

Catalog Description

Study of features of programming languages and of the methods used to specify and translate them. Topics include LISP, virtual machines, syntax and semantics, binding times, scoping rules, implementation choices, procedure calling, and parameter passing. CSC 282 Data Structures and Algorithms is a prerequisite for this course.

Instructor

Dr. Gene B. Chase. Office, Frey 123. Phones: office 766-2511, ext. 2770; home 766-7904. Electronic mail: chase@messiah.edu Home page: www.GeneBChase.com. My office hours are posted on my office door, where you may sign for an automatic appointment.

Note: This is a new office location.

Objective

I have two objectives for this course.

1. To present computer science as a mathematically rigorous discipline.
2. To equip you to understand the spectrum of programming languages so that you can choose the right language for the job or create the right language for the job if no appropriate one exists.

Outcomes

1. You will program in C++, Lisp, and Prolog.
2. You will compare the features of these and other languages (Java, Perl, APL, XSLT, and ML for example) in three closed-book exams.
3. You will prove small program segments to be correct.
4. You will write a paper that uses multiple sources and specific program code to defend or to critique your choice of a programming language for a specific task.

Materials Needed

Robert W. Sebesta. *Concepts of Programming Languages*, seventh edition. Pearson/Addison-Wesley, 2006.

Possibly other source books depending on the language you choose to study, if there are no good on-line tutorials for your topic area.

Americans with Disabilities Act

Messiah College welcomes students with disabilities. If you have a documented disability and wish to discuss academic accommodations for this specific course, please contact me as soon as possible. All disability accommodations must be pre-approved through Dr. Keith Drahn, in the Office of Disability Services, telephone x7258.

Academic Integrity

Plagiarism—representing another's work as your own, copying another person's work without credit, or allowing your work to be copied—will surely result in a lower grade in this course, and may result in failing the course depending on its severity. You must document any sources that you use, whether from the internet, another person, or printed materials. This includes especially the work of other students who are currently taking this course or who have taken this course before. Academic integrity is broader than

plagiarism. It includes such things as returning library materials promptly so that you are not keeping another student from completing his or her work.

All students at Messiah College must read and abide by the College's policy on academic integrity, which is found in the 2004–2005 edition of the Student Handbook on pages 108–110. The handbook is available on the Internet at

www.messiah.edu/offices/student_affairs/student_handbook/resources/policies.pdf

Grading

45% exams. Three closed-book tests worth 15% each: On September 30, October 31, and December 14. The third, although scheduled during the regularly scheduled final examination week, is not comprehensive.

10% participation. Keep up with daily readings from your text and discuss them in class. Report anticipated absences beforehand unless an emergency. Report emergency absences as soon as possible. Unexcused absences are an automatic 1% deduction of the course grade. Unannounced quizzes on readings are a way to find out if you are participating by keeping up with the readings.

20% labs. Four lab reports are weighted 5% each: Turing machines, Scheme, C++, and Prolog.

25% project. A project topic is chosen by September 14, and a list of three sources for your project is due September 21. You will give a 10-min. report on your project on December 5, 7, or 9 (depending on alphabetical order). An 8-10 page paper analyzing your project topic is due at 5:30 p.m. on Friday, December 9.

Overview of Topics

- Programming languages types
 - General: Functional, imperative, object-oriented, other (dataflow, visual, ...)
 - Purity vs. convenience: E.g. assignment as expression, ternary if
 - Special-purpose: SQL, process control, graphics, gaming, ...
- Criteria for judging a good programming language
 - Semantic fit to problem domain
 - Chapter 1 criteria
- History of programming languages
 - People: 1 John Backus & team, 2 Grace Murray Hopper & team, 3 John McCarthy, 4 John Kemeny & Thomas Kurtz, 5 Nikolas Wirth, 6 Ken Iverson, 7 Ralph Griswold, 8 Ken Thompson & Dennis Ritchie, 9 Bjarne Stroustrup, 10 James Gosling, 11 Larry Wall
 - Languages: 1 Fortran, 2 Cobol, 3 Lisp, 4 Basic, 5 Algol and Pascal and Modula, 6 APL, 7 Snobol and Icon, 8 C, 9 C++, 10 Java, 11 Perl
- Compiler theory, general
- Syntax
 - Tokens, lexemes
 - Recognition, generation, derivations
 - Chomsky Hierarchy of Grammars
 - FSG, FA, and regular expressions; CSG, BNF, EBNF and parse trees; PDA and LBA; Turing Machines

