Compilers
(Comp Sci 431, Section 001)

Instructor: Erik Krohn
E-mail: krohne@uwosh.edu
Text Message Only: 319-214-2567
Class Information: Monday, Tuesday, Thursday: 202 Halsey, 11:30am – 12:30pm
Wednesday: 101C Halsey, 11:30am – 12:30pm
We may have lectures on Friday. Check d2l often.
Office Location: 216 Halsey
Office Hours: Monday, Wednesday, Thursday: 1:30pm – 3:00pm
Prerequisites: CompSci-331 and Math-212 each with a grade of C or better.
Course Website: http://www.uwosh.edu/d2l/

Course Information
An introduction to compiler writing techniques for translating a higher level programming source language into a lower level target language. Topics to be covered include: definition of programming languages, lexical and syntactic analysis, low level code generation and optimization, run time systems, and error detection, reporting, and recovery. A major programming project will be assigned to provide experience with compiler design.

Course Website
You should check d2l on a regular basis - it will contain lecture notes, handouts, assignments, announcements, and grades. I'll do my best to let you know when something new and important comes up, but it is your responsibility to check the web site frequently for information that you might not get otherwise.

Grading
Course grades will be based on 5-6 programming projects, a few labs, quizzes and three exams. Your final grade will be computed as follows:

40% - Programming Projects
20% - Quizzes/Labs
40% - Exams

Grading will be on a plus/minus system. Grading *may* be done on a curve depending on the overall performance of the class. If no curve is used, your grade will be computed based on the following:

Percentage	Grade	Percentage	Grade
>91	A	71 - 77	C
89 - 91	A-	69 - 71	C-
87 - 89	B+	67 - 69	D+
81 - 87	B	61 - 67	D
79 - 81	B-	55 - 61	D-
77 - 79	C+	<55	F

Exams
Exam material will come from the lecture notes, labs, and programming assignments. There will be more information about each exam as it approaches. The tentative exam dates are listed below. All exams will be taken during the regular class period. These *may change*, so as the date approaches make sure you've got the most recent information.

Midterm 1: Thursday, October 4th, 2012
Midterm 2: Thursday, November 8th, 2012
Final: Wednesday, December 12th, 2012

If you are unable to take a scheduled exam, it may be possible to take a make-up exam provided that you do both of the following, which are then subject to my approval:

1. Make arrangements prior to the scheduled exam (for last minute emergencies, telephone me at 424-7080 or leave a message at the computer science office, 424-2068 or send me a text at the number on the first page). No after-the-fact notifications will be accepted.
2. Have a written medical excuse signed by the attending physician OR have a note of justification from the Dean of Students Office.

If allowed, only one make-up exam will be given. It will be a comprehensive exam given at an arranged time during the last week of the semester.

Quizzes
You will have quizzes throughout the semester. Quizzes are generally short and should only take a few minutes to complete. You will be given a quiz every 3-5 class periods to ensure you are staying current with the material. There are no make-up quizzes.

Projects & Labs
Most projects and labs will consist of short programming projects. One of your goals (during this class and beyond, in Java or *any* programming language) should be to write understandable, readable code. You should be making every effort to comment anything that might be confusing to a reader unfamiliar with your program, to name variables intelligently, to use indentation that reflects the code's organization, and so on. All of this will be taken into account during grading: poorly organized or written code may have a negative impact on your grade, even if the resulting program works fine.

One of the goals of this class is to teach you to write functioning programs in and Java - thus, your code *must* compile and run correctly in order for you to receive full credit. **Code that does not compile will receive at most 50% credit, and often substantially less.** Keep this in mind when writing programs: write your code in small pieces, making sure each piece works before moving on to the next one. It is much better to turn in a project that is not finished but has many working pieces than to turn in one that doesn't work at all, even though most of the code is written.

Each project *must* include a README text file with any information I need to know regarding this project. The information in this file will be taken into account during grading, so it will be beneficial for you to make sure that everything I need to know about your work is written in this file.

All assignments must be submitted electronically via d2l. It is your responsibility to ensure that your submission was submitted correctly. You must double check to ensure your program was uploaded correctly. Each assignment must be submitted by 11:00 PM on the night of the due date. **Late assignments are penalized at 10% per day up to 7 days.** If you believe an assignment or exam was graded incorrectly or unfairly and would like to have it re-graded, please let me know about it *within one week* of having the assignment or exam graded. I will re-grade the entire assignment and you may gain or lose points.

Academic Dishonesty

Academic dishonesty of any kind will not be tolerated. All assignments, labs, quizzes and exams are to be completed individually. While discussion of ideas and problems with fellow students is encouraged, all projects must be done individually. In certain circumstances, code fragments from the instructor may be provided to eliminate tedious coding or to provide a common framework for all students. All other code must be original. Online resources may be used to help you understand the material, but you may not copy online code, as is "borrowing" code from other students, past or present.

Any suspected academic dishonesty will be dealt with on a case-by-case basis. Any clarification of what does or does not constitute academic dishonesty must take place *before* you turn in questionable work. For clarification on what constitutes academic dishonesty, contact me or consult the printed policy in the UWO Student Discipline Code 2007, Chapter UWS 14.

Course Outcomes

- Describe the compilation process, including the roles of the scanner (lexical analyzer). parser, semantic analyzer, intermediate code generator, optimizer, and code generator
- Define a deterministic finite state automaton (DFA)
- Explain how a DFA can be used to recognize a token
- Discuss the relationship between DFA's and regular expressions
- Define the lexical structure of a programming language using regular expressions
- Implement a lexical analyzer using a scanner generator
- Explain how a context-free grammar can be used to express the syntactical structure of a programming language
- Compare the respective capabilities of regular expressions and context-free grammars
- Construct a parse tree for an expression defined by a context-free grammar
- Discuss the differences between LL, LR, and LALR grammars
- Trace the actions of a recursive descent parser in processing an expression defined by an LL grammar
- Trace the actions of a top-down table-driven predictive parser defined by an LL grammar
- Trace the actions of a bottom-up table-driven parser defined by an LR grammar
- Define the syntactical structure of a programming language using a context-free grammar
- Implement a parser using a parser generator
- Explain how syntax-directed translation is used to augment a context-free grammar with translation rules
- Define an abstract syntax tree
- Compare abstract syntax trees to parse trees
- Apply syntax-directed translation to be able to generate abstract syntax trees
- Explain the role of semantic analysis and its relationship to type checking and scoping issues in programming languages
- Discuss the role played by symbol tables in semantic analysis
- Compare different implementation strategies for symbol tables
- Implement a semantic analyzer for a programming language
- Explain what an activation record (stack frame) is and discuss its role in function calls and parameter passing
- Explain the role played by intermediate code generation and compare a compilation process that uses it to one that does not
- Implement a code generator for a programming language
- Test each of the implementations above with an appropriately designed set of test cases
- Collaborate with team members, apply sound software engineering principles, design patterns, and robust coding techniques, to successfully complete a large, software project