A language that doesn't affect the way you think about programming, is not worth knowing.

- Alan Perlis

CSE341 Programming Languages

Gebze Technical University Computer Engineering Department
2023-2024 Fall Semester
Introduction

1

Programming Languages

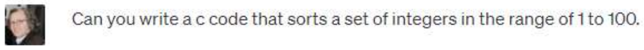
2

Let’s Program

Write a function that sorts a given set of numbers within the range of [1,100]....

3

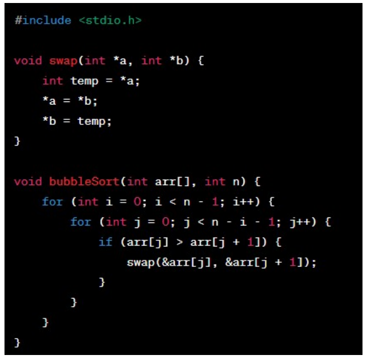
Let’s Program



Can you write a c code that sorts a set of integers in the range of 1 to 100.

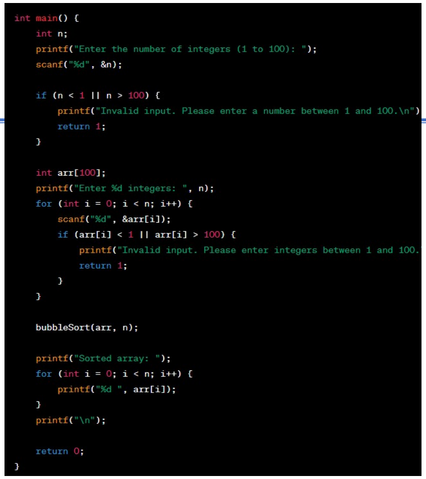
4

Let’s Program



```c
#include <stdio.h>

void swap(int *a, int *b) {
    int temp = *a;
    *a = *b;
    *b = temp;
}

void bubbleSort(int arr[], int n) {
    for (int i = 0; i < n - 1; i++) {
        for (int j = 0; j < n - i - 1; j++) {
            if (arr[j] > arr[j + 1]) {
                swap(&arr[j], &arr[j + 1]);
            }
        }
    }
}
```

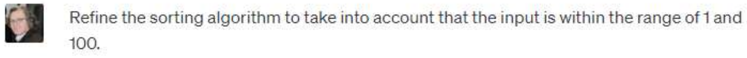
5



```c
int main() {
    int n;
    printf("Enter the number of integers (1 to 100): ");
    scanf("%d", &n);

    if (n < 1 || n > 100) {
        printf("Invalid input. Please enter a number between 1 and 100.\n")
        return 1;
    }

    int arr[100];
    printf("Enter %d integers: ", n);
    for (int i = 0; i < n; i++) {
        scanf("%d", &arr[i]);
        if (arr[i] < 1 || arr[i] > 100) {
            printf("Invalid input. Please enter integers between 1 and 100.
            return 1;
        }
    }

    bubbleSort(arr, n);

    printf("Sorted array: ");
    for (int i = 0; i < n; i++) {
        printf("%d ", arr[i]);
    }
    printf("\n");

    return 0;
}
```

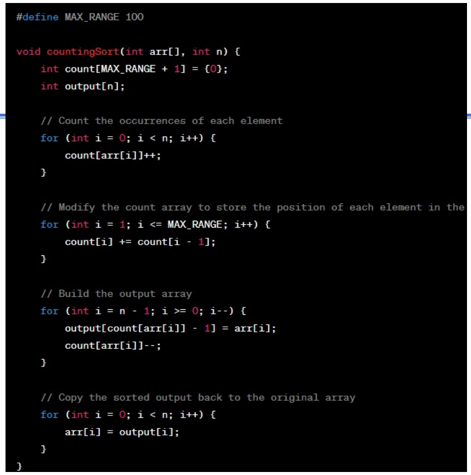
6

Let’s Program



Refine the sorting algorithm to take into account that the input is within the range of 1 and 100.

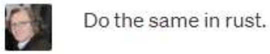
7



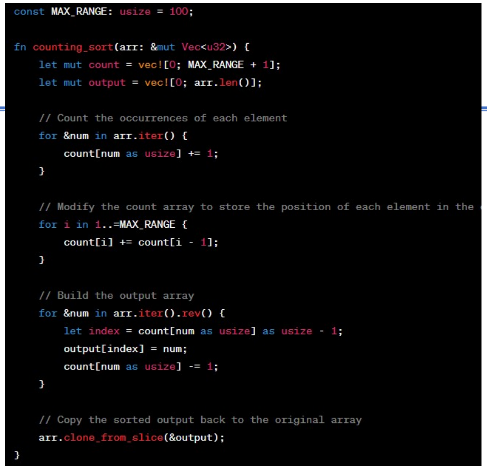
```c
#define MAX_RANGE 100

void countingSort(int arr[], int n) {
    int count[MAX_RANGE + 1] = {0};
    int output[n];

    // Count the occurrences of each element
    for (int i = 0; i < n; i++) {
        count[arr[i]]++;
    }

    // Modify the count array to store the position of each element in the
    for (int i = 1; i <= MAX_RANGE; i++) {
        count[i] += count[i - 1];
    }

    // Build the output array
    for (int i = n - 1; i >= 0; i--) {
        output[count[arr[i]] - 1] = arr[i];
        count[arr[i]]--;
    }

    // Copy the sorted output back to the original array
    for (int i = 0; i < n; i++) {
        arr[i] = output[i];
    }
}
```

8

Let’s Program



Do the same in rust.

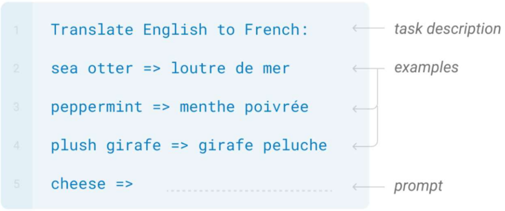


```rust
const MAX_RANGE: usize = 100;

fn counting_sort(arr: &mut Vec<u32>) {
    let mut count = vec![0; MAX_RANGE + 1];
    let mut output = vec![0; arr.len()];

    // Count the occurrences of each element
    for &num in arr.iter() {
        count[num as usize] += 1;
    }

    // Modify the count array to store the position of each element in the
    for i in 1..=MAX_RANGE {
        count[i] += count[i - 1];
    }

    // Build the output array
    for &num in arr.iter().rev() {
        let index = count[num as usize] as usize - 1;
        output[index] = num;
        count[num as usize] -= 1;
    }

    // Copy the sorted output back to the original array
    arr.clone_from_slice(&output);
}
```

Let’s Prompt

Few-shot

In addition to the task description, the model sees a few examples of the task. No gradient updates are performed.



```
Translate English to French:
sea otter => loutre de mer
peppermint => menthe poivrée
plush girafe => girafe peluche
cheese =>
```

Programming Languages

What's a programming language?

A programming language is an artificial language designed to communicate instructions to a machine, particularly a computer [Wikipedia].

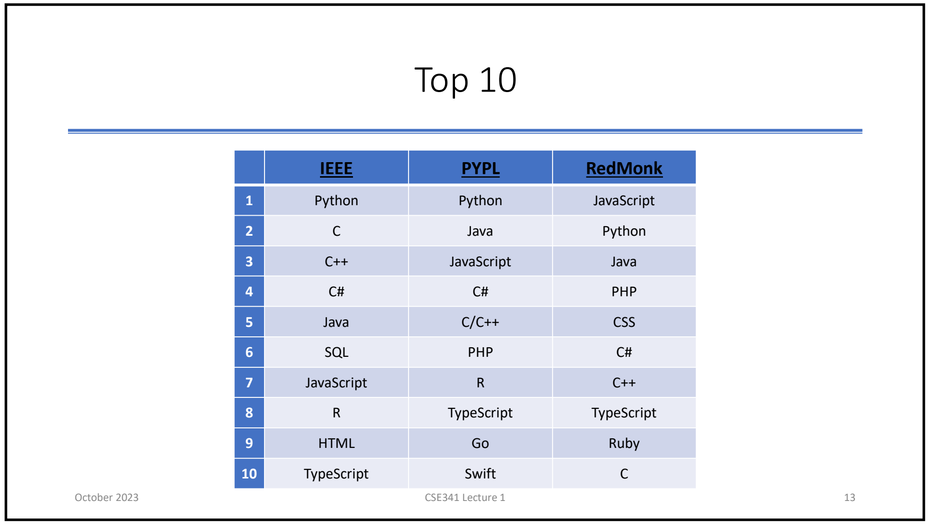
A language is a “conceptual universe” providing a framework for problem-solving and useful concepts and programming methods [Perlis].