- Parsing
 - Top down parsers, e.g. recursive descent as an LL parser
 - Bottom up (shift-reduce parsers), e.g. LR table parsing
- Ambiguous grammars
- Semantics
 - Static semantics, e.g. attribute grammars
 - Dynamic semantics to prove program correctness
 - Operational
 - Axiomatic
 - Assertions: Preconditions, postconditions, loop invariants, weakest preconditions
 - Denotational
 - E.g. short-circuit evaluation
 - E.g. scope of index in a loop
- Translation
 - Preprocess, compile, link, load, execute
 - Interpret
 - Eager vs. lazy evaluation
- Learning Perl
- Learning C++
- Terms to know
 - side effects, operator overloading, orthogonality, aliasing, object-based vs. object-oriented
- Binding times and lifetime
 - Times: Language definition time, compile time, link time, load time, run time
 - Memory management: garbage collection and memory leakage, dereferencing and dangling pointers; eager vs. lazy heap management
- Scoping
 - Dynamic vs. static scoping
 - Local vs. non-local (misnamed global) scoping
- Types
 - Strong vs. weak
 - Type compatibility: name, structural
 - Subtyping
 - Union types (free unions, discriminated unions)
 - User-defined typing (typedef, objects as types)
 - Type conversions: explicit, implicit (promotion), coercion
- Data structures
 - Persistent data structures
 - Heap management
 - new, malloc; delete, dispose, free
 - Whole-data structure operations (e.g. APL)
- Arrays
 - Implementations
 - As objects (Java), hence "jagged" permitted
 - Associative arrays, as hashes (Perl, Javascript, possibly PHP), hence "jagged" permitted
 - Scientific arrays, as adjacent memory (Fortran), hence data parallelism is easy
 - Column-major order (Fortran) vs. row-major order (all other languages)
 - Slices
 - Subscript binding (static, fixed stack-dynamic, fixed heap-dynamic, heap dynamic)
- Variables and constants

- Descriptors, esp. string descriptors
 - r-values vs. l-values
 - implicit vs. explicit declarations
 - static vs. stack-dynamic vs. implicit heap-dynamic vs. explicit heap-dynamic
 - Named constants
 - Anonymous variables
 - Control structures
 - `break` `labelName`, `continue`
 - assertions (§14.4.7, but anticipated by Konrad Zuse in 1945)
 - Functional abstraction: Subprograms
 - Semantic models of parameter passing methods: in, out, in-out (Chapter 9)
 - Pass by value, by result, by value-result, by reference, by name
 - Implementation of the above (Chap. 10): activation records, displays, deep vs. shallow access
 - Subprograms as parameters
 - Generics
 - Data abstraction
 - Records (struct)
 - Enumerations
 - Assemblies, namespaces, packages.
 - Object-based programming: C++, Java, Ada95, C#, JavaScript
 - Paper by Peter Wegner, 1989, contrasting object-based with object-oriented programming.
 - Polymorphism, virtual methods, single inheritance vs. multiple inheritance, private vs. public vs. package vs. protected scope, friend functions in C++, static binding (for final, private, or static methods).
 - Which subclasses are subtypes? (§§ 12.3.2 and 12.5.2)
 - Functional programming
 - Examples: Scheme, ML, others (Derive, Haskell)
 - Functions as first class entities; defining `mapcar` (§15.5.11.2)
 - The read-eval-print loop
 - Constraint programming: Logic programming languages
 - Predicate calculus extends propositional calculus
 - Resolution theorem proving for Horn clauses, via unification
 - Declarative vs. procedural (with tracing) semantics
 - Prolog as an example of a logic programming language
 - Pattern matching with backtracking as another model of parameter passing
 - The negation problem and the closed world assumption
 - Applications of Prolog: database access, expert systems, natural language processing
 - Concurrency (parallelism)
 - Event handling, exceptions, and errors
 - Multithreading vs. coroutines: `resume`
 - Dijkstra's guarded statements: `if`, `while`
 - Physical vs. logical concurrency
 - Journal theme issue on concurrency: “Make way for multiprocessors,” *ACM Queue*, Vol. 3, No. 7, Sept. 2005.
- [Much omitted here, some of which is a part of CSC 416 Operating Systems, and other parts of which make for good projects.]

Project Ideas

Project suggestion: Get an early start!

You do not have to wait until the due date for picking a paper topic in order to discuss it with me. Since you or your team must choose a topic different from those of your classmates, the earlier you decide, the more latitude you will have. Some good languages to analyze include APL, Smalltalk, Snobol4, Haskell, ML, Python, or Common Lisp.

For the practically minded, <ftp://gandalf.umcs.maine.edu/pub/msdos/compilers> has public domain/shareware compilers for some languages that would be appropriate. If you need a computer to mount software on, you may arrange with me to use a computer in Frey 366.

