

## CMSC 430 (Section 101)

### Theory of Language Translation (Introduction to Compilers)

#### Staff

Name	Office	Office Hours	email
Chau-Wen Tseng	AVW 4135	Tue, Thu 2–3pm	tseng@cs.umd.edu
Nan Wang	AVW 1151	Wed 1–3pm	nwang@cs.umd.edu

#### Course Description

Introduction to compiler construction (emphasis on compiler front ends). Course contents include the following: Formal translation of programming languages, program syntax and semantics. Finite state recognizers and regular grammars. Context-free parsing techniques such as LL(k) and LR(k). Code generation, improvement, syntax-directed translation schema.

#### Prerequisites

CMSC 330 (Organization of Programming Languages).

#### Texts

- Andrew Appel, *Modern Compiler Implementation*, Cambridge University Press.

#### Computer Accounts

Class accounts are on the OIT Unix cluster machines; they are to be used only for programming projects. All files are subject to inspection by the instructors and assistants. Projects are to be submitted electronically using `submit`. The `submit` program only records the last version submitted before the deadline, so feel free to submit early and often.

#### Class Web Page

The class web page is at: <http://www.cs.umd.edu/class/spring2001/cmssc430/>

#### Academic Honesty

All graded materials (examinations and programming assignments) must be strictly individual efforts. Cooperation on programming assignments is limited to general discussion of the problem (not its solution), and assistance with errors. Transmitting a copy of a solution (in either hardcopy or electronic form), or falsely representing the correctness of a program are considered forms of academic dishonesty. Students are expected to read and adhere to the guidelines presented in the Computer Science Department newsletter which is distributed along with this syllabus.

## Grading

- Examinations. Midterm (20%), Final (30%). There will be no makeup examinations except for medical emergencies.
- Programming projects. 50%.
  1. convert regular expressions to minimal DFAs.
  2. scanner/parser using `JLex` and `CUP`.
  3. simple type checker.
  4. Java byte code generation.
  5. advanced code generation & optimization.

Project grades consist of 50% for correct execution on test data made available to students at time of the assignment, 45% for correct execution on the instructor's test data, and 5% for documentation, program structure, and efficiency. There will be no partial credit for partial correct execution. Projects 3–5 rely on previously completed projects.

There is a late policy as follows. There is a 20% penalty for projects submitted past the deadline, with an additional 10% penalty for each day past the deadline. No credit will be given for projects one week past the due date.

## Topics

- Prerequisite material. Run-time organization of programming languages, storage allocation, addressing nonlocal variables and parameters, array references.
- Overview, 1 week: organization and function of compilers.
- Regular Languages, 1 week: regular languages and regular expressions, NFAs and DFAs, minimizing the states of a DFA.
- Context-Free Languages, 3 weeks: context-free grammars: ambiguity, precedence, associativity, bottom-up parsing: SLR(k), LALR(k) and LR(k) grammars, grammar transformations: left factoring and removal of left recursion, top-down parsing: recursive descent and LL(1).
- Syntax-Directed Translation, 2 weeks: type checking, attribute grammars.
- Intermediate Code Generation, 3 weeks: boolean expressions with explicit values and short-circuit evaluation, control flow in conditional statements, backpatching.
- Code Generation, 2 weeks: basic blocks and flow graphs, DAGs, generating code from DAGs.
- Code Optimization, 2 weeks: global data flow analysis of structured programs, iterative solutions to data flow equations.