Contents

1 Course Information

1.1 Website

https://www.cs.umb.edu/~siyer/teaching/cs451/

1.2 Catalog Description

Introduction to compiler organization and implementation, including formal specifications and algorithms for lexical and syntactic analysis, internal representation of the source program, semantic analysis, run-time environment issues and code generation. Students will write a compiler for a reasonably large subset of a contemporary language, targeted to a virtual machine.

Prerequisites: CS310 and CS420 or CS622; or permission of the instructor.

Students who successfully complete this course will be able to: write parsers and produce an abstract syntax tree (AST); analyze and generate code for a programming construct represented by an AST; and allocate physical registers (a limited resource) to a program expressed in terms of virtual registers (an unlimited resource).

1.3 Staff

Swami Iyer will be the primary instructor for the course. He will be assisted by a graduate teaching assistant (TA).

1.4 Lecture

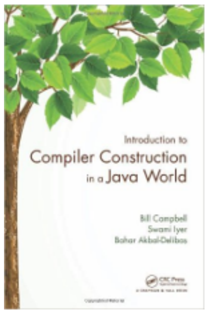
There will be two lectures per week. In each lecture, the instructor will present the material for that lecture. Roughly once a week, the instructor will also conduct an online quiz on recently covered material.

1.5 Study Material

For each topic covered, you will have access to lecture slides (in HTML/PDF formats) and exercises with solutions. The materials are based on the following optional texts:

Introduction to Compiler Construction in a Java World by Bill Campbell, Swami Iyer, and Bahar Akbal-Delibaş

This text enables a deep understanding of the Java programming language and its implementation. It covers all of the standard compiler topics, including lexical analysis, parsing, abstract syntax trees, semantic analysis, code generation, and register allocation.



1.6 Grading Scheme

1.6.1 Assessments

Item	% of Final Grade
Programming Assignments (1, 2, 3, 5, and best of 4 and 6)	35
Exams (2)	60
Participation	5

- The goal of the programming assignments is to make sure that you can apply the concepts learned in class to enhance the functionality of the base *j--* and *iota* compilers.
- The closed-book exams will test your understanding of the theoretical concepts covered in class.

- Your participation score will be based on weekly in-class quizzes. Only your top 10 quiz scores will count.
- If you score at least 83% (B) on each exam, the highest score will be considered as your exam average. For example, if your exam scores are 88, 95, and 83, your average exam score will be 95.
- You will receive 0.01x% extra points if x% of the class completes the end-of-semester course evaluation.
- Your overall percentage score will be rounded up.

1.6.2 % Score to Letter Grade

`[93, 100]: A, [90, 93): A-, [87, 90): B+, [83, 87): B, [80, 83): B-, [77, 80): C+, [73, 77): C, [70, 73): C-, [67, 70): D+, [63, 67): D, [60, 63): D-, [0, 60): F`

1.7 Software Needed

1.7.1 Piazza

We will use Piazza as the Q&A platform for the course. If you have any general questions about the assignments, exams, or the lecture material, the most effective way to get them answered is by posting them on Piazza. You can expect your questions to be answered by the course staff or your peers.

1.7.2 Gradescope

We will use Gradescope to grade the in-class quizzes, programming assignments, and exams.

1.7.3 Programming Environment

To write and execute Java programs in this course, you will need access to a desktop/laptop computer (Linux, Mac, or Windows) that is properly configured with the necessary software. Click here for setup instructions.

1.7.4 Zoom

We will use Zoom for all remote meetings.

1.8 Linux Lab

The desktop computers in the Linux Lab (`M-3-0731`) are set up with the programming environment for the course. To use these computers, you must have a CS Account, which you can obtain by visiting the Computer Science Portal. If you are not a member already, you need to first register for a portal account. Once you have the portal account, sign into the portal and select your courses for the semester. You will be notified via your UMB email once your account is created/activated for the semester.

Lab Hours: `Mon - Thu 9:00 AM - 9:00 PM` and `Fri 9:00 AM - 5:00 PM`. To access the lab outside of those times, you must call Public Safety.

Note: The computers in the Linux Lab do not support roaming profiles, which means your data on each computer is local to that computer.

