**COMP 520: Compilers**

Administrative Details and Syllabus
Spring 2015

**Classroom:** Brooks Computer Science Building FB 007 (level 0).
**Time/Dates:** TTh 3:30 – 4:45 pm (Thu Jan 8 – Thu Apr 23)
**Instructor:** Jan Prins, FB 334, Tel. 919-590-6213, Email: prins@cs.unc.edu

**Description.** This is an upper-level undergraduate course (also open to graduate students) covering several areas of computer program translation and execution, including compilation, interpretation, runtime organization, linking, and loading. Upon completion of the course, you should:

- Understand the theory and practice of compilers, interpreters, assemblers, linkers, and loaders, and be able to work effectively with such programs.
- Be able to design and implement parsers and language translators for simple languages.
- Appreciate the effects of decisions in programming language and computer architecture design on the translation process and run-time system.
- Have experience with the design and implementation of large and complex programs using Java.
- Retain knowledge of compilation techniques in preparation for advanced courses in compilers and/or other advanced computer science studies.

**Communication.** All course materials (reading assignments, lecture notes, problem sets, and project assignments) will be available on our course web page [http://www.cs.unc.edu/~prins/Classes/520/](http://www.cs.unc.edu/~prins/Classes/520/). Check this page regularly for updates. When accessed outside of UNC, some portions of this site may require authentication using your CS login. For questions, we will use Piazza for online discussion. Please use [http://piazza.com/unc/spring2015/comp520/](http://piazza.com/unc/spring2015/comp520/) to join our discussion group in Piazza.

**Prerequisites.** COMP 410 (Data Structures) and COMP 411 (Computer Organization), and COMP 455 (Theory of Computation), and proficiency in Java programming are required. Familiarity with programming language design and implementation concepts (COMP 524) is helpful but not required.


**Approach.** Class lectures will 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 (syntactic analysis, contextual analysis, 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 an integral part of the course. You will construct a compiler for miniJava, a subset of the Java language. The compiler will output instructions for a stack-oriented abstract machine that will be provided. The compiler should follow the compiler design strategy described in the course text and in class. A complete compiler available online in source form illustrates the approach to compile the Triangle language (which differs substantially from miniJava). Certain parts and specifications in our project will be distributed and must be used 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 will be made available shortly after the due date).

**Grading.** The course grade will be based on the programming project (50%), a midterm and a final exam (34%), written assignments (13%), and participation in class and online discussions (3%).

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

The midterm will be given the week before spring break. The course final will be held given Tuesday April 28 at 4PM in compliance with UNC final exam regulations and according to the UNC Final Exam calendar. The midterm and final have equal weight and will be open book and notes.

The programming project can be completed individually or by a team of two. A team effort earns 80% project credit per individual, but this may be made up by undertaking optional extensions of the compiler. If a team effort is selected, 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.

You are encouraged to discuss project design issues with your classmates and the teaching staff on our Piazza discussion board, but all code submitted must be written by yourself or by your team. Problem sets are to be completed individually but questions can be directed to the discussion board. In addition you can contact the instructor individually whenever you are in doubt about the appropriateness of discussing materials with your classmates.

**Honor code.** 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 and the teaching staff on 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 exchanged. If in doubt about the suitability of a question or posting, feel free to contact me individually for advice. *Soliciting help outside of Piazza and permitted discussions described above is specifically prohibited.*

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](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 be submitted with a statement that you complied with the requirements of the Honor Code and the rules listed in this section in all aspects of the submitted work.
**Computer access.** While you may use any computer you wish for your program development, each checkpoint requires you to upload Java source files (and documents in some cases) to a designated directory in the Computer Science department’s AFS file system. It is your responsibility to ensure that your programs work as required using Java SE (Standard Edition) version 7 when executed on the Linux environment found on department computer server classroom.cs.unc.edu. You are encouraged to develop your project using the Eclipse IDE, using version “Juno” or later (current is “Kepler”). The configuration you need is “Eclipse IDE for Java Developers”.

Access to the AFS file system requires a departmental login. Students from other departments registered in this class can pick up their CS login from the Technical Support Center (TSC) in room SN 112 in the first week of class. All CS Linux machines can be accessed from remote locations (without need for VPN) using ssh, scp, or sftp. For windows, the application secureCRT (free to UNC students and staff) is recommended for remote login and file transfer. If you wish to have physical access to the CS building on evenings and weekends, you need to pick up a cardkey from the TSC office.

**Syllabus.** The course syllabus is on the next page.
Syllabus. Here 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. Changes to the syllabus, including project due dates and test dates, are possible. These changes will be announced as early as possible.

Introduction (3 lectures) Chapters 1-3 (selected sections)
- Compilers and Interpreters, overview of the translation process, motivation
- Specification of programming languages
- Components and structure of the translation process
- Reminder of key Java concepts

Syntactic Analysis (5 lectures) Chapter 4
- Context-free grammars and parsing
- Top-down and bottom-up parsing
- Recursive descent parsers
- Lexical analysis
- Abstract syntax trees (AST)

Semantic analysis (4 lectures) Chapter 5
- AST traversal
- Identifier resolution: declarations and references
- Type checking

Run-time organization (4 lectures) Chapter 6
- Storage and execution model
- Data representation and access
- Storage allocation and management
- Procedure activation records and parameter passing
- Object-oriented execution: inheritance and virtual method invocation

Code generation (3 lectures) Chapter 7
- Stack and register-based code generation
- Address generation
- Expression evaluation
- Procedure invocation and parameter passing

Virtual Machines and Interpretation (4 lectures) Chapter 8
- Virtual machine principles
- Interpreter organization
- Case studies: TAM, JVM and .NET MSIL/CLR

Additional topics (4 lectures) Supplementary materials
- Linkers, loaders and debuggers
- Just-in-time (JIT) compilation
- Code optimization: data flow analysis
- Compiler bootstrapping