CSCI 365

Programming Language Concepts

Spring 2013 MFW 2:00–3:05 am SCI 131

http://cs.wheaton.edu/~tvandrun/cs365

Thomas VanDrunen ☎630-752-5692 630-639-2255 ✉Thomas.VanDrunen@wheaton.edu
Office: SCI 163 Office hours: MWF 8:45–10:15 am; Th 8:45–11:15 am.

Contents

CATALOG DESCRIPTION. Formal definition of programming languages including syntax and semantics; recursive descent parsing, data structures, control constructs, recursion, binding times, expression evaluation, compiler implementation; symbol tables, stacks, dynamic allocation, compiler compilers.

EXPLANATION. The catalog description is an outdated, vague list of terms that could be relevant in a programming languages course. It doesn't define the course very well; we should bring it up to date sometime.

I think of "programming languages" courses under three varieties: Courses on *comparative programming languages* where students learn a smidgen of many languages, comparing and contrasting how various features are realized and the style of programming that each encourages; courses on *compiler construction* where students learn the pragmatics of how to implement a programming language; and courses on *programming language design* where students study the theory of advanced features and their implications for implementation.

This course is **not a comparative programming languages course**. We won't be dabbling in a long list of classic old languages or fancy new ones. We don't have a course like that because (1) that sort of thing is built into other places of the curriculum, (2) in ten years, there will be all new languages being used, (3) if you ever need to learn a new language for a job or research project, you should be able to learn it on your own fairly well, and (4) I don't think it's as interesting.

> *I often feel that the American programmer would profit more from learning, say, Latin than from learning yet another programming language.* —Edsger W Dijkstra, EWD611

Instead, this course steers a middle course between the other two models, giving an even-handed mix of theory and practice. We will study formal models for specifying language features and reasoning about them. We will write many small analyzers, interpreters, and compilers to consider the implementation of language features close-up.

TEXTBOOK. There is no textbook for this course. In past years I have used Tucker and Noonan, *Programming Languages: Principles and Paradigms,* McGraw Hill, but this year I am providing all the material directly through lecture and handouts. I may, however, provide and assign short passages from other relevant texts.

OBJECTIVES. The chief goal of this course is to understand how the features of a programming language are specified and implemented and how constraints on the models and systems effect programming language design decisions. This objective is pursued by two main activities:

- using formal models to specify, describe, and reason about programming languages and their features.
- implementing programming languages by writing parts of analyzers, interpreters, and compilers.

THEMES. A variety of themes are interwoven among the topics of this course and drive the decisions about what to cover.

Programming paradigms. The old way of categorizing programming languages into paradigms (like imperative, object-oriented, and functional) is showing its age. Newer languages tend to incorporate features from many paradigms. Even though this distinction is no longer useful for categorizing *languages*, it still is handy for categorizing *features*, as well as programming *styles/mindsets* (the program is a *list of instructions* vs *set of interacting objects* vs *composition of functions*).

We will use a collection of small languages (designed for this course) to study various programming language features. The imperative and object-oriented features will be studied using Java-like languages (most of them actually subsets of Java); the functional features will be studied using subsets of ML.

Formalization. We will consider various models for defining the semantics of a language, at different levels of abstraction.

Implementation. We will consider various ways to implement programming languages, generally: interpretation (write a program that executes programs in a language, modeling a machine at various levels of abstraction or following a formal model at various levels of consistency); source-to-source compilation (write a program that translates programs from one language to another language); and compilation (write a program that translates programs from one language to the machine language for a real or virtual machine).

Static analysis. We will consider what analyses can be performed on a program in a language without running the program to catch certain errors or make optimization decisions. (Typechecking is the most important example of static analysis.)

OUTLINE. See the course website for schedule of these topics.