How many programming languages are there?

Thousands!

Which one to use?

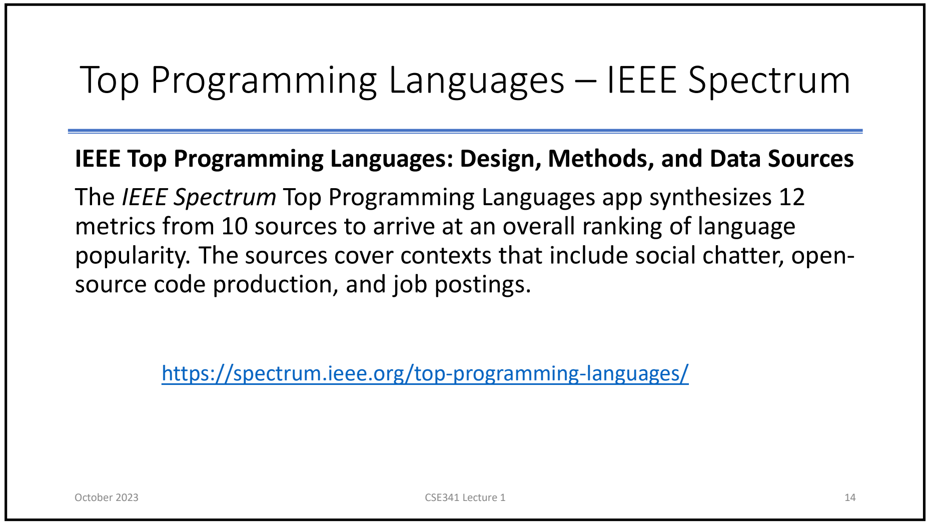
We’ll hopefully have a good idea after completing this course.



Top 10

	IEEE	PYPL	RedMonk
1	Python	Python	JavaScript
2	C	Java	Python
3	C++	JavaScript	Java
4	C#	C#	PHP
5	Java	C/C++	CSS
6	SQL	PHP	C#
7	JavaScript	R	C++
8	R	TypeScript	TypeScript
9	HTML	Go	Ruby
10	TypeScript	Swift	C

13



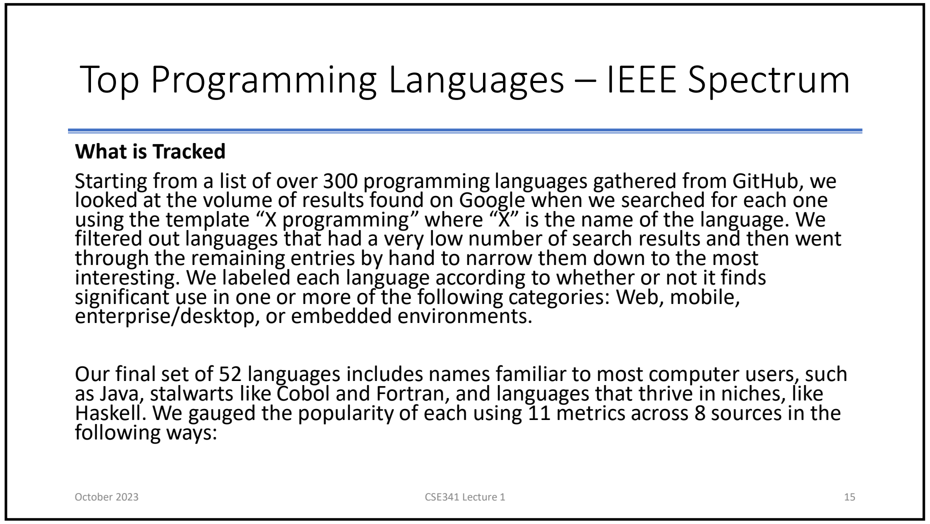
Top Programming Languages – IEEE Spectrum

IEEE Top Programming Languages: Design, Methods, and Data Sources

The *IEEE Spectrum* Top Programming Languages app synthesizes 12 metrics from 10 sources to arrive at an overall ranking of language popularity. The sources cover contexts that include social chatter, open-source code production, and job postings.

https://spectrum.ieee.org/top-programming-languages/

14



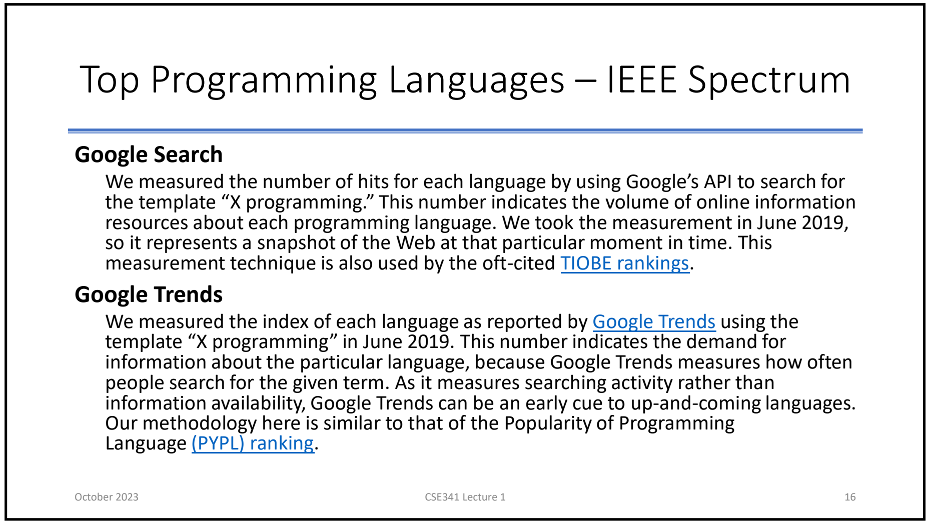
Top Programming Languages – IEEE Spectrum

What is Tracked

Starting from a list of over 300 programming languages gathered from GitHub, we looked at the volume of results found on Google when we searched for each one using the template “X programming” where “X” is the name of the language. We filtered out languages that had a very low number of search results and then went through the remaining entries by hand to narrow them down to the most interesting. We labeled each language according to whether or not it finds significant use in one or more of the following categories: Web, mobile, enterprise/desktop, or embedded environments.

Our final set of 52 languages includes names familiar to most computer users, such as Java, stalwarts like Cobol and Fortran, and languages that thrive in niches, like Haskell. We gauged the popularity of each using 11 metrics across 8 sources in the following ways:

15



Top Programming Languages – IEEE Spectrum

Google Search

We measured the number of hits for each language by using Google’s API to search for the template “X programming.” This number indicates the volume of online information resources about each programming language. We took the measurement in June 2019, so it represents a snapshot of the Web at that particular moment in time. This measurement technique is also used by the oft-cited TIOBE rankings.

Google Trends

We measured the index of each language as reported by Google Trends using the template “X programming” in June 2019. This number indicates the demand for information about the particular language, because Google Trends measures how often people search for the given term. As it measures searching activity rather than information availability, Google Trends can be an early cue to up-and-coming languages. Our methodology here is similar to that of the Popularity of Programming Language (PYPL) ranking.

16

Top Programming Languages – IEEE Spectrum

- **Twitter**

  We measured the number of hits on Twitter for the template "X programming" for the 12 months ending June 2019 using the Twitter Search API. This number indicates the amount of chatter on social media for the language and reflects the sharing of online resources like news articles or books, as well as physical social activities such as hackathons.

