COMP 520: Compilers

Syllabus and Administrative Details
Spring 2022

Time/Dates: TTh 2:00 – 3:15 pm (Tue Jan 11 – Thu Apr 28, 2022)
Instructor: Jan Prins, FB 334 (Brooks Building), Email: prins@cs.unc.edu
TA: Tao Tao, Email: ttao@cs.unc.edu , office and office hours TBA.
Format: In-person lectures, office hours (TBA), and Piazza for asynchronous Q&A and discussions

Description. This upper-level undergraduate class (also available for graduate credit) builds on, extends, and integrates material from prerequisite courses to build a compiler for a non-trivial subset for the Java programming language. Java will also be the implementation language for the compiler. Upon completion of this course, you should:

- Understand the theory and practice of compilers, linkers, debuggers, and program execution using hardware or abstract machines.
- Appreciate the effect of trade-offs in programming language design and computer architecture on program compilation and the run-time support system.
- Have gained additional experience with the design and implementation of a large and complex program using Java.
- Be prepared for advanced study in programming languages and optimizing compilers.

Communication. All course materials (reading assignments, lecture notes, problem sets, and project assignments) will be available from our course web page http://www.cs.unc.edu/~prins/Classes/520/. Check this page regularly for updates. For questions and discussion outside of class and office hours use online discussion via Piazza. Please consult the course web page to sign up for piazza.

Prerequisites. Prerequisites are COMP 301 (Foundations of Programming), COMP 311 (Computer Organization) and COMP 455 (Models of Languages and Computation). Familiarity with programming language design and implementation concepts (COMP 524) is helpful but not required.

Text. We will use *Programming Language Processors in Java: Compilers and Interpreters*, by David Watt and Deryck Brown, Prentice Hall, 2000 (ISBN 0-130-25786-9). This book is out of print, but a pdf version is available on our website.

Approach. Class lectures describe the theory and practice of compiling a programming language into a form suitable for execution by computer. Topics covered include compiler organization, with detailed study of the various phases of compilation (concrete syntactic analysis, abstract syntax tree construction, contextual analysis, and code generation). We will also consider run-time organization and various interpretive techniques to support execution of compiled programs. Special consideration will be given to the implementation of object-oriented languages such as Java.

The compiler construction project is a central part of the course. You will construct a compiler for miniJava, a subset of the Java language. Your compiler should generate instructions for a stack-oriented abstract machine for which an interpreter and debugger will be provided. Construction of the miniJava compiler should follow the design strategy described in the course text and in class. A complete compiler available online in source form illustrates the approach for the Triangle language described in the text (however, Triangle differs substantially from miniJava). Specifications for each phase of the project will be distributed and must be implemented in your project. The compiler construction project will occupy a significant amount of time outside of the classroom. Five functional milestones in the compiler project will be used to keep on schedule, and will be graded. Start each phase of the project promptly – late checkpoint submissions will not receive credit (functionality tests for each milestone are made available shortly after the due date).

Grading. The course grade will be based on the miniJava compiler project (48%), a midterm and a final exam (32% total), four short written assignments (16%), and class participation (4%).

Written assignments must be completed individually and submitted electronically by midnight on the due date. Compiler milestones must be uploaded to the appropriate course submission directory by the specified due date and time. *Late assignments will not be accepted, except for medical reasons*.

The midterm exam is scheduled during class time on Tuesday Mar 1. The final exam is scheduled for Friday May 5 at noon in compliance with UNC final exam regulations and according to the UNC Final Exam calendar. You may consult your notes and other course materials in the exams. Communication with anyone other than the instructor is prohibited.

The programming project can be completed individually or by a team of two. A team effort earns 80% project credit per team member, additional credit is available through optional extensions of the compiler. A team effort must be applied for and approved by the instructor in writing (email) before the first project checkpoint due date. Upon approval, the two members are fixed for the duration of the project. If a team encounters irreconcilable problems working together, it may be disbanded following consultation with and approval from the instructor, with the remaining individual projects earning credit at a negotiated rate.

Honor code. The Honor Code and the Campus Code are in effect for this course. I am committed to treating Honor Code violations seriously and urge all students to become familiar with its terms as set out at `http://instrument.unc.edu`. If you have questions, it is your responsibility to ask me about the Code's application. All exams, written work, and programming projects must comply with the requirements of the Honor Code and the rules listed in this section in all aspects of the submitted work.

In this course written assignments and exams must be completed individually and cannot be discussed with other students. For the compiler project, you are encouraged to discuss project design issues with your classmates, the teaching staff, or using our Piazza discussion board, but *all code submitted must be written by yourself or by your team*. In individual discussions and in Piazza, specific code sequences solving a problem should not be posted or exchanged. If you are in doubt about the suitability of a question or posting, feel free to contact me for advice. *Consultation or use of any other compiler produced in any offering of this class is specifically prohibited.*

Computer access. You may use any computer you wish for your compiler development. However, each checkpoint will require you to upload your Java source files to a specific directory on our class server `comp520-1sp22.cs.unc.edu` — it is your responsibility to ensure that your programs work as required using Java SE (Standard Edition) version 8 when executed on the Linux environment found on this server. Additional details related to the server will be clarified in class. You are strongly encouraged to develop your project using the Eclipse IDE - the configuration you need is "Eclipse IDE for Java Developers".

Syllabus. Following are the key topics planned for study, the approximate number of lectures to be spent on each, and the corresponding chapters in the course text.

Introduction (2 lectures) *Chapters 1-3 (selected sections)*
- Compilers and Interpreters, overview of the translation process, motivation
- Specification of programming languages

Syntactic Analysis (8 lectures) *Chapters 4, 3*
- Context-free grammars and parsing
- EBNF grammars and recursive descent parsers
- Lexical analysis and scanner construction
- Bottom-up parsing and precedence parsing
- Components and structure of the translation process
- Abstract syntax trees and AST traversal

Contextual analysis (4 lectures) *Chapter 5*
- Identifier resolution: declarations and references
- Type checking framework
- Contextual analysis for Triangle and Java

Run-time organization (3 lectures) *Chapter 6*
- Storage and execution model
- Procedure activation records and parameter passing
- Object-oriented execution: inheritance and virtual method invocation

Code generation (3 lectures) *Chapter 7*
- Expression evaluation and control flow
- Code templates
- Code generation examples

Virtual Machines and Interpretation (3 lectures) *Chapter 8*
- Virtual machine principles
- Case studies: TAM, mJAM, JVM and .NET MSIL/CLR
- Just-in-time (JIT) compilation

Additional topics (3 lectures) *Supplementary materials*
- Register allocation
- Data flow analysis
- Compiler bootstrapping