1.9 Policies

1.9.1 Classroom

Come to lecture on time and stay for the entire session. If you have to leave early, let the instructor know in advance. Have your mobile phone silenced or turned off during the entire session. Use of earphones/headphones during the session is not permitted. Laptops may be used during the session, but only for class purposes. Do not talk to each other during the session. If you have any questions, bring them up to the instructor.

1.9.2 Piazza

If you have a question, first make sure that it has not already been asked/answered. Clearer questions get better answers, so re-read your question before you post it. Ask your questions early. Posts are categorized using tags, so pick an appropriate tag for your post. Use the platform only for questions that can be asked in a general way, without sharing code or other assignment-related work. However, if you are stuck on a problem despite your valiant efforts to solve it, you may seek help from the course staff by posting your code privately, as properly formatted text (not images). Any post that is inappropriate or violates the academic honesty code will be deleted by the course staff.

1.9.3 Makeup Exam

You must provide appropriate documentation if you were/are unable to take an exam on the scheduled date and want to arrange a makeup. The documentation must be a letter from the Dean of Students ↗ if the type of your absence is among those listed on their website. For other types of absences, the supporting documentation must be emailed to the instructor directly.

Note: There will be no makeup of missed quizzes.

1.9.4 Assignment Deadline

Assignment deadlines are firm — late submissions will not be accepted. The only exception to this policy is if you have been granted extended time on assignments through the Ross Center (see the section on accommodations below), in which case you are allowed a 24-hour extension per assignment. To avail this extension, you must email me at least 48 hours prior to the assignment deadline.

1.9.5 Regrade Request

If you have any concerns about the grading of a particular quiz/assignment/exam, you may submit a regrade request via Gradescope. You must submit the request within a week from the date the quiz/assignment/exam grades are published, or else your request will be turned down.

1.9.6 Collaboration

Click here ↗ for the collaboration policy and penalties for infractions of the policy.

1.9.7 Accommodations for Students with Disabilities

Section 504 of the Americans with Disabilities Act of 1990 offers guidelines for curriculum modifications and adaptations for students with documented disabilities. If applicable, students may obtain adaptation recommendations from the Ross Center for Disability Services ↗. The student must present these recommendations and discuss them with the instructor within a reasonable period, preferably by the end of Add/Drop period.

1.9.8 Campus Closure

In the event of a campus closure, all activities will be conducted remotely, via Zoom. If there is an exam scheduled to take place on that day, the exam will be postponed to the next suitable date.

2 Topics Covered

- Course Mechanics *[Lecture 1]*
- Programming Environment *[Lecture 1]*
- Chapter 1: Compilation
  - Preliminaries *[Lecture 2]*
  - Overview of the *j*-- Compiler *[Lecture 3 and 4]*

- Chapter 2: Scanning
  - Preliminaries *[Lecture 5]*
  - Handcrafting a Scanner *[Lecture 6]*
  - Generating a Scanner *[Lecture 7]*
  - JavaCC Scanner for *j--* *[Lecture 8]*
- Chapter 3: Parsing
  - Preliminaries *[Lecture 9]*
  - Top-down Recursive Descent Parsing *[Lecture 10]*
  - Top-down LL(1) Parsing *[Lecture 11]*
  - Bottom-up LR(1) Parsing *[Lecture 12]*
  - JavaCC Parser for *j--* *[Lecture 13]*
- Chapter 4: Type Checking
  - Preliminaries *[Lecture 14]*
  - Pre-analysis of *j--* Programs *[Lecture 15]*
  - Analysis of *j--* Programs *[Lecture 16 and 17]*
- Chapter 5: Codegen
  - Preliminaries *[Lecture 18]*
  - Codegen for *j--* Classes and their Members *[Lecture 19]*
  - Codegen for *j--* Control, Message, Field Selection, and Array Access Expressions *[Lecture 20]*
  - Codegen for *j--* Assignment, String Concatenation, Cast, and Other Operations *[Lecture 21]*
- Chapter 6: The *iota* Compiler
  - Preliminaries *[Lecture 22]*
  - High-level Intermediate Representation (HIR) *[Lecture 23]*
  - Low-level Intermediate Representation (LIR) *[Lecture 24]*
  - Register Allocation *[Lecture 25 and 26]*

3 Assignments

3.1 The List

There are 6 programming assignments in all. These are due at midnight (11:59 PM to be precise) on the dates indicated on the Calendar page of the course website.