I. Prolegomena
   A. Abstraction, paradigms, etc
   B. Compiler structure
   C. Implementation strategies
   D. Programming language vocabulary

II. A warm-up: grammars and parsing
   A. Context-free grammars and other lexical and syntactic specification
   B. Parsing algorithms
   C. Concrete vs abstract syntax
   D. The Visitor pattern

III. Imperative programming
   A. Types and type-checking
   B. Formal semantic models
   C. Implementation
   D. Features
      1. Loops, blocks, and scope
      2. Switch statements
      3. Casting
      4. Floating point
   E. Procedures

1. Local scope and activations
2. Parameter passing modes
3. Nested procedures

F. Dynamic memory

1. Pointers
2. Arrays
3. Records (structs)

IV. Interlude: various topics

A. The history of programming languages
B. Gotos
C. Compilation
D. The lambda-calculus

V. Object-oriented programming

A. Objects and classes
B. Types, subtypes, and polymorphism
C. Inheritance
D. Alternate formulations

VI. Functional programming

A. Bootstrapping a functional language
B. Types and type soundness proofs
C. Tail form and continuation-passing style

Course procedures

How we do this course. The typical rhythm for this course is (1) you will read up on a new programming language concept or language feature in a handout and practice some related exercises (in homework); (2) we will explore a formal description of that feature using semantic models (in lecture); (3) we will inspect an implementation of the feature or concept written in Java, ML, or Python (in class); (4) you will extend the implementation to a related feature or concept written in Python (in a project).

Preparation for class. I will regularly give handouts that need to be read before class, including practice problems. These are an important part of the class. There is more material to be covered than there is time for me to talk about in lecture. In order for this course to work, you must work on learning the basics of each topic before we discuss it in class.

Assignments. There are two kinds of assignments. Some assignments will have the purpose of preparing you for class; these will have a small weight toward the semester grade, will be checked only for completeness, and may be collaborated on. Other assignments will have the purpose of assessing your progress on the formal side of the course; these are in leu of a midterm examination and must be done independently (see *Academic integrity* below for more details).

Projects. Most of the work outside of class will be on programming projects, of various lengths. Each of these will involve writing part of a programming system (that is, an interpreter, analyzer, or compiler) for a small Java-like or ML-like language. The projects themselves, however, will be written in Python (that is, Python will be our *implementation language*, the language we use to implement the languages we're studying). Most of the in-class examples will also be in Python, though a few may be in Java or ML. These should be worked on independently (see *Academic integrity* below). See the course website for approximate assignment and due dates. Note that some projects will have overlapping timeframes.

Project will be graded using automated scripts. I will look only at your projects' output; I do not intend to look at your code. Accordingly, programming style will not be a factor in grading; you should follow good coding style for your own sake. Moreover, I will not give you the test cases I will use in grading; devising your own test cases is part of the project (though not a part that needs to be turned in).

Examination. The final exam (Tuesday, May 6, 1:30 PM) will cover the entire semester. Moreover, some assignments will contain exam-like problems that play a role somewhat like a take-home midterm.

Grading.

instrument	*weight*
Homework	25
Projects	50
Final exam	25

Policies etc

Academic integrity. Although class-preparation problems may be worked on collaboratively, exam-like problems must be done independently. Students are allowed to discuss these problems for clarifying main ideas, but solutions may not be shared. Projects also must be done independently (as, for example, in CSCI 235 or CSCI 245). As in other computer science courses with significant project components, students may discuss the project in the abstract, help each other debug, and especially share test cases; students may not program together, however, and correct code may not be shared.

Students may not use solutions to assignments or projects from the Internet or other resources (electronic, print, or human). If students get material *ideas* from other sources that help them in solving a project problem, then that source must be cited in a comment (analogous to citing a source in a research paper). Using such sources is discouraged; the project in this course are intended to be solved from scratch by the students.

An assignment or project on which a student violates these policies will be rejected. Repeated offenses will be handled through the college's official disciplinary procedures.

