Final Project

Academic Year 2025-2026 Semester 1

420343 - Compiler Principles

Final Project Task: Compiler-Compiler Construction

1. Requirements

1.1 Mission

- Master the classical algorithms for automatic compiler construction.
- Understanding of compiler construction algorithms and compiler programming methods.
- System development abilities including **requirements analysis, algorithm design, coding, and documentation**.
- Ability to conduct **literature research** and refer to **open-source project resources** for compiler development.
- Team management and collaboration skills, emphasizing **joint participation and task division**:
  - Code must be hosted on **GitHub or Gitee**, with **team cooperation and individual contributions** demonstrated via Contributor activity and code commits etc.
- Abilities in **summary, expression, presentation, communication, and critical thinking**.
- Reasonable use of AI assistants.

1.2 Task

- Select classical algorithms studied in class for automatic construction of scanner(lexical analyzers), parser(syntax analyzers), and intermediate code generation, and implement a **Compiler Generator**.
- The scanner and parser should be automatically generated using formal descriptions as input. The compiler generator should take as input the **lexical rules** (e.g.， regular grammar) and **syntactic rules** (e.g., context-free grammar) of a language, and output a **compiler(scanner & parser)** for that language.
- Automatic generation of intermediate code is not required, but at least one test language's intermediate code generation should be supported.

- The generated compiler should take **source code** as input and produce **correct intermediate code** (e.g., three-address code) as output, with appropriate error reporting for incorrect source code.
- **Syntax-directed translation** must be used to generate intermediate code simultaneously during parsing and save it to a file.
- **Classical Algorithms**
  i. **Lexical Analysis:** Thompson’ s construction, subset construction, equivalence state method, etc.
  ii. **Syntax Analysis:** LL(1) parsing, LR parsing, etc.
  iii. **Intermediate Code Generation:** Attribute grammars, Syntax-directed translation, etc.

1.3 Team work

- **Total Weight:** 30%
- **Team Size:** Up to 4 members

---

2. Submission

- **Submission Materials:**
  a. Source code
  b. Program design documentation (Word/PDF)
  c. Defense PPT
- **Submit via Canvas:**
  - All three of the above materials must be submitted via Canvas as part of exam records; only **one submission per team** is required.
  - Please place the source code into a folders, compress them into one ZIP file, and submit it via *Canvas*.
- **Project must be open-source:**
  - upload code to GitHub/Gitee. Include a **README** introducing the project, providing test case design, and showing test results. Testing should include:
    - **A:** Testing the Compiler-Compiler itself, generating compilers for different language rules, here the compiler can only include scanner & parser.
    - **B:** Testing the generated compiler by compiling source code snippets and demonstrating the generated intermediate codes.
- **Program Design Documentation Requirements:**

- **Suggested Content:** Introduction to design and implementation of the compiler-compiler.
    i. **System Design:** Overview and detailed design.
    ii. **System Implementation:** Detailed description of specific functions, including screenshots of key code.
    iii. **System Testing:** Test case design and results, ensuring completeness and challenge of test cases.
    iv. **AI Assistant Usage:** Description of how AI tools were used.
- **Requirements:** Clear logic, appropriate level of detail, and professional formatting.

- **PPT Requirements：**
    - **Suggested Content:**
        - Overview of system functions
        - Introduction of test cases
        - Team collaboration showcase (GitHub/Gitee)
        - System demonstration

---

3. Presentation

- **Presentation:** Within 10 minutes, all members must attend
- **Q&A:** 5 minutes
- **Order:** By team number, 16th week.

Friendly reminders:

1. Test your computer in the classroom beforehand to ensure PPTs run and programs execute correctly.
2. Upload all materials to Canvas “Final Project” before the prensentation.

---

4. References

- No restrictions on the form of lexical or syntax rules: typically, **lexical rules use regular expressions**, and **syntax rules use Chomsky context-free grammar or Backus-Naur Form (BNF)**.
- Recommended reference formats from popular compiler tools:
    - **ANTLR:** https://www.antlr.org/

- **BISON(YACC):** https://www.gnu.org/software/bison/
- **FLEX(LEX):** https://github.com/westes/flex/blob/master/examples/testxxLexer.l

- The definition of programming languages can be referred to as follows:
  - **PL/0**： https://en.wikipedia.org/wiki/PL/0?utm_source=chatgpt.com
  - **Decaf**: https://anoopsarkar.github.io/compilers-class/decafspec.html

---

5. Grading Criteria:

Grading Criterion	Weight
Implementation of functionalities (correctness, completeness, originality, etc.)	50%
Presentation quality	20%
Documentation quality	10%
Repository publication(GitHub/Gitee)	10%
Team collaboration	10%

Looking forward to your presentation!