- **GitHub**

  GitHub is a site where programmers can collaboratively store repositories of code. Using the GitHub API and GitHub tags, we measured two things for the 12 months ending June 2019: (1) the number of new repositories created for each language, and (2) the number of active repositories for each language, where "active" means that someone has edited the code in a particular repository. The number of new repositories measures fresh activity around the language, whereas the number of active repositories measures the ongoing interest in developing each language.

- **Stack Overflow**

  Stack Overflow is a popular site where programmers can ask questions about coding. We measured the number of questions posted that mention each language for the 12 months ending June 2019. Each question is tagged with the languages under discussion, and these tags are used to tabulate our measurements using the Stack Exchange API.

- **Reddit**

  Reddit is a news and information site where users post links and comments. On Reddit we measured the number of posts mentioning each of the languages, using the template "X programming" from June 2018 to June 2019 across any subreddit on the site. We collected data using the Reddit API.

Top Programming Languages – IEEE Spectrum

- **Hacker News**

  Hacker News is a news and information site where users post comments and links to news about technology. We measured the number of posts that mentioned each of the languages using the template "X programming" for the 12 months ending June 2019. Just like those used by the websites Topsy, Stack Overflow, and Reddit, this metric also captures social activity and information sharing around the various languages. We used the Algolia Search API.

- **CareerBuilder**

  We measured the demand for different programming languages on the CareerBuilder job site. We measure the number of fresh job openings (those that are less than 30 days old) on the U.S. site that mention the language. Because some of the languages we track could be ambiguous in plain text—such as D, Go, J, Processing, and R—we use strict matching of the form "X programming" for these languages. For other languages we use a search string composed of "X AND programming," which allows us to capture a broader range of relevant postings. We collected data in June 2019 using the CareerBuilder API.

- **IEEE Job Site**

  We measured the demand for different programming languages in job postings on the IEEE Job Site. Because some of the languages we track could be ambiguous in plain text—such as D, Go, J, Processing, and R—we use strict matching of the form "X programming" for these languages. For other languages we use a search string composed of "X AND programming," which allows us to capture a broader range of relevant postings. Because no externally exposed API exists for the IEEE Job Site, data was extracted using an internal custom-query tool in June 2019.

- **IEEE Xplore Digital Library**

  IEEE maintains a digital library with over 3.6 million conference and journal articles covering a range of scientific and engineering disciplines. We measured the number of articles that mention each of the languages in the template "X programming" for the years 2018 and 2019. This metric captures the prevalence of the different programming languages as used and referenced in scholarship. We collected data using the IEEE Xplore API.

Top Programming Languages – TIOBE

TIOBE Index for September 2022

September Headline: Julia is getting close to the TIOBE index top 20

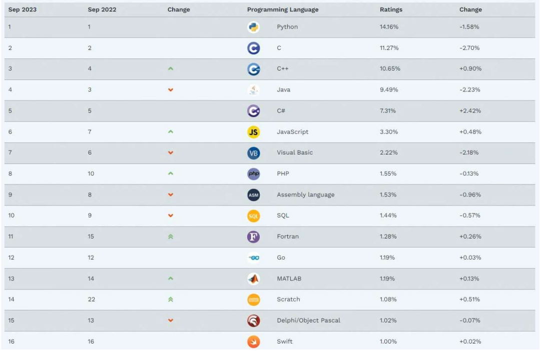
The Julia programming language is only 0.05% away from a top 20 position. Julia is designed for numerical analysis and computational science. There are many competing languages in that field. So what makes Julia stand out? Julia beats Matlab because it is much more modern and it can be used free of charge. Furthermore, Julia beats Python and R because it is much faster. Since there is a huge demand in the number crunching and modeling field, Julia has a serious chance to enter the top 20 in the near future. Note that the language Rust has also been knocking on the top 20 door for quite some time, but did not succeed so far. Time will tell whether Julia will endure the same fate. -- Paul Jansen CEO TIOBE Software

The TIOBE Programming Community index is an indicator of the popularity of programming languages. The index is updated once a month. The ratings are based on the number of skilled engineers world-wide, courses and third party vendors. Popular search engines such as Google, Bing, Yahoo!, Wikipedia, Amazon, YouTube and Baidu are used to calculate the ratings. It is important to note that the TIOBE index is not about the best programming language or the language in which most lines of code have been written.

The index can be used to check whether your programming skills are still up to date or to make a strategic decision about what programming language should be adopted when starting to build a new software system. The definition of the TIOBE index can be found here.

https://www.tiobe.com/tiobe-index/

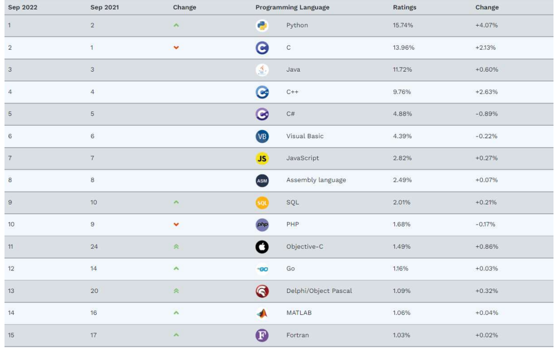
Top Programming Languages – TIOBE



Sep 2023	Sep 2022	Change	Programming Language	Ratings	Change
1	1		Python	14.16%	-1.58%
2	2		C	11.27%	-2.70%
3	4	▲	C++	10.65%	+0.90%
4	3	▼	Java	9.49%	-2.23%
5	5		C#	7.31%	+2.42%
6	7	▲	JavaScript	3.30%	+0.48%
7	6	▼	Visual Basic	2.22%	-2.18%
8	10	▲	PHP	1.55%	-0.13%
9	8	▼	Assembly language	1.53%	-0.96%
10	9	▼	SQL	1.44%	-0.57%
11	15	▲▲	Fortran	1.28%	+0.26%
12	12		Go	1.19%	+0.03%
13	14	▲	MATLAB	1.19%	+0.13%
14	22	▲▲	Scratch	1.08%	+0.51%
15	13	▼	Delphi/Object Pascal	1.02%	-0.07%
16	16		Swift	1.00%	+0.02%

TIOBE Programming Community Index for September 2023

Top Programming Languages – TIOBE

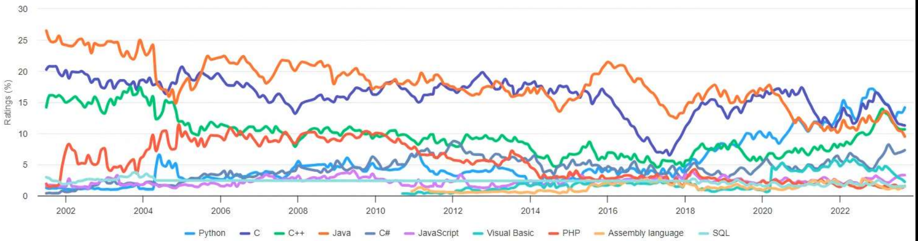


Sep 2022	Sep 2021	Change	Programming Language	Ratings	Change
1	2	^	Python	15.74%	+4.07%
2	1	⌄	C	13.96%	+2.13%
3	3		Java	11.72%	+0.60%
4	4		C++	9.76%	+2.63%
5	5		C#	4.88%	-0.89%
6	6		Visual Basic	4.39%	-0.22%
7	7		JavaScript	2.82%	+0.27%
8	8		Assembly language	2.49%	+0.07%
9	10	^	SQL	2.01%	+0.21%
10	9	⌄	PHP	1.68%	-0.17%
11	24	⏫	Objective-C	1.49%	+0.86%
12	14	^	Go	1.16%	+0.03%
13	20	⏫	Delphi/Object Pascal	1.09%	+0.32%
14	16	^	MATLAB	1.06%	+0.04%
15	17	^	Fortran	1.03%	+0.02%

