

COSC 382 Organization of Programming Languages
Messiah College**Syllabus**
Fall Term 2006**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. COSC 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.

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

1. This notebook.
2. Robert W. Sebesta. *Concepts of Programming Languages*, seventh edition. Pearson/Addison-Wesley, 2006.
3. 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

Your signature on the second day of class certifies that you have read and understood and intend to comply with this policy.

Grading

45% Exams. Three closed-book tests are worth 15% each. The third—although scheduled during the regularly scheduled final examination week—is not comprehensive

30% Labs. Three lab reports are weighted 10% each: Turing machines, C++, and Lisp & Prolog.

25% project. A project topic is chosen by date as scheduled (2%), and a list of three sources that you have read for your project is due as scheduled (3%). You will give a 10-min. report on your project on the assigned date (8%). An 8-10 page paper analyzing your project topic is due at 5:30 p.m. on the assigned date (12%). **Lateness except for emergency is not acceptable.**

Absences/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 a 1% deduction of the course grade per absence. Unannounced quizzes worth 1% on assigned readings are my way to find out if you are participating by keeping up with the readings. Extra credit small class presentations on Wednesdays can earn 1% bonus. In our course, unlike extra credit Wednesdays in some of my other courses, the presentation must be related to the topic of our course, but may not be from your textbook.

Overview of Topics

This list of topics includes everything that this course could be about. We will not do all of it. When we review for exams, we will check off which topics relate to that exam.

G Programming languages types

G General: Functional, imperative, object-oriented, other (dataflow, visual, ...)

G Purity vs. convenience: E.g. assignment as expression, ternary if

G Special-purpose: SQL, process control, graphics, gaming, ...

G Criteria for judging a good programming language (Sebesta, Chapter 1)

G Semantic fit to problem domain

G Chapter 1 criteria

G History of programming languages (Sebesta, Chapter 2)

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

- G Compiler theory, general (Sebesta, Chapters 1 & 4 in part)
- G Syntax (Sebesta, Chapters 3, and 4.1)
 - G Tokens, lexemes
 - G Recognition, generation, derivations
 - G Chomsky Hierarchy of Grammars
 - G FSG, FA, and regular expressions; CSG, BNF, EBNF and parse trees; PDA and LBA; Turing Machines
 - G Parsing
 - G Top down parsers, e.g. recursive descent as an LL parser
 - G Bottom up (shift-reduce parsers), e.g. LR table parsing
 - G Ambiguous grammars
- G Semantics (Sebesta, Chapter 3)
 - G Static semantics, e.g. attribute grammars
 - G Dynamic semantics to prove program correctness
 - G Operational
 - G Axiomatic
 - G Assertions: Preconditions, postconditions, loop invariants, weakest preconditions
 - G Denotational
 - G E.g. short-circuit evaluation
 - G E.g. scope of index in a loop
- G Translation
 - G Preprocess, compile, link, load, execute
 - G Interpret
 - G Eager vs. lazy evaluation
- G Learning Perl
- G Learning C++
- G Terms to know
 - G side effects, operator overloading, orthogonality, aliasing, object-based vs. object-oriented
- G Binding times and lifetime
 - G Times: Language definition time, compile time, link time, load time, run time
 - G Memory management: garbage collection and memory leakage, dereferencing and dangling pointers; eager vs. lazy heap management
- G Scoping
 - G Dynamic vs. static scoping
 - G Local vs. non-local (misnamed global) scoping
- G Types
 - G Strong vs. weak
 - G Type compatibility: name, structural
 - G Subtyping
 - G Union types (free unions, discriminated unions)
 - G User-defined typing (typedef, objects as types)
 - G Type conversions: explicit, implicit (promotion), coercion

- G Data structures
 - G Persistent data structures
 - G Heap management
 - G new, malloc; delete, dispose, free
 - G Whole-data structure operations (e.g. APL)
- G Arrays
 - G Implementations
 - G As objects (Java), hence “jagged” permitted
 - G Associative arrays, as hashes (Perl, Javascript, possibly PHP), hence “jagged” permitted
 - G Scientific arrays, as adjacent memory (Fortran), hence data parallelism is easy
 - G Column-major order (Fortran) vs. row-major order (all other languages)
 - G Slices
 - G Subscript binding (static, fixed stack-dynamic, fixed heap-dynamic, heap dynamic)
- G Variables and constants
 - G Descriptors, esp. string descriptors
 - G r-values vs. l-values
 - G implicit vs. explicit declarations
 - G static vs. stack-dynamic vs. implicit heap-dynamic vs. explicit heap-dynamic
 - G Named constants
 - G Anonymous variables
- G Control structures
 - G `break` `labelName`, `continue`
 - G assertions (§14.4.7, but anticipated by Konrad Zuse in 1945)
- G Functional abstraction: Subprograms
 - G Semantic models of parameter passing methods: in, out, in-out (Chapter 9)
 - G Pass by value, by result, by value-result, by reference, by name
 - G Implementation of the above (Chap. 10): activation records, displays, deep vs. shallow access
 - G Subprograms as parameters
 - G Generics
- G Data abstraction
 - G Records (struct)
 - G Enumerations
 - G Assemblies, namespaces, packages.
- G Object-based programming: C++, Java, Ada95, C#, JavaScript
 - G Paper by Peter Wegner, 1989, contrasting object-based with object-oriented programming.
 - G 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).
 - G Which subclasses are subtypes? (§§ 12.3.2 and 12.5.2)
- G Functional programming
 - G Examples: Scheme, Common Lisp, others (Derive, ML, Haskell)
 - G Functions as first class entities; defining `mapcar` (§15.5.11.2)
 - G The read-eval-print loop

- G Constraint programming: Logic programming languages
 - G Predicate calculus extends propositional calculus
 - G Resolution theorem proving for Horn clauses, via unification
 - G Declarative vs. procedural (with tracing) semantics
 - G Prolog as an example of a logic programming language
 - G Pattern matching with backtracking as another model of parameter passing
 - G The negation problem and the closed world assumption
 - G Applications of Prolog: database access, expert systems, natural language processing
 - G Concurrency (parallelism)
 - G Event handling, exceptions, and errors
 - G Multithreading vs. coroutines: `resume`
 - G Dijkstra's guarded statements: `if`, `while`
 - G Physical vs. logical concurrency
 - G 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

Get an early start!

Frey 351 has a big pile of papers done by student of this course in former years. The names of the students have been erased so that I can include my grading of the papers. This will give you a great feel of the scope of projects, and what counts as A level work. More is mentioned about that below.

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 public domain/shareware compilers for some languages are available, and googling <free compilers> or checking out Yahoo's <dir.yahoo.com/Computers_and_Internet/Programming_and_Development/Languages>. 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: <en.wikipedia.org/wiki/Turing_award>. See how many of the papers address programming language issues. Here are names that we will meet in our course: Edsger Dijkstra (Algol, parallel constructions), 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.

XSL

Contrast XSL, the programming language in XML syntax, with another language.

VRML (Virtual Reality Modeling Language)

Contrast VRML's event-handling model with that of Java.

Octave

This free software under Linux is largely compatible with Matlab. We have both at Messiah College. Visit <http://www.octave.org/> to download Octave. 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 in 1992, and I tried using Matlab without success to draw the data for Dr. Douglas Phillippy's dissertation in 3D for him once.

The semantic web

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

RDF (Resource Description Framework)

See < 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. See < grunge.cs.tu-berlin.de/~tolk/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/~tolk/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 seven 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?