1 About this Course
   • Goals
   • Syllabus

2 Languages and Translation
   • General Purpose Languages
   • Translators

3 Language Standards
   • What is a Standard?
   • Confusion about Standards
   • The Standardization Process

4 Homework
1. About this Course

Website: http://eliza.newhaven.edu/lang/

Course goals

Textbook, Grades, and Work
The course goals are to learn about languages...

- How to approach a new language and master it quickly, including downloading and installation.
- All languages are alike at the core, no matter what disguise they are in.
  - A language definition supplies lexical form, syntax, and semantics.
  - All support a minimal core of control structures.
  - These are processed by a compiler or an interpreter.
  - Every language also has libraries, style, and intended usage.
- Languages can be compiled, based on bytecode, or interpreted. We will learn three at an introductory level.
  - FORTH is based on a very simple virtual machine, so simple that we can easily decompile programs.
  - Scheme is a functional language: it uses binding, not assignment.
  - Javascript is a scripting language intended for web applications.
- We will also cover many things about a variety of other languages.
The course goals are to learn:

- Many facts about specific languages and about languages in general.
- Mathematical Formalisms: Finite state machines, regular languages, context-free languages.
- How compilers work.
- How interpreters work.
- Language design issues.
- Javascript.
- Scheme.
- Forth.
And to learn about translation:

- The stages of translation and the technology used at each stage.
- The structures that are used to implement compilers and interpreters.
- The benefits and drawbacks of various language designs, including late binding, byte code interpreters, and garbage collection.
- How types are implemented.
- How objects, templates, and polymorphism are implemented.

Correct technical terminology and expression are important. I hope you will develop the ability to write clear, concise, correct, and precise explanations of the abstractions listed above.
The Textbook


- This book is out-of-print, but used copies are often available on the web.
- Use the "Chapters from the text" link, on the website to access PDF copies of the chapters.

Why use such an old book?

- It is free.
- The field is constantly changing, books become obsolete very fast.
- It is hard to find a book for this course that students can read!
- This is a book you can read that explains complicated things in straightforward language.
Grades.

These are approximate grades that assume your exams are at least as good as your homework.

- Do all the work = A+
- Do some of every type well and do all of some type well = A
- Do some of every type of work, but none of the difficult projects = B
- Lack a clear understanding of many of the issues = C
- Don’t turn in anything = ‘ : (  

If the exams are worse, they will bring down the homework grade.
Homework and Labs

I will assign weekly homework.

- Almost every week there will be a page or two of written homework.
- Often a lab will be included. A “lab” is an exercise for which you need to use a computer and a language translator. Labs include
  - Introductory exercises in 3 languages.
  - Small programs in these languages.
  - One or two longer programs that implement parts of compiler theory.
Final Project

There will be no final exam. We will use the last day for group or individual project presentations. Choose a project that is interesting for you, will teach you something, will entertain the class productively, and is not too large or difficult.

Suggestions:

- Choose a language, X, that you do not yet know. Find out who uses X and why. Explore its properties and compare it to a well-known such as C or Java or C++. Present the circle program in X.

- Choose a language feature and compare its implementation across several languages, being sure to include features or languages that were not covered extensively in class.

- Choose a language that you know slightly. Complete a relatively sophisticated project in it.

- Or... tell me your idea.

Proposals (a paragraph or two) are due on March 1.
Weekly Written Homework: Clarity

I will be assigning written homework regularly. Your answers should be brief, clear and precise. (Less than 1 typewritten page per assignment).

Example: define “dog”.
- First effort: A 4-legged animal with a tail that barks. (not bad)
- More precise: A 4-legged animal that barks and has a tail. It is of the family canine and typically is domesticated.

Another example: define a CS 6636 student.
- First effort: Someone who is attending the CS 6636 class.
- More precise: A graduate student who is registered for CS 6636 and is attending class regularly.

Answers that are unclear, too short, too shallow, or imprecise will not get full credit. Your goal is to convince me that you understand the technical material.
Weekly Written Homework: Too little detail

These answers are much too general and much too brief:

1. Here is a correct but foggy statement about Java

   *Java uses beans.*

   Baked beans? Coffee beans? How are they used?

2. Here is a correct but foggy statement about translation

   *A lexer finds the words in a program.*

   What are *words* in this context? When does this happen? How does it relate to the other phases of translation?
Weekly Written Homework: Terminology

Use the correct technical terminology, and use it appropriately. Use standard English. Slang and twitter-talk are not acceptable.

Example from an end-of-term student evaluation:

*Dr. Eggert is a mad freeze.*

I am sure from the context that this was not intended as an insult. I doubt this means that he is angry, or a cold, unfriendly person. It might mean that he is “crazy, man, crazy” and “cool”.

My dictionary fails to define these words in a way that makes sense in this context. An answer that uses undefined words, or uses words incorrectly will be marked “wrong word”.

