# CSE 304 : Compiler Design

## *Syllabus*

***Term: Fall 2021***
***Instructor:*** *Tony Mione*
***Course Meeting Times:*** *Tue & Thur, 10:30 – 11:50 AM*
***Office:*** *B425*
***Phone:*** *+82 032-626-1226*
***Email:*** *antonino.mione@sunykorea.ac.kr*
***Office Hours: Mon/Wed: 11:00-12:00 & 1:00-2:00PM, Tue/Thu: 1:00-2:00 PM***
***Course Homepage:*** *www3.cs.stonybrook.edu/~amione/CSE304_Course/index.html*
***Text:*** Aho, Alfred V., Lam, Monica S., Sethi, Ravi, Ullman, Jeffrey *Compiler Principles, Techniques, and Tools*, Publ, ISBN 978-0321486813
***Recommended Reference:*** Levine, John, Brown, Doug, and Mason, Tony, lex & yacc, O'Reilly, ISBN *978-1565920002*

## Course Overview

Studying compilers will give the student an in depth knowledge of programming languages, including text analysis, grammar and semantics, as well as the theories behind them. It also teaches a wealth of knowledge about programming language internals. This helps the student to write more efficient high level language code.
Specifically, we will discuss how to write grammars for languages, how to parse and translate code in the language, and the theories behind these tasks. Over the course of the semester, we will implement a compiler for a small language that will generate MIPS assembler code. Through the implementation, we will learn how programming language elements are implemented, how to set up a runtime environment, and an assembly language.
There will be lectures in class but some components of flipped learning are incorporated including text book readings and a few supplemental videos.

## Course Objectives/Outcomes

Upon completion of the courses, students are expected to:

- An ability to use of formal attributed grammars for specifying the syntax and semantics of programming languages.
- Working knowledge of the major phases of compilation, particularly lexical analysis, parsing, semantic analysis, and code generation.
- An ability to design and implement a significant portion of a compiler for a language chosen by the instructor.

## Major Topics Covered in the Course:

- Learn overall compiling steps using important tools
- Lexical analysis (Regular expressions, NFA, DFA)
- Syntax analysis (Context-free grammars, Top-Down parsing, Bottom-Up parsing)
- Semantic analysis (Syntax directed translation, Type checking)
- Runtime environment (Memory allocation)
- Code generation (x86 assembly language, runtime environment, register allocation and assignment)

## Prerequisite

- C or higher: CSE216 or CSE 219 or CSE 260; CSE 220
- Advisory Prerequisites: CSE 303 or CSE 350

## Grades and Evaluation

The course provides a total of 500 points distributed across the below categories.
Your grade in the course will be based on the following work:

**Assignments –** *25% (125 points)* - A number of assignments will be given from textbook problems to help the student understand the theoretical concepts in the text.
**Project** – *30% (150 points)* – A large project will be given in several (4-5) segments that will involve implementing part of a compiler for a small language. Combined, these form the term project that should be functional by the end of the semester.
**Class Attendance/Participation –** 5% (25 points) – ***missing more than 20% of the classes will result in a grade of F***
**Midterm Exam 1 –** *10% (50 points)* - A midterm exam based on reading and concepts presented in the lecture.
**Midterm Exam 2 –** *10% (50 points)* - A midterm exam based on reading and concepts presented in the lecture.
**Final Exam** – 20*% (100 points)* - A cumulative final exam will provide questions that will cover the key concepts taught through the entire semester.

### Final Grade Calculation

The final grade is based on the accumulated points from all quizzes, exams, and assignments (with the entire class comprised of 500 points). Letter grades are given on the following scale:

| **Letter** | **Minimum Percentage** | **Minimum 'points'** |
|---|---|---|
| A | 93 | 465 |
| A- | 90 | 450 |
| B+ | 87 | 435 |
| B | 83 | 415 |
| B- | 80 | 400 |
| C+ | 77 | 385 |
| C | 73 | 365 |
| C- | 70 | 350 |
| D+ | 67 | 335 |
| D | 60 | 300 |
| F | <60 | <300 |

## Attendance

The range of topics covered in this course is extensive, and due to the limited lecture
and lab time, these topics are covered in an intensive manner. Therefore, attendance
at both lectures and lab are mandatory in order to keep up and perform well.

- Attendance will be taken in the beginning of each lecture and lab session.
- If a student has over 20% unexcused absences, the final course grade will be an F.

## Re-grading

For re-grading of an assignment or exam, please meet with the person (instructor or teaching assistant) responsible for the grading. All such requests that are later than one week from the date the graded work is returned to the class will not be entertained. To promote consistency of grading, questions and concerns about grading should be addressed *first to the TA* (if there is a TA for the course) and then, if that does not resolve the issue, to the instructor. You are welcome to contact the TA by email or come to his/her office hour. If you would like to speak with the TA in person, and have a schedule conflict with his/her office hour, you are welcome to make an appointment to meet the TA at another time.