TIOBE Programming Community Index for September 2022

21

Top Programming Languages – TIOBE



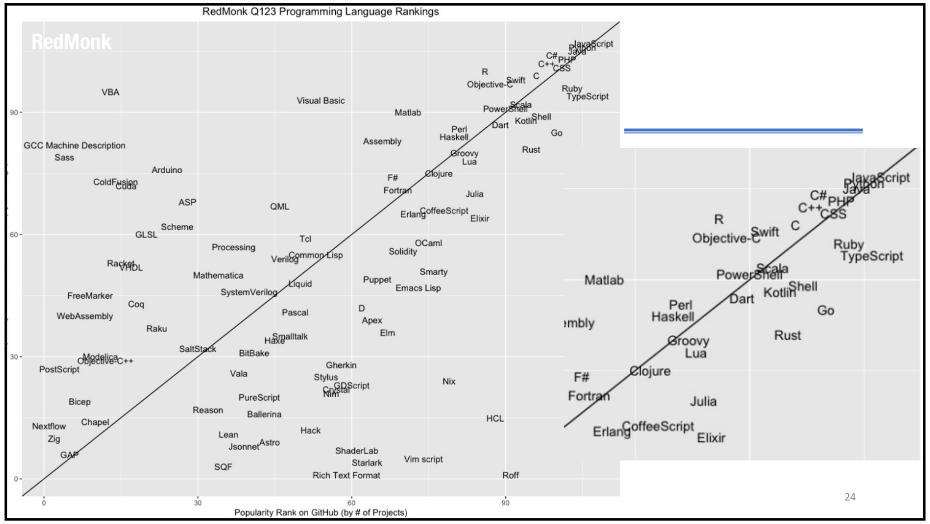
TIOBE Programming Community Index for September 2023

22

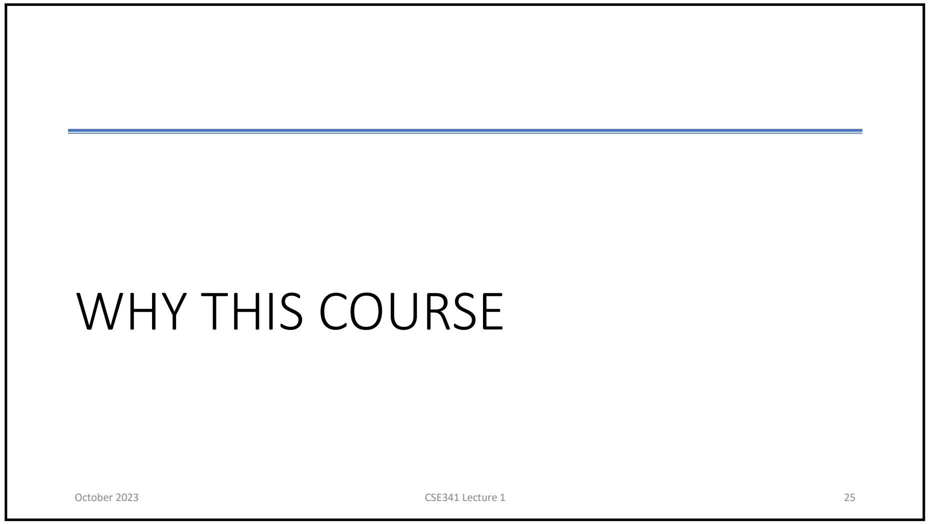
Programming Languages – Popularity

https://redmonk.com/sogrady/2023/05/16/language-rankings-1-23/

23

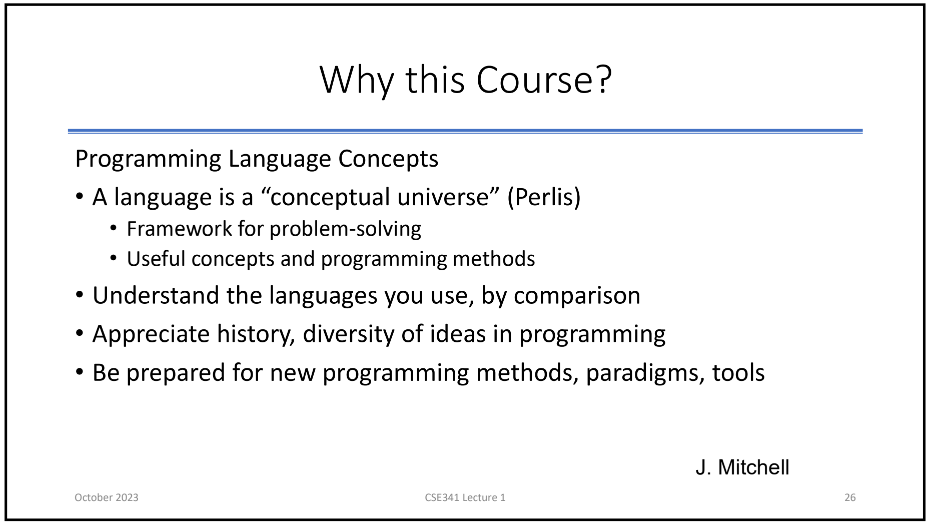

RedMonk Q123 Programming Language Rankings

24



WHY THIS COURSE

25



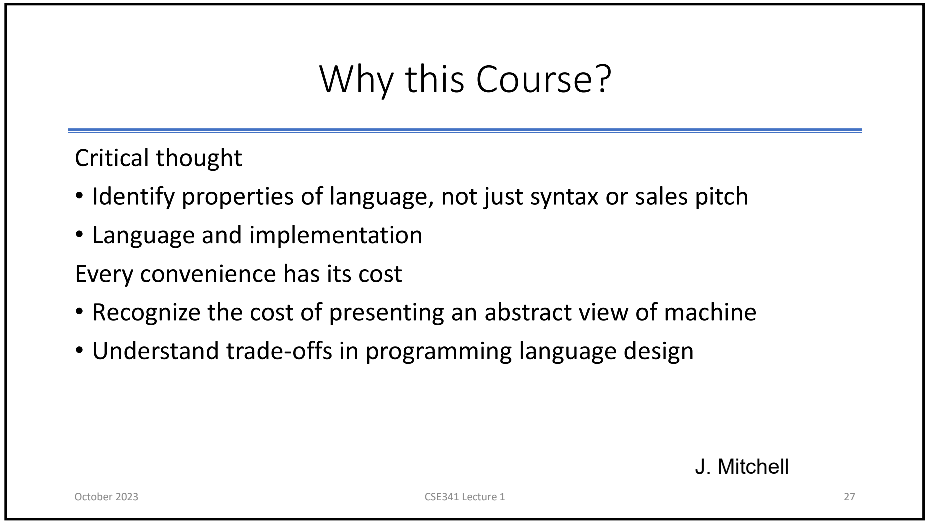
Why this Course?

Programming Language Concepts

- A language is a “conceptual universe” (Perlis)
  - Framework for problem-solving
  - Useful concepts and programming methods
- Understand the languages you use, by comparison
- Appreciate history, diversity of ideas in programming
- Be prepared for new programming methods, paradigms, tools

J. Mitchell

26



Why this Course?

