

Master Course Syllabus
School of Engineering and Computer Science
Washington State University Vancouver

CS 355
Programming Language Design
3 Semester Hours

Catalog Description

Design concepts of high-level programming languages; survey of existing languages, experience using some languages; formal specification of syntax and semantics.

Prerequisite Courses

CS 223	Advanced Data Structures
CS 224	Programming Tools

Prerequisite Topics

- Proficient in at least one imperative programming language.
- Experience in editing, compiling/linking, executing, testing, and (source level) debugging of moderately sized programs.
- Experience with multiple source file programs
- Implementation of common data structures for lists, trees, stacks, queues, graphs, hash tables, etc.

Measured Course Outcomes

Students taking this course will (among other things):

1. Classify and evaluate programming languages according to the design paradigm and features provided by each. When evaluating a language's suitability for a specific task, the student will identify tradeoffs considering such issues as space efficiency, time efficiency (of both the computer and the programmer), safety, and power of expression. (*Contributes to performance criteria I-3*).
2. Write at least one program module that defines an Abstract Data Type in a language providing appropriate support for encapsulation. The student will enumerate the benefits of using these mechanisms for writing large programs. (*Contributes to performance criterion K-2*).
3. Write a program in a functional language (e.g., Scheme, Common Lisp, ML) or in a logic programming language (e.g. Prolog, CLIPS) using the language's intended paradigm. (*Contributes to performance criterion I-2*).

4. Write a short description or review of a programming language or language feature described in recent literature (e.g., from ACM Transactions on Programming Languages and Systems). (*Contributes to performance criterion H-4*).

Required Textbooks

Concepts of Programming Languages (7th edition) by Robert W. Sebesta, Addison Wesley, ©2005, ISBN 0321330250.

or

Programming Language Pragmatics (2nd Edition) by Michael L. Scott, Elsevier © 2006, ISBN 0-12-633951-1.

Reference Material

ANSI Common Lisp by Paul Graham, Prentice Hall, ©1995, ISBN 0133708756.

Objective-C Pocket Reference by Andrew M. Duncan, O'Reilly, ©2003, ISBN 0-596-00423-0

Major Topics Covered in the Course

1. History and evolution of programming languages. Types of languages: imperative, functional, object oriented, logic, scripting, static versus dynamic typing, etc.
2. Syntax and semantics.
3. Translation strategies: lexical and syntax analysis, symbol tables, semantic analysis, code generation, optimization, interpreters, intermediate languages, virtual machines, separate compilation.
4. Declarations and types: names, binding, scopes, lifetimes.
5. Type Systems: primitive and aggregate types, parameterized types, type checking, Abstract Data Types.
6. Expressions.
7. Statements and flow control.
8. Subprograms: macros, procedures, functions. Formal and actual parameters, pass by value versus pass by reference, code reuse, overloading.
9. Subprogram implementation: calling conventions, activation records, stack frames, etc.
10. Support for Abstract Data Types: encapsulation, programming to an interface, containers, iterators.

11. Support for Object-Oriented Programming: ADT's, inheritance, polymorphism, composition, interface inheritance, multiple inheritance, dynamic dispatch, virtual function tables.
12. Memory management: static versus dynamic allocation of variables, programmer explicit, reference counting, automatic garbage collection.
13. Generic programming.
14. Exception handling.
15. Functional programming: computation by evaluation instead of by side effect, functions as first-class objects, closures, lazy evaluation, memoization.
16. Logic programming.

Laboratory Projects

- Programming project that includes the implementation of an abstract data type in a language supporting encapsulation and separate interface and implementation definitions. The project will include testing and use of the ADT in a complete program.
- Programming project that uses a non-imperative paradigm (i.e., either a functional or logic programming approach).

CSAB Category Content

	FUNDAMENTAL	ADVANCED		FUNDAMENTAL	ADVANCED
Data Structures	0	0	Computer Organization and Architecture	0	0
Algorithm & Software Design	1	0	Concepts of Programming Languages	2	0

Oral and Written Communications

The student will write a short paper (2 to 3 pages in length) that reviews a programming language or language feature described in the recent professional or research literature. The intended audience of the paper is experienced programmers. The paper will be evaluated on how clearly and concisely it motivates and describes its topic.

Social and Ethical Issues

There is no significant social or ethical content.

Theoretical Content

Topic	Hours
Syntax and semantics.	2
Lexical and syntax analysis.	3

Problem Analysis

The main areas where students are involved in problem analysis are:

- Formal language description and translation.
- Abstractions for programming “in the large.”
- Evaluation of programming languages in terms of their suitability for specific tasks.

Solution Design

A variety of tools, techniques, and language features are presented for solving problems in the arena of program language design, such as:

- Recursive descent and shift-reduce parsing techniques for LALR(1) languages.
- Program abstraction techniques for writing and maintaining large programs.
- Reference counting and automatic garbage collection for robust memory management.
- Robust and elegant error handling techniques via exception handling.

CC2001

This course provides coverage of topics in the following areas (hours listed are minimums):

AR2. Machine level representation of data [core]	1
AR3. Assembly level machine organization [core]	1
OS12. Scripting [elective]	1
PL1. Overview of programming languages [core]	6
PL2. Virtual machines [core]	1
PL3. Introduction to language translation [core]	2
PL4. Declarations and types [core]	3
PL5. Abstraction mechanisms [core]	3
PL6. Object-oriented programming [core]	4
PL7. Functional programming [elective]	3
PL8. Language translation systems [elective]	3
PL9. Type systems [elective]	4
PL10. Programming language semantics [elective]	3
PL11. Programming language design [elective]	6

Course Coordinator: Wayne O. Cochran
Last Updated: November 22, 2005 (Approved)
Syllabus Version Number: 1.2