Here is another statement that uses a wrong word. Can you explain the problem?

*The Python compiler checks the types of objects at run time.*
Weekly Written Homework: Showing Comprehension

You need to show me that you understand the material.

Write enough, in your own words, to clearly communicate your understanding.

Don’t write too much. Show me that you know what is important by not writing everything you can think of.

Do not give me answers that are copied verbatim from the lecture notes, the book or the internet.

An answer that appears too good for your level of development will be marked copied. Such answers generally show that the student does not understand the material at all, but can look up keywords on the internet.
2. What is a Programming Language?

General Purpose Languages
Translators
General Purpose Languages

We will be learning about general purpose programming languages. This excludes some kinds of computer coding systems:

- HTML is not a language. It lacks the minimal control structures necessary to be a language.
- Microsoft Excel has macros, but it is not a language in our sense. Could you implement an operating system in Excel?
- SQL is a language but does not stand alone. It must be embedded in a database engine. It is not general purpose; would you implement a compiler in SQL?
- LaTeX is the typesetting language I use for all my work. It has macros, conditionals and definable environments (subroutines) and extensive libraries (math, beamer) but it is not a general purpose language – it is useful only for typesetting.
Languages have much in common.

Programming languages are all alike at some level. Otherwise, we would not call them “programming languages”.

- All let you write formulas for computations.
- All permit you to attach symbolic names to data and computational results.
- All permit definition of stored procedures.
- All support conditionals, repetition.
- All support some way to get data into and out of the program.

But Lisp and Java look as different as night and day.

Ruby looks more like Lisp, and Python looks more like Java. Does this appearance carry over into something more than skin deep?
Languages have many differences.

Each language arises from a perceived need. It might be

- The research project of a graduate student or a professor attempting to give language an important new capability. (Lisp, MAD, Scheme, Modula, Smalltalk, Miranda, ML, Haskell, . . .)
- The tool of a teacher, hoping to provide a simple, complete, teachable way to write programs. (BASIC, Pascal, Alice, . . .)
- By accretion and evolution from a precursor (Perl, Bash, . . .)
- From a company with a profitable idea (Wolfram/Mathematica, Sun/Java, Adobe/Postscript)
- To meet a need that other languages don’t. (COBOL, FORTRAN, C, LISP, Scheme, Ada, C++, . . .)
Assembly Languages

Every family of computers has its own machine language that reflects the hardware architecture of the machine. Corresponding to that are one or more symbolic assembly languages.

- **Input:** A source program with symbolic names for variables and functions, but basically one line of code for each machine operation in the executable version. No type definitions or type checking.
- **Output:** Object code, ready to be linked with libraries to produce an executable module.
- **Environment:** Executable modules run directly on top of the OS.
- **Portability:** Programs can only be used on the same OS and hardware architecture for which they were written.

The kernel of an OS, or part of it, must be written in assembly code.
Fully Compiled Languages

Compilers type-check the code at compile time and produce the most efficient and most stable executables.

- **Input**: A program is written in a symbolic language that supports expressions, function calls, control statements, and possibly classes.
- **The compiler’s output is object code, ready to be linked with libraries and run.**
- **Environment**: Programs run directly on top of the OS.
- **Portability**: Programs can be used on many different machines. However, they must be recompiled for each combination of hardware and operating system, and the resulting executable programs may not operate identically on all platforms.

A compiler can be written for any higher-level language that requires full definition of the properties of objects at compile time.
Byte-code Languages

Java and FORTH are byte-code-languages. A byte-code language has both a compiler and an interpreter.

- **Input:** A program is written in a symbolic language that supports expressions, function calls, control statements, possibly classes . . .
- **The compiler’s output is byte code:** a series of instructions in the “machine language” of a virtual machine (VM).
- **Environment:** The compiled programs run only within the virtual machine, which in turn runs on top of the OS. the VM implements each byte code by executing one or more machine instructions.
- **Portability:** Pre-compiled byte-code programs can be used on any OS that supports an appropriate virtual machine. The results will be the same on all platforms.

Most byte-code languages type-check the code at compile time and produce relatively stable code.
Interpreted Languages: Definition

- **Environment**: All programming, editing, and execution are done within the language’s own environment.
- **Input**: the programmer enters functions, symbol definitions, and expressions to be evaluated.
- **Output**: The language system evaluates expressions and prints the results.
- **Intermediate form**: Function and object definitions are stored in the current “workspace”, in an internal form that is equivalent to the input except that initial parsing has been completed and comments and unnecessary spaces have been removed. A collection of these elements is called a script.
Interpreted Languages: Execution

- The language system carries out the machine instructions that correspond to each statement in the program.
- Type-checking is done, but only for primitive functions and types.
- Portability: Source code programs can be used anywhere the language has been installed. The results on different platforms may vary.