Late assignments. You are allowed a total of three days during the course of the semester, which may be divided up (in whole-day units) among the projects in any way. (Note this is one more late day than I allow in CSCI 235 or CSCI 245.) Other late projects and assignments will not be accepted. *Please notify me if you are turning an assignment in late; this helps me plan grading.*

Attendance. Students are expected to attend all class periods *on time*. It is courtesy to inform the instructor when a class must be missed.

Examination. The final exam is Tuesday, May 6, 1:30 PM. I do not allow students to take finals early (which is also the college's policy), so make appropriate travel arrangements.

Gender-inclusive language. For academic discourse, spoken and written, the faculty expects students to use gender inclusive language for human beings. This is not actually relevant for this course, but it's an official college policy that the syllabus contain some sort of notice like this.

Special needs. *Institutional boilerplate:* Wheaton College is committed to providing reasonable accommodations for students with disabilities. Any student with a documented disability needing academic adjustments is requested to contact the Academic and Disability Services Office as early in the semester as possible. Please call 630.752.5941 or send an e-mail to `jennifer.nicodem@wheaton.edu` for further information.

My own statement: Whenever possible, classroom activities and testing procedures will be adjusted to respond to requests for accommodation by students with disabilities who have documented their situation with the registrar and who have arranged to have the documentation

forwarded to the course instructor. Computer Science students who need special adjustments made to computer hardware or software in order to facilitate their participation must also document their needs with the registrar in advance before any accommodation will be attempted.

Office hours. My office hours this semester are MWThF 8:45–10:15 am with an extra hour (ie, until 11:15 am) on Thursdays. My teaching schedule this semester is very afternoon-heavy (12:45–4:20 pm MWF, 1:15–3:05 pm Th), so please do not expect me to be available in the afternoons, especially MWF.

If these times do not work for you, please make an appointment for sometime Tuesday or late Thursday afternoon. An email saying "May I stop by in five minutes?" is adequate for making an appointment.

Also, any time my door is closed, it means I'm doing something uninterruptable, such as making an important phone call. Rather than knocking, please come back in a few minutes or send me an email.

After-hours help. I have found that because of the complexity of the projects in this course, students sometimes need clarifications to the project description—or bug fixes in the given code—late at night or over the weekend. I will do my best to be reasonably available to answer questions (and fix things, if necessary). If you have a question about a project (or suspect there is a bug in code I have given you), do not hesitate to contact me by text using the cell number given at the top of this syllabus. If you do, please follow these guidelines:

- Keep the text simple, notifying me what kind of a problem you have; provide the details in an accompanying email.
- In that accompanying email, provide all the information I need to identify the problem: you may need to attach your code and/or a test case that illustrates the problem (what online support sites call a "complete, minimal example"), and you should describe the results you are getting and how they differ from what you expect.
- Please do not text me between 11 PM and 5 AM, and do not text me on Sunday before 8 PM.
- As mentioned above, "I don't understand this part of the project description" and "I think there is a bug in the code you gave us" are appropriate reasons to text me after hours. "I'm stuck on this bug in my own code" may be an appropriate reason, but consider whether it can wait until business hours the next working day. "I don't know where to start on this project" is never an appropriate reason to text me after hours.

Dress and deportment. Please dress in a way that shows you take class seriously—more like a job than a slumber party. (If you need to wear athletic clothes because of activities before or after class, that's ok, but try to make yourself as professional-looking as possible.) If you must eat during class (for schedule or health reasons), please let the instructor know ahead of time; we will talk about how to minimize the distraction.

Electronic devices. Please talk to me before using a laptop or other electronic device for note-taking. You will need to convince that it truly aides your comprehension. No student has convinced me yet, but if youre the first one then I will give you a stern warning against doing anything else besides note-taking. Trying out programming concepts on your own during class time (for example) is not productive because it takes you away from class discussion. You cannot multi-task as well as you think you can. Moreover, please make sure other electronic devices are silenced and put away. ***Text in class and DIE.***