Critical thought

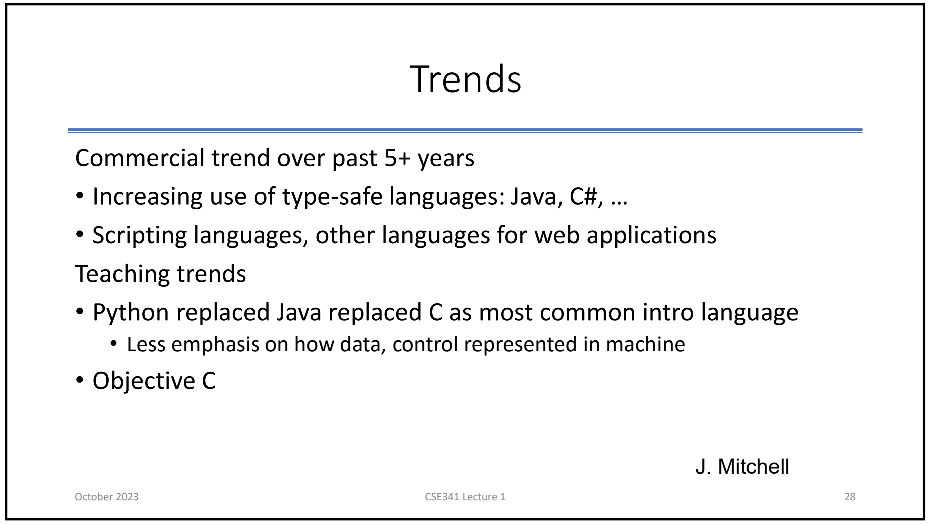
- Identify properties of language, not just syntax or sales pitch
- Language and implementation

Every convenience has its cost

- Recognize the cost of presenting an abstract view of machine
- Understand trade-offs in programming language design

J. Mitchell

27



Trends

Commercial trend over past 5+ years

- Increasing use of type-safe languages: Java, C#, …
- Scripting languages, other languages for web applications

Teaching trends

- Python replaced Java replaced C as most common intro language
  - Less emphasis on how data, control represented in machine
- Objective C

J. Mitchell

28

Trends

Research and development trends

- Modularity
  - Java, C++: standardization of new module features
- Program analysis
  - Automated error detection, programming env, compilation
- Isolation and security
  - Sandboxing, language-based security, …
- Web 2.0
  - Increasing client-side functionality, mashup isolation problems

J. Mitchell

Example

What is D?

D is the culmination of *decades of experience implementing compilers* for many diverse languages and has a unique set of features:

- *high level* constructs for great modeling power
- *high performance*, compiled language
- static typing
- direct interface to the operating system API's and hardware
- blazingly fast compile-times
- memory-safe subset (SafeD)
- *maintainable*, *easy to understand* code
- gradual learning curve (C-like syntax, similar to Java and others)
- compatible with C application binary interface
- limited compatibility with C++ application binary interface
- multi-paradigm (imperative, structured, object oriented, generic, functional programming purity, and even assembly)
- built-in error detection (contracts, unittests)
- … and many more features.

```
import std.stdio;
import std.algorithm;
import std.range;

void main()
{
   // Let's get going!
   writeln("Hello World!");

   // Take three arrays, and without allocating any new
   // memory, sort across all the arrays in-place
   int[] arr1 = [4, 9, 7];
   int[] arr2 = [5, 2, 1, 10];
   int[] arr3 = [6, 8, 3];
   sort(chain(arr1, arr2, arr3));
   writeln("%s\n%s\n%s\n", arr1, arr2, arr3);
}
```

https://tour.dlang.org/

Example

Major Design Goals of D

- Enable writing fast, effective code in a straightforward manner.
- Make it easier to write code that is portable from compiler to compiler, machine to machine, and operating system to operating system. Eliminate undefined and implementation defined behaviors as much as practical.
- Provide syntactic and semantic constructs that eliminate or at least reduce common mistakes. Reduce or even eliminate the need for third party static code checkers.

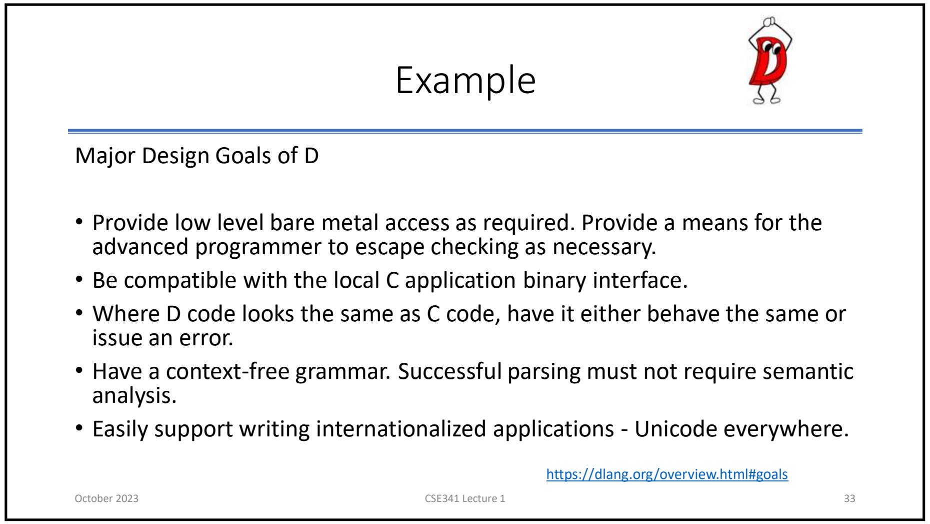
https://dlang.org/overview.html#goals

Example

Major Design Goals of D

- Support memory safe programming.
- Support multi-paradigm programming, i.e. at a minimum support imperative, structured, object oriented, generic and even functional programming paradigms.
- Make doing things the right way easier than the wrong way.
- Have a short learning curve for programmers comfortable with programming in C, C++ or Java.

https://dlang.org/overview.html#goals



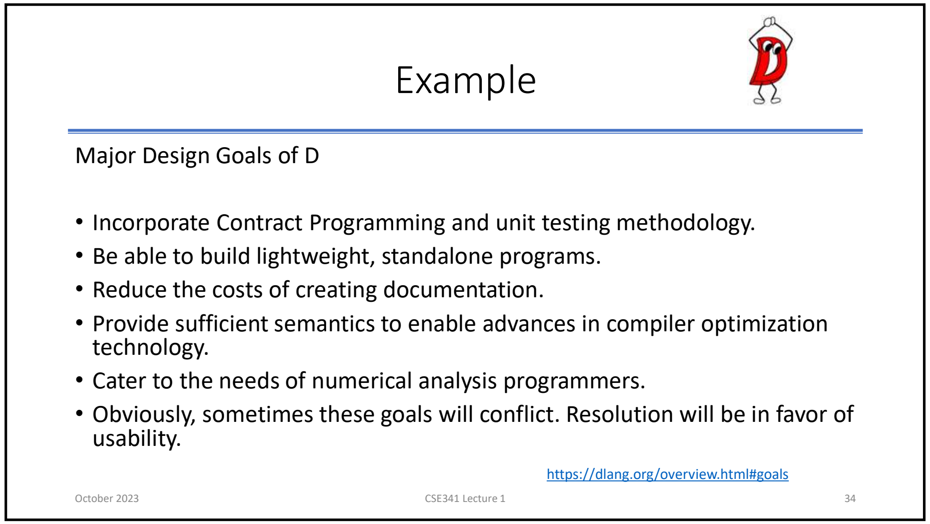
Example

Major Design Goals of D

- Provide low level bare metal access as required. Provide a means for the advanced programmer to escape checking as necessary.
- Be compatible with the local C application binary interface.
- Where D code looks the same as C code, have it either behave the same or issue an error.
- Have a context-free grammar. Successful parsing must not require semantic analysis.
- Easily support writing internationalized applications - Unicode everywhere.