## Programming Assignments

### Extensions

Programming assignments must be turned in on the day they are due. Students are urged to plan ahead to avoid problems such as congestion or failure of computer facilities at the last minute. If your assignment is incomplete or is not working by the due date, turn in whatever you have. Note due to limited resources for grading, programs which do not compile or run for testing may not be graded. If some sort of emergency prevents you from submitting your assignment on time, supplying me with suitable documentation and notification prior to the assignment deadline will be considered. A penalty may be applied.

## Course Schedule

Following is a tentative schedule for the class topics:

| **Week/Day** | **Lecture Topics** | **Readings** | **Tests/Vids** |
|---|---|---|---|
| W1: 8/31 | Course Overview | | |
| 9/2 | Compiler Design Overview | Aho: Chapter 1 | |
| W2: 9/7 | Simple compiler: Syntax Definitions, Syntax-directed Translation | Aho: Chapter 1 | |
| 9/9 | Lexical analysis: Regular Expressions, Transition diagrams, NFA, DFA, Conversion from NFA to DFA | Aho: Chapter 3 (3.1-3.8) | |
| W3: 9/14 | Lex & Yacc | Aho: Chapter 2 | |
| 9/16 | Lex & Yacc: Translation to abstract stack machine | | |
| W4: 9/21 | Chuseok – No classes | | |
| 9/23 | Introduction to Symbol Management | PLP: c26-c32 | |
| W5: 9/28 | Runtime Storage Management | | |
| 9/30 | Implementing a Symbol Table | | |
| W6: 10/5 | Midterm I Review | | |
| 10/7 | Midterm I | | Midterm I |
| W7: 10/12 | Syntax analysis: Top - Down Parsing (Nonrecursive Predictive Parsing) | Aho: Chapter 4 (4.1-4.4) | |
| 10/14 | Syntax analysis: Bottom-Up Parsing (SLR Parser) | Aho: Chapter 4 (4.5-4.6) | |
| W8: 10/19 | Syntax analysis: Bottom-Up Parsing (LR Parser, LALR Parser) | Aho: Chapter 4 (4.7-4.8) | |
| 10/21 | Syntax directed translation: Overview, S-attributed definitions | Aho: Chapter 5 (5.1-5.3) | |
| W9: 10/26 | Top-Down translation of L-attributed definitions | Aho: Chapter 5 (5.4-5.5) | |
| 10/28 | Bottom-Up translation of Inherited attributes | | |
| W10: 11/2 | Type Checking | Aho: Chapter 6 (6.3-6.5) | |
| 11/4 | Runtime environments | Aho: Chapter 7 (7.1-7.3) | |
| W11: 11/9 | Midterm II Review | | |
| 11/11 | Midterm II | | Midterm II |
| W12: 11/16 | Assembly language | | |
| 11/18 | Code generation (Lex & Yacc, implementation) | | |
| W13: 11/23 | Intermediate code generation (Three address code, Part 1) | Aho: Chapter 6 (6.1-6.2) | |
| 11/25 | Intermediate code generation (Three address code, Part 2) | Aho: Chapter 6 (6.6-6.8) | |
| W14: 11/30 | Code Generation (Part I) | Aho: Chapter 8 | |
| 12/2 | Code Generation (Part II) | | |
| W15: 11/30 | Code Optimization | | |
| 12/2 | Open Topics | | |
| W16: 12/7 | Review for Final | | |
| 12/9 | Adjustment Dqy : Monday classes | | |

## Academic Dishonesty

You may *discuss* the practice problems with anyone you like, however each students' *assignment (including coding)* which they submit must be **their own work, and only their own work. Any evidence that source code or solutions have been copied, shared, or transmitted *in any way* (this includes using source code downloaded from the Internet or written by others in previous semesters!) will be regarded as evidence of academic dishonesty.**

### Guidelines for Assignments

Working together to find a good approach for solving a programming problem is cooperation; listening while someone dictates a solution is cheating. You must limit collaboration to a *high-level discussion of solution strategies*, and stop short of actually writing down a group answer. Anything that you hand in, whether it is a written problem or a computer program, must be written in your own words. If you base your solution on any other written solution, ***you are cheating***

### Guidelines for Taking Exams

When taking an exam, you must work completely independently of everyone else. Any collaboration here, of course, is cheating. All examinations will be closed-notes and closed-book. No electronic devices of any kind will be permitted to be used during exams. All cell phones must be silenced or turned off during exams. You will be allowed one sheet of notes, both sides (8.5 x 11 or A4).

### General Guidelines

**Be advised that any evidence of academic dishonesty will be treated with utmost seriousness. *We do not distinguish between cheaters who copy others' work and cheaters who allow their work to be copied.***

If you cheat, you will be given an F on the assignment. Any incidence of cheating will be reported to Academic Affairs. If you have any questions about what constitutes cheating, please ask.

## Students with Disabilities

If you have a physical, psychological, medical or learning disability that may impact your course work, please let the instructor know. Reasonable accommodation will be provided if necessary and appropriate. All information and documentation are confidential.

## Critical Incident Management

The University expects students to respect the rights, privileges, and property of other people. Faculty are required to report to the Office of Judicial Affairs any disruptive behavior that interrupts their ability to teach, compromises the safety of the learning environment, or inhibits students' ability to learn.