Interpreters were used initially to define Python, LISP, APL, and BASIC.

Later, compilers were developed for Python and LISP that produced executable native code.
3. Language Standards

Confusion about Standards
What is a Standard?
The Standardization Process
Some Untrue Statements.

- EXCEL is the standard spreadsheet in the US.
- `<stdafx>` is a standard include file for C++.
- All the common compilers implement the relevant standard.
- All standard compilers make executables that give the same answers.
- If my compiler accepts a program, it is written according to the standard.
- Gnu C (gcc) is nonstandard because it provides warning comments for format errors.
Why do we need language standards?

A standard defines the syntax and semantics of the minimal language. Extensions may be added. Deviations are not allowed.

We need a standard so that . . .

- All compiler writers implement the same syntax and semantics.
- Programs are portable from machine to machine.
- Programmers can move from one job to another and still be effective.
- We can effectively teach students how to program.
- Student homework projects will run properly on the instructor’s machine.
What questions does a language standard answer?

Each layer of the language needs to be defined.

- Exactly what keywords and identifiers are allowed?
- Is it case sensitive?
- What is the syntax for statements, comments, control structures, blocks, functions.
- What meaning is ascribed to each syntax? Function calls? Loops? Expressions? Literals?
- What are the type rules, if any?
- What are the primitive data types?
- What are the primitive operators and function libraries?
- What language extension mechanisms are supported?
- Is there a virtual machine?
How do we define a language?

Old way: whatever the compiler does defines the language, with inevitable errors, gotchas, and omissions.

New way: a formal definition, including...
- Precise descriptive text, with supporting theory, where it applies.
- Mathematical formalisms to define lexical rules and syntax.
- Formal semantic rules (for functional languages only).

Formal definitions of semantics were attempted for PL/1 but failed miserably because the result was too complex to be useful. They cannot capture typing rules with any reasonable simplicity.
What is a language standard?

- A language standard provides a complete and unambiguous definition of the language. You can tell whether something conforms or not.
- Included in the standard are the lexical rules, syntax, semantics, and requirements of the type system.
- It defines the syntax and correct semantics of all required language elements. It does not define any particular implementation.
- A language element can be required by the syntax but not only partially defined by the semantics. Such semantics are said to be “undefined”. (Example: In C, the size of an int.)
- A standard implementation is permitted to do more than the standard requires, but not less than or something different from the standard requirements.

See: FORTH Standard
In the beginning, a language is born.

- An individual, a company, or a group might invent a language.
- A language acquires a community of users.
- The users work in different environments and their implementations may differ here and there. Such differences cause non-portability.
- If the language ever enters common use, the global community may become interested in creating a shared, standard definition.
- A language standard provides a basis for creating compatible implementations of the language.
- It is the joint work of many of the best respected and most knowledgable people in the field.
- Language implementors who ignore the standard do great harm to the eventual growth and usage of the language.
The Standardization Process

When a standard is needed...

- A committee is formed, announced and convened by ANSI.
- The committee considers the original informal language definition, common use, recent proposed additions, design errors, murky semantics, and problem spots.
- The committee agrees upon what the standard language will be.
- A formal definition is written and posted for public comment.
- After a year or so, the committee will consider the comments, make necessary changes, and possibly resubmit the proposal for comment.
- Eventually, the modified proposed standard becomes final.
- After ANSI makes a standard, it goes to ISO for international consideration and further tweaks.
Language Growth and Change

After standardization,

- Languages never sit still. People keep inventing new, useful, features.
- Implementations drift apart as different features are added.
- People discover important omissions and defects in the standard.
- New language extensions and libraries are added when new application areas develop.
- After several years, a new standards committee is convened to consider all these changes and fix the problems.

When a company invents a language (Sun/Java), it might keep exclusive ownership of the language definition rather than submit it to a standards process. In this case, changes are made internally and announced.
Due on January 30

- Read Chapters 1 and 2 of the text.
- HW1: Written Exercises (see below)
Hw 1: Language Standards

1. Explain why each of these statements is false. Write 1 or 2 sentences for each answer.

1. EXCEL is the standard spreadsheet in the US.
2. `<stdafx>` is a standard include file for C++.
3. All the common compilers implement the relevant standard.
4. All standard compilers make executables that give the same answers.
5. If my compiler accepts a program, it is written according to the standard.
6. Gnu C (gcc) is nonstandard because it provides warning comments for format errors.
2. HTML is not a general purpose programming language because too many things are missing. List several properties that HTML lacks that are present in a general purpose programming language.

3. Choose a language that you have used. Does it have a standard? If so, give the date of the current standard and the most recent earlier standard. If not, who controls the language? Cite your references.

4. Give a complete and precise definition, in your own words, of the meaning of “undefined” in the C standard. Be careful – the meaning is surprising. Use your textbook for reference.