https://dlang.org/overview.html#goals

33



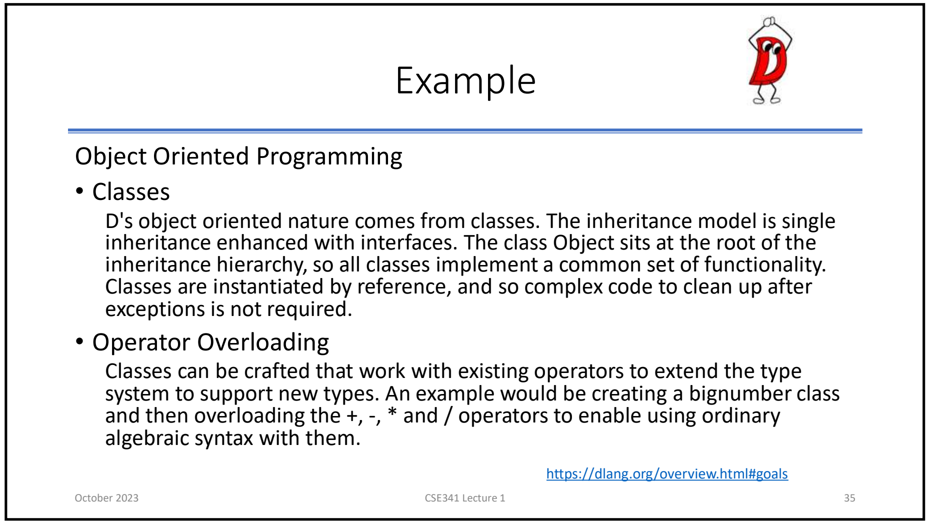
Example

Major Design Goals of D

- Incorporate Contract Programming and unit testing methodology.
- Be able to build lightweight, standalone programs.
- Reduce the costs of creating documentation.
- Provide sufficient semantics to enable advances in compiler optimization technology.
- Cater to the needs of numerical analysis programmers.
- Obviously, sometimes these goals will conflict. Resolution will be in favor of usability.

https://dlang.org/overview.html#goals

34



Example

Object Oriented Programming

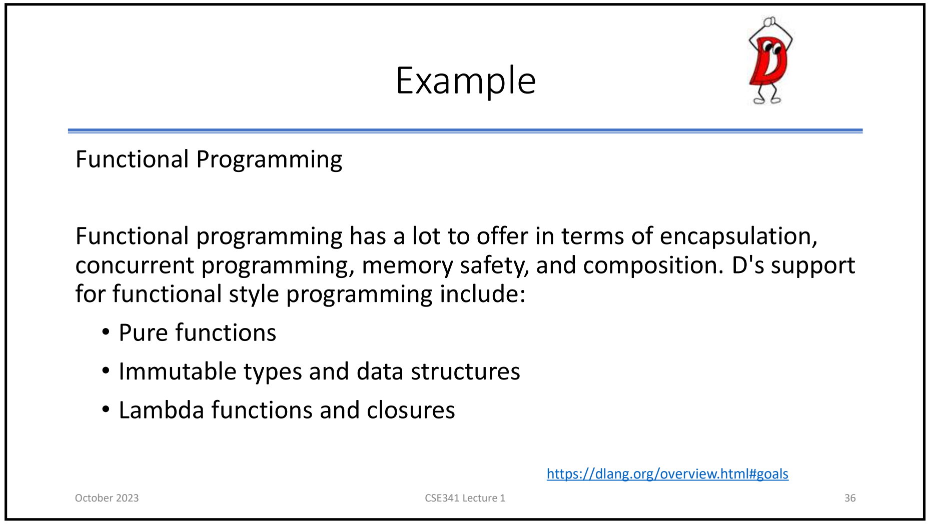
- Classes

  D's object oriented nature comes from classes. The inheritance model is single inheritance enhanced with interfaces. The class Object sits at the root of the inheritance hierarchy, so all classes implement a common set of functionality. Classes are instantiated by reference, and so complex code to clean up after exceptions is not required.
- Operator Overloading

  Classes can be crafted that work with existing operators to extend the type system to support new types. An example would be creating a bignumber class and then overloading the +, -, * and / operators to enable using ordinary algebraic syntax with them.

https://dlang.org/overview.html#goals

35



Example

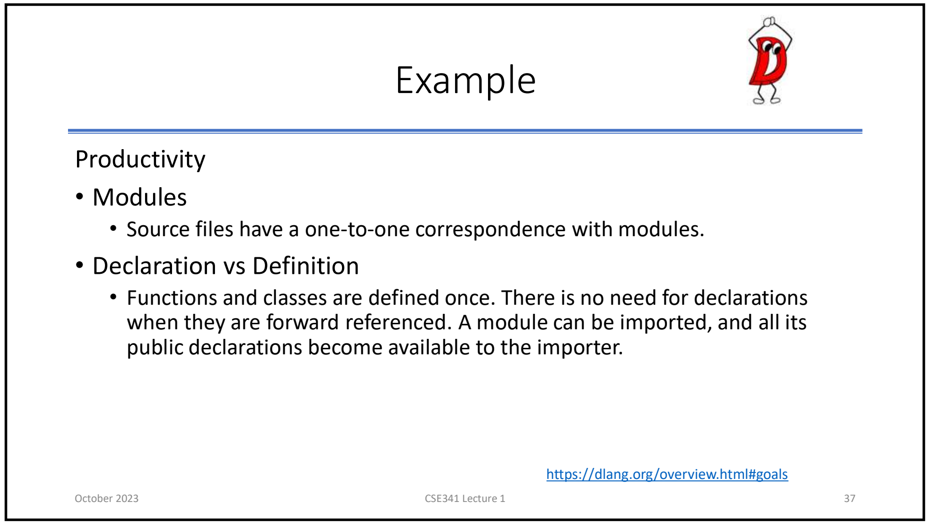
Functional Programming

Functional programming has a lot to offer in terms of encapsulation, concurrent programming, memory safety, and composition. D's support for functional style programming include:

- Pure functions
- Immutable types and data structures
- Lambda functions and closures

https://dlang.org/overview.html#goals

36



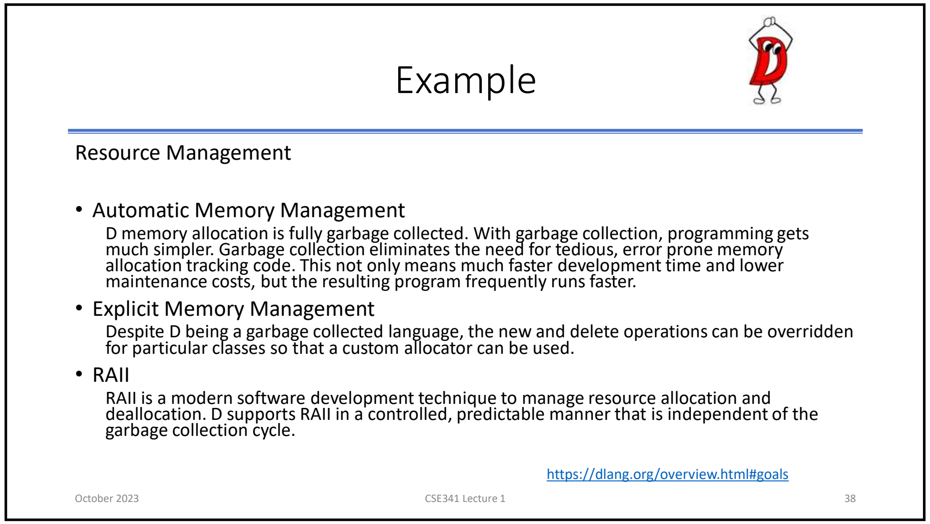
Example

Productivity

- Modules
  - Source files have a one-to-one correspondence with modules.
- Declaration vs Definition
  - Functions and classes are defined once. There is no need for declarations when they are forward referenced. A module can be imported, and all its public declarations become available to the importer.

https://dlang.org/overview.html#goals

37



Example

Resource Management

- Automatic Memory Management

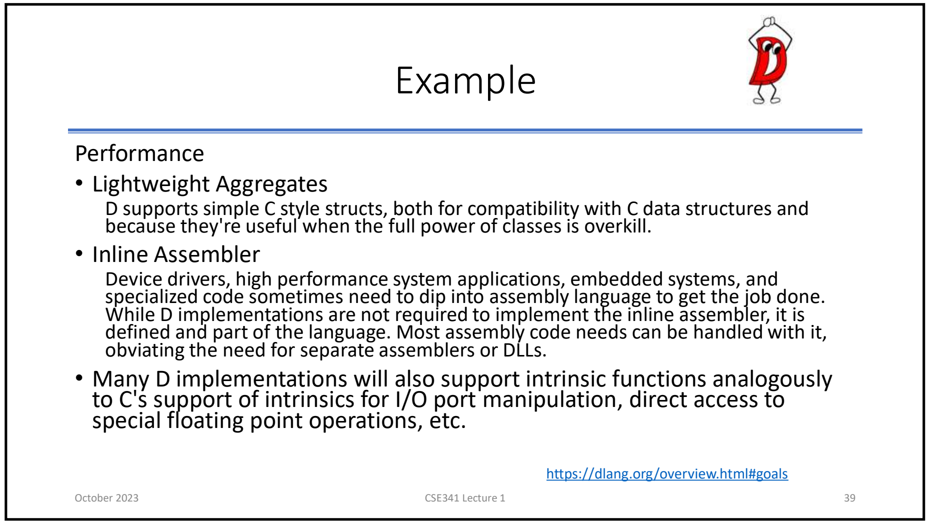
  D memory allocation is fully garbage collected. With garbage collection, programming gets much simpler. Garbage collection eliminates the need for tedious, error prone memory allocation tracking code. This not only means much faster development time and lower maintenance costs, but the resulting program frequently runs faster.
- Explicit Memory Management