For the theoretically minded, some programming language issues are discussed in the Alan M. Turing awards, an annual award that is sometimes considered the “Nobel prize” of Computer Science. I’d be glad to discuss them with you. Here is a list of award winners:

< http://en.wikipedia.org/wiki/Turing_award >. You can see how many of the papers address programming language issues by the names that we will meet in our course: Edsger Dijkstra (Algol, parallel construction), Robert Floyd (axiomatic semantics), John Backus (Fortran, functional programming), Ken Iverson (APL), Ted Codd (SQL), Robin Milner (ML), Alan Kay (object-oriented programming, or Dana Scott (denotational semantics).

I’m a “programming languages junkie,” so I’d be delighted to go on a journey with you to a topic that I don’t know anything about.

The language Octave

What are some good characteristics of a language for graphics?

That’s a good question to ask if you are taking both Organization of Programming Languages (OPL) and Computer Graphics (CG) in the same semester, and you want to have a single project that spans both interests.

In other years in this course, students have discussed the expensive graphics language Flash Action Script by Macromedia and the free ray-tracing language POV-Ray 3.1. You might be interested in the new XML graphics standard language SVG, although by itself, it’s slim pickings as programming languages go: no local variables, no event handling, no subroutines, and no complex data types. You might be interested in the Virtual Reality Modeling Language (VRML), which actually does have an event-handling model that you could contrast with Java’s.

One language used for graphics mostly, but a variety of engineering and mathematical applications more generally is Matlab. We have Matlab in our labs. It is not free.

Octave is free software to run under Linux whose programming language is largely compatible with Matlab. Visit <http://www.octave.org/> to download it. Octave is sometimes misspelled Oactive—just a heads-up for you if you are using a search engine to learn more.

I haven’t had much experience with Matlab. I published one article using Matlab back in 1992, and I tried using Matlab without success to draw the data for Dr. Douglas Phillippy’s dissertation in 3D for him one time.

So here's my suggested two-course project: for CG draw Dr. Phillipy's data; for OPL, discuss the suitability of the features of the language that you used, based on Sebesta's criteria for a good language, supplemented by our class discussion of Sebesta. If you want to use this as an opportunity to learn a little Linux in the process, using Octave would be a nice supplement, either in our Frey 366 laboratory, or on your personal box.

The semantic web

Is Prolog a suitable language for the "Semantic Web" project? See <http://www.w3.org/RDF/>

We will be learning some Prolog in this course. I got the idea for this question from the article "Back to the Future," by Michael Swaine, the editor of Dr. Dobb's Journal. It was in the August 2001 issue, on pages 97-102. Our Library has a copy.

Information about the Resource Description Framework (RDF) is available at <http://www.xml.org>
What would a good Resource Description Language (RDL)— to use Swain's term—look like? Is Prolog better than SQL?

COBOL

We joke about how old fashioned COBOL is: No recursion; limited scoping options. But it supports random access files and indexed sequential files. Both industry and government have thousands of lines of COBOL to maintain. If you are interested in a resume-builder, you might want to learn COBOL. It's an easy language to learn. I have a free copy of Realia COBOL and a book about the language. Flexus offers a free Java front ends to allow COBOL to use a browser as its client (www.infogoal.com/cbd/cbdz001.htm). There are COBOL compilers that compile to the Java Virtual Machine. A 10-day evaluation version of one is available.

(grunge.cs.tu-berlin.de/~talk/vmlanguages.html) Obviously with that short a window of opportunity, you would probably be best-served by learning COBOL using Realia and then switching to see how it performs under the JVM. By the time that you read this, a free COBOL compiler might be available for the JVM.

Other languages

The site grunge.cs.tu-berlin.de/~talk/vmlanguages.html just mentioned lists scores of languages that run under the JVM, giving hints for other appropriate languages to study as a project for OPL. There are functional languages like Scheme and ML; object-oriented languages like Smalltalk and Eiffel; scripting languages like Tcl and Perl; stack-oriented graphics languages like Forth (or like Postscript, although that's not available to run under the JVM yet).

Ideas from previous semesters

Don't use these ideas directly. I've commented on them to help you see what would make a good project.

Why Mercury is a better functional language than Prolog (Jason Barkanic)

How strong typing can improve a functional language. Mercury is being used by Microsoft, so it was a good choice. Jason did a great job defending his position.

Why Smalltalk is the wave of the future (Scott Benedict)

Since Smalltalk (1972) was probably the second object-oriented (OO) language after the original Simula I (1962) and Simula 67, it sounds funny at first to say that an old language is better than the most recently invented OO language. That's what made Scott's paper so interesting. He defended his thesis well.

Java for databases (Chad Braun-Duin)

The title is deceptive, since it doesn't make clear that the paper in fact is evaluating and comparing three approaches to putting Java in a database: SQLJ is now an ANSI standard in syntax, semantics, and binary compatibility; JDBC is a call-level SQL API based on ODBC; and then Chad did a third that I forget right now.