#	Title	Goal
1	Supporting Simple Operations	Become familiar with *j--* and the Java Virtual Machine (JVM); and extend the *j--* language by adding support for some arithmetic operators, conditional expression, and do statement.
2	Scanning	Modify the handcrafted scanner to support multiline comments; additional tokens (reserved words and operators); and `long` and `double` literals in *j--*.
3	Parsing	Modify the handcrafted parser to support `long` and `double` basic types; additional operators; and for, break, continue, and switch statements in *j--*.
4	Scanning and Parsing with JavaCC	Modify the `j--.jj` file used for generating a scanner and parser (using JavaCC) for *j--* to support multiline comments; `long` and `double` basic types; additional tokens (reserved words and operators); conditional expression; and do, for, break, continue, and switch statements.
5	Type Checking and Code Generation	Implement type checking and JVM code generation for the programming constructs that were added to *j--* in Assignment 3 (Parsing).
6	Register Allocation	Implement the graph coloring based register allocation algorithm in *iota*.

3.2 Submitting Your Work

You will use Gradescope to submit your Java programs (ie, `.java` files) and the `notes.txt` file. Make sure that you only submit files listed under the Files to Submit section of the assignment writeup.

You may submit your files as many times as you like up until the assignment deadline. The most recent submission is considered active by default and your score on the active submission is your official score for the assignment as well. You have the option of making any of your previous submissions active before the deadline.

Note: It is your responsibility to make sure that your active submission contains the correct versions of the programs and the notes file, ie, the ones that you worked on and intended to submit. We will not entertain any requests to switch/update an active submission.

3.3 How the Assignments will be Scored

3.3.1 Correctness

Your solution for each exercise/problem will be evaluated for correctness by an autograder. Each test that is used for this purpose is worth some number of points; your solution will receive all the points from a test that passes and 0 points from a test that does not.

3.3.2 Code Quality

Your solution for each problem will additionally be checked by the TA for code quality. The TA will apply one or more of the following rubrics (#1 carries no penalty, but the rest do) to your solution:

1. **All good:** your solution passes all the autograder tests, is well formatted/organized, uses meaningful variable names following proper naming conventions, includes useful comments, meets specific efficiency requirements (if any), and only uses concepts that are allowed by the assignment/course.
2. **Fails some autograder tests:** your solution does not pass all the autograder tests.
3. **Poorly formatted/organized:** your solution is not properly formatted/organized, ie, is messy/unreadable.
4. **Poor/unconventional variable names:** your solution does not use meaningful variable names (ie, names relevant to the problem) or does not follow proper variable naming conventions.
5. **Lacks useful comments:** your solution does not include any comments or is commented just for the sake of commenting (ie, they say the obvious and are not useful to someone reading your code).

6. **Does not meet efficiency requirements, if any:** your solution does not satisfy specific efficiency (ie, space and time complexity) requirements, if any.
7. **Uses concepts that are not covered/allowed:** your solution uses concepts that are not allowed by the assignment/course.
8. **No working code submitted:** you did not submit any solution or your solution did not pass any of the autograder tests.

3.3.3 Notes File

The given `notes.txt` file for an assignment must be uploaded with the three sections (#1 mandatory, #2 if applicable, and #3 optional) filled in as appropriate. In section #1, for each problem, you must describe your approach along with any issues you encountered and if/how you managed to solve those issues. The TA will grade your notes for each problem by applying exactly one of the following rubrics (#1 carries no penalty, but the rest do):

1. **Excellent:** solid, high-level (ie, with minimal jargon) description that demonstrates a solid understanding of the problem and the implemented solution.
2. **Good:** bit too technical (ie, reads more like code than prose) but otherwise demonstrates a clear grasp of the problem.
3. **Average:** lacks important details suggesting gaps in the comprehension of the problem.
4. **Poor:** too short or vague suggesting very little to no understanding of the problem.
5. **Very poor:** empty or simply a rehash of the directions provided in the discussion document.