  Despite D being a garbage collected language, the new and delete operations can be overridden for particular classes so that a custom allocator can be used.
- RAII

  RAII is a modern software development technique to manage resource allocation and deallocation. D supports RAII in a controlled, predictable manner that is independent of the garbage collection cycle.

https://dlang.org/overview.html#goals

38



Example

Performance

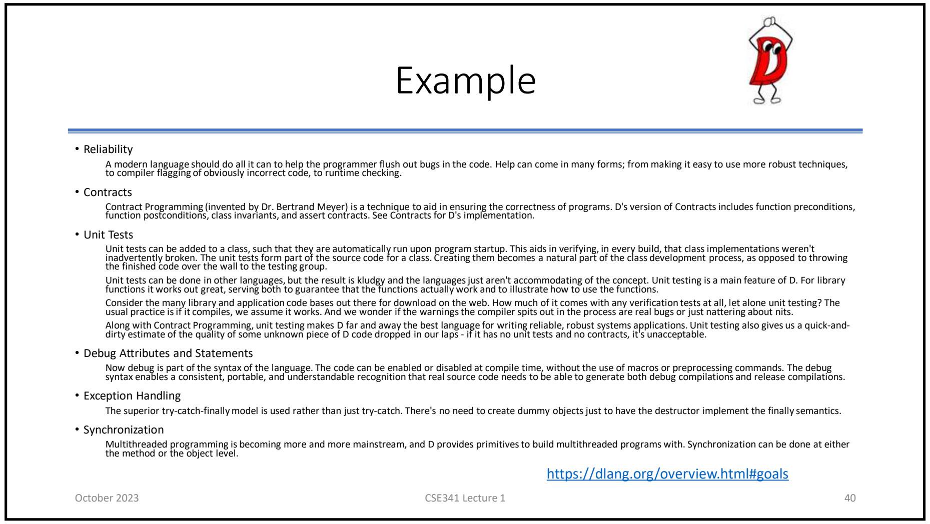
- Lightweight Aggregates

  D supports simple C style structs, both for compatibility with C data structures and because they're useful when the full power of classes is overkill.
- Inline Assembler

  Device drivers, high performance system applications, embedded systems, and specialized code sometimes need to dip into assembly language to get the job done. While D implementations are not required to implement the inline assembler, it is defined and part of the language. Most assembly code needs can be handled with it, obviating the need for separate assemblers or DLLs.
- Many D implementations will also support intrinsic functions analogously to C's support of intrinsics for I/O port manipulation, direct access to special floating point operations, etc.

https://dlang.org/overview.html#goals

39



Example

- Reliability

  A modern language should do all it can to help the programmer flush out bugs in the code. Help can come in many forms; from making it easy to use more robust techniques, to compiler flagging of obviously incorrect code, to runtime checking.
- Contracts

  Contract Programming (invented by Dr. Bertrand Meyer) is a technique to aid in ensuring the correctness of programs. D's version of Contracts includes function preconditions, function postconditions, class invariants, and assert contracts. See Contracts for D's implementation.
- Unit Tests

  Unit tests can be added to a class, such that they are automatically run upon program startup. This aids in verifying, in every build, that class implementations weren't inadvertently broken. The unit tests form part of the source code for a class. Creating them becomes a natural part of the class development process, as opposed to throwing the finished code over the wall to the testing group.

  Unit tests can be done in other languages, but the result is kludgy and the languages just aren't accommodating of the concept. Unit testing is a main feature of D. For library functions it works out great, serving both to guarantee that the functions actually work and to illustrate how to use the functions.

  Consider the many library and application code bases out there for download on the web. How much of it comes with any verification tests at all, let alone unit testing? The usual practice is if it compiles, we assume it works. And we wonder if the warnings the compiler spits out in the process are real bugs or just nattering about nits.

  Along with Contract Programming, unit testing makes D far and away the best language for writing reliable, robust systems applications. Unit testing also gives us a quick-and-dirty estimate of the quality of some unknown piece of D code dropped in our laps - if it has no unit tests and no contracts, it's unacceptable.
- Debug Attributes and Statements

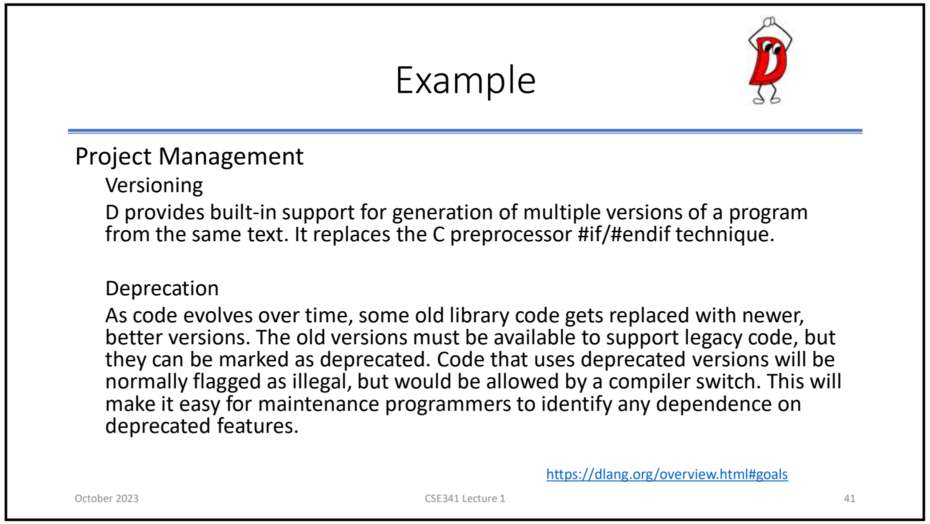
  Now debug is part of the syntax of the language. The code can be enabled or disabled at compile time, without the use of macros or preprocessing commands. The debug syntax enables a consistent, portable, and understandable recognition that real source code needs to be able to generate both debug compilations and release compilations.
- Exception Handling

  The superior try-catch-finally model is used rather than just try-catch. There's no need to create dummy objects just to have the destructor implement the finally semantics.
- Synchronization

  Multithreaded programming is becoming more and more mainstream, and D provides primitives to build multithreaded programs with. Synchronization can be done at either the method or the object level.

https://dlang.org/overview.html#goals

40



Example

Project Management

Versioning

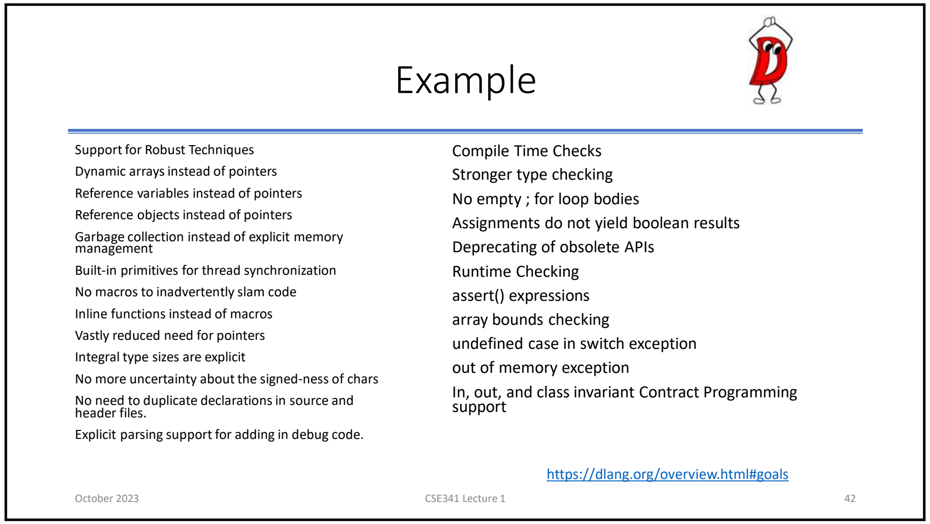
D provides built-in support for generation of multiple versions of a program from the same text. It replaces the C preprocessor #if/#endif technique.

