern languages implement type inference?
  - How is the Hindley-Milner type system used in Haskell or OCaml?
- **Category 5: tools and ecosystem**
  - Language server protocols: Survey how a compiler's frontend is leveraged to provide features like "go-to-definition," autocomplete, and live linting through an LSP server.
  - Compiler testing techniques: fuzzing, differential testing, LLM-based techniques, etc.

Course Project (Research + Practice)

- **Team size:** 2-3 students (there is no additional bonus for small-sized or single-person teams)

- **Task 2: Building a working compiler for a C-like language**
  - Phase 1: Lexical analysis (2 weeks)
  - Phase 2: Syntax analysis (2 weeks)
  - Phase 3: Semantic analysis (2-3 weeks)
  - Phase 4: LLVM IR generation (3-4 weeks)
  - Phase 5: RISC-V code generation (3-4 weeks)
  - Phase 6: Code optimization (2-3 weeks)

Milestone check 1
Week 6 (tentative)

Milestone check 2
Week 12 (tentative)

Final inspection
Week 17 (tentative)

Tools and Environment

- We will use two sets of tools (a mixture of classical and modern tools)
  - **Flex + Bison**: to understand core concepts of compilation
  - **Antlr + LLVM**: to build your own compiler

- Please set up your environment in week 1.
  - Project 1 (Phase 1) will be released in week 2

What is Flex?

- Flex is a fast lexical analyser generator. It is a tool for generating programs that perform pattern-matching on text. Flex is a non-GNU free implementation of the well known Lex program.
  - GitHub repo: https://github.com/westes/flex
  - Manual: https://www.epaperpress.com/lexandyacc/download/flex.pdf
- Please follow the instructions here to install Flex: https://sqlab-sustech.github.io/CS323-Compilers-2025F-docs/env/.

What is Bison?

- Bison is a general-purpose parser generator that converts an annotated context-free grammar into a parser. You can use it to develop a wide range of language parsers, from simple desk calculators to complex programming languages.
  - Official website: https://www.gnu.org/software/bison/
  - Manual: https://www.gnu.org/software/bison/manual/

- Please follow the instructions here to install Bison: https://sqlab-sustech.github.io/CS323-Compilers-2025F-docs/env/.

What is Antlr?

- **ANTLR** (ANother Tool for Language Recognition) is a powerful **parser generator** (like Flex+Bison) used to build languages, tools, and frameworks. It takes a grammar file—a set of rules that define a language's structure—and automatically generates the source code for a **lexer** and a **parser**. Some key Features:
  - **Grammar-Driven Development:** You define your language using a clear, expressive grammar in a .g4 file. ANTLR handles the complex task of generating the parsing code.
  - **Multi-Language Target:** ANTLR itself is written in Java, but it can generate lexers and parsers for many target languages, including **Java, C#, Python, JavaScript, Go, Swift, and C++**.
  - **Ease of use:** It provides two elegant mechanisms to traverse the parse tree to execute your logic:
    - **Listener:** An event-driven interface that automatically walks the tree.
    - **Visitor:** A pattern where you can explicitly control the traversal, ideal for tasks requiring output (like interpretation or code generation).
- Antlr is open-source: https://github.com/antlr/antlr4. Installation instructions can be found at https://sqlab-sustech.github.io/CS323-Compilers-2025F-docs/labs/w1-intro/.

What is LLVM?

- **LLVM** (formerly an acronym for **Low Level Virtual Machine**) is a collection of modular and reusable compiler and toolchain technologies.
  - It's not a single tool, but a project that provides a modern infrastructure for building compilers, optimizers, and just-in-time (JIT) code generators.
- LLVM's key innovation is its clean separation of the compilation process into three distinct phases:
  - **Frontend:** Translates source code in a specific language (e.g., C, C++, Rust, Swift) into an intermediate representation (IR).
    - Clang is the LLVM-based frontend for the C, C++, and Objective-C languages.
  - **Optimizer (Middle End):** Takes the IR code and performs various analyses and transformations (language-agnostic and machine-independent) to improve it.
  - **Backend:** Translates the optimized IR into machine-specific assembly code for a specific target platform (e.g., x86, ARM, GPU). It handles architecture-specific tasks like instruction selection and register allocation.

Find Your Teammates ☺

- 【腾讯文档】CS323-2025F Project Teams https://docs.qq.com/sheet/DSk9MR3hkaHhFek9o?tab=BB08J2

- Please do this by the end of the second week. Otherwise, you may miss some deadlines.