JavaSpaces (Erica Brubaker)

JavaSpaces is a parallel programming language based on the language Linda, the work of Yale University's David Gelerntner in 1989 (see Nicholas Carriero and David Gelernter, "Linda in context," *Communications of the ACM*, Vol. 32, No. 4, April 1989; see also www.cs.yale.edu/Linda/linda.html). IBM has a similar language, TSpaces. Erica discussed the programming language aspects of JavaSpaces.

Is Delphi better than Visual Basic for business applications? (Emily Clepper)*Is C++ better than Eiffel?* (David Clymer)*PHP3 is a good alternative to ASP and is open source* (Jamie Detweiler)*How C# is better than C++ or Java* (Joe Hoot)*How Perl is better than Java for server-side CGI* (Mike Horst)*How Prolog models relational databases well* (Melissa Kofroth)*Is Perl better than JavaScript?* (Graham Wert)

It's not that these other examples don't deserve a paragraph of explanation, but by now you get the idea that an excellent project will evaluate a computer language, you get the idea that one way to evaluate something is to compare it with something else, and you get the idea that the best comparison requires a context: Better for what?

Course Outline

Day	Topics (tentative) and due dates (not tentative)
August	
1 W 31	Programming language types
September	
2 F 2	Criteria for judging a good programming language (Read Chapter 1)
3 M 5	History of programming languages (Read Chapter 2.1–2.8)
4 W 7	Compiler theory, general (Read Chapter 2.9–2.19; Chapter 4 to p. 182)
5 F 9	Syntax (Read Chapter 2.20; think about project ideas)
6 M 12	BNF for CFG (Read Chapter 3.1–3.3)
7 W 14	Chomsky Hierarchy overview; Project topic area and partner selection if any are due
8 F 16	Chomsky Hierarchy (cont.)
9 M 19	Lab 1: Turing machines
10 W 21	Static semantics: Attribute grammars (Read §3.4); Project abstract and three references due
11 F 23	Operational semantics (Read §3.5 to p. 150)
12 M 26	Axiomatic semantics (Read §3.5 to p. 163); Lab 1 due
13 W 28	Loop invariants, weakest preconditions
14 F 30	Exam 1 , closed book
October	
15 M 3	Denotational semantics, overview, numeric examples (Read remainder of Chap. 3)
16 W 5	Denotational semantics, boolean examples (Read internet assignment by

Patrick Winston on C++)

- 17 F 7 Denotational semantics, control structures (Read §§12.5–12.6)
 18 M 10 **Lab 2:** C++ (Read Peter Wegner's article, distributed; quiz at start of lab over reading)
 19 W 12 **Lab 3:** C++ (Read §§12.1–12.4 carefully enough to answer these questions:
 Are subclasses subtypes? Is multiple inheritance good? How does Smalltalk differ
 from Java or C++?)

[20] F 14 Presidential Inauguration Day; self-paced **Lab 4:** C++

- 21 M 17 Recursive descent parsing (Read Chap. 4, pp. 183–192)
 22 W 19 Bottom up parsing [omit if we are behind] (Read Chap. 4, pp. 193–200); **Labs 2–4 due**

20–23 Fall Recess

- 23 M 24 Binding times (Read Chap. 5 to p. 222)
 24 W 26 Types (Read §§5.5–5.7 and §7.4)
 25 F 28 Scoping (Read §§5.8–5.10)

26 M 31 **Exam 2**, closed book

November

- 27 W 2 **Lab 5:** Scheme and functional programming (Read §§15.1–15.5)
 28 F 4 Implementing control structures (Read selections to be assigned from Chap. 8)
 29 M 7 Parameter passing: by value, by result, by value-result, by reference (Read §9.5 to top p.
 394)
 30 W 9 Parameter passing: by name; subprograms as parameters (Read p. 394 & §9.6)
 31 F 11 Implementation of parameter passing (Read pp. 394–408); **Lab 5 due**
 32 M 14 Implementing strings, enumerations, subranges (Read §§6.1–6.4)
 33 W 16 Implementing arrays, records, pointers, (Read §§6.5–6.9)
 34 F 18 Predicate logic (Read §§16.1–16.3)
 35 M 21 **Lab 6:** Pure prolog (Read §§16.4–16.6)

23–27 Thanksgiving Recess

36 M 28 Concurrency: exceptions, coroutines, guarded statements (Read §8.5 and selections of
 Chap. 13 to be assigned)

37 W 30 [Catch-up]; **Lab 6 due**

December

38 F 2 [Catch-up]

39 M 5 **Project reports** if needed.

40 W 7 **Project reports**

41 F 9 **Project reports**; Review for final exam; **Projects due**

Wednesday, December 14, 8:00–10:00 p.m.: **Exam 3**, two hours, closed book.