Deprecation

As code evolves over time, some old library code gets replaced with newer, better versions. The old versions must be available to support legacy code, but they can be marked as deprecated. Code that uses deprecated versions will be normally flagged as illegal, but would be allowed by a compiler switch. This will make it easy for maintenance programmers to identify any dependence on deprecated features.

https://dlang.org/overview.html#goals

41



Example

Support for Robust Techniques

Dynamic arrays instead of pointers

Reference variables instead of pointers

Reference objects instead of pointers

Garbage collection instead of explicit memory management

Built-in primitives for thread synchronization

No macros to inadvertently slam code

Inline functions instead of macros

Vastly reduced need for pointers

Integral type sizes are explicit

No more uncertainty about the signed-ness of chars

No need to duplicate declarations in source and header files.

Explicit parsing support for adding in debug code.

Compile Time Checks

Stronger type checking

No empty ; for loop bodies

Assignments do not yield boolean results

Deprecating of obsolete APIs

Runtime Checking

assert() expressions

array bounds checking

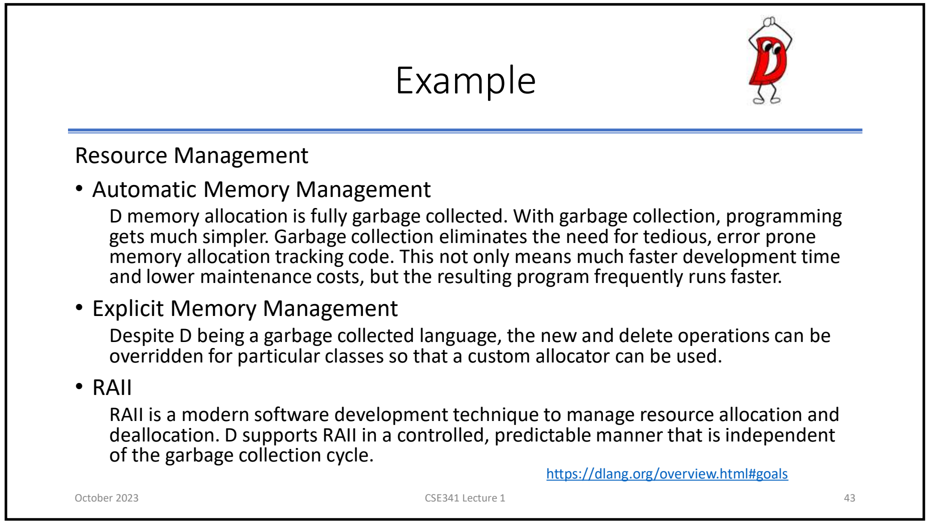
undefined case in switch exception

out of memory exception

In, out, and class invariant Contract Programming support

https://dlang.org/overview.html#goals

42



Example

Resource Management

- Automatic Memory Management

  D memory allocation is fully garbage collected. With garbage collection, programming gets much simpler. Garbage collection eliminates the need for tedious, error prone memory allocation tracking code. This not only means much faster development time and lower maintenance costs, but the resulting program frequently runs faster.

- Explicit Memory Management

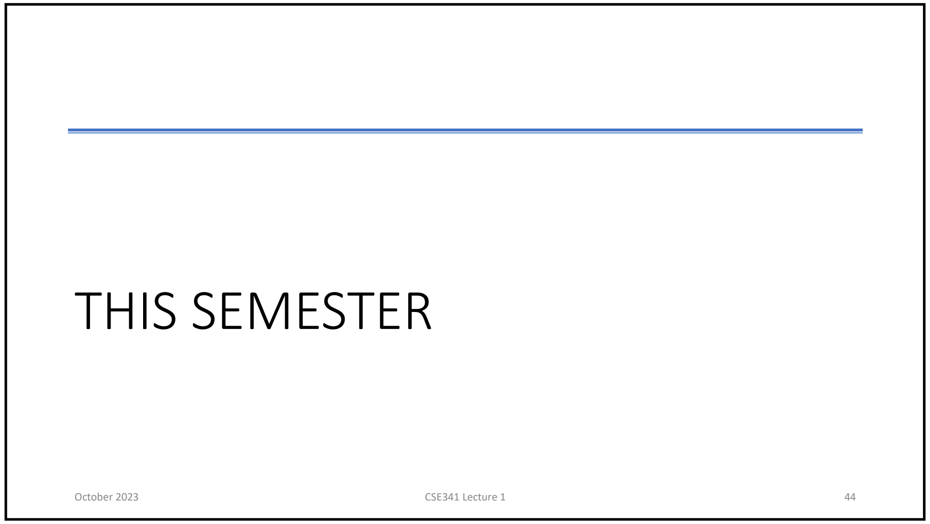
  Despite D being a garbage collected language, the new and delete operations can be overridden for particular classes so that a custom allocator can be used.

- RAII

  RAII is a modern software development technique to manage resource allocation and deallocation. D supports RAII in a controlled, predictable manner that is independent of the garbage collection cycle.

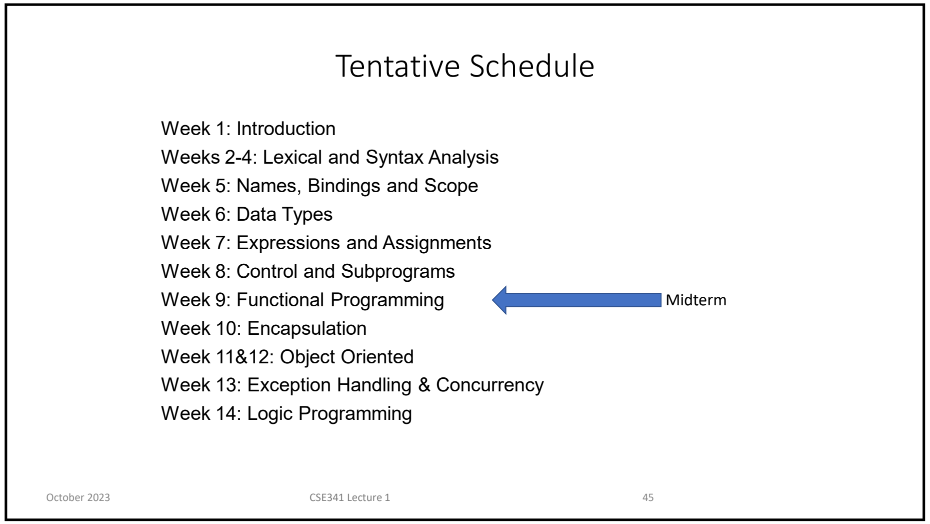
https://dlang.org/overview.html#goals

43



THIS SEMESTER

44



Tentative Schedule

Week 1: Introduction
Weeks 2-4: Lexical and Syntax Analysis
Week 5: Names, Bindings and Scope
Week 6: Data Types
Week 7: Expressions and Assignments
Week 8: Control and Subprograms
Week 9: Functional Programming ← Midterm
Week 10: Encapsulation
Week 11&12: Object Oriented
Week 13: Exception Handling & Concurrency
Week 14: Logic Programming

45

Thank you for listening!

46