EECS 440 System Design of a Search Engine

Winter 2021

Lecture 1: Introduction

Nicole Hamilton
https://web.eecs.umich.edu/~nham/
nham@umich.edu

Nicole Hamilton

nham@umich.edu
https://web.eecs.umich.edu/~nham/
Office: Beyster 2649
C: 425-765-9574

Office hours:
https://umich.zoom.us/j/2852894520
Mon-Thu 4:30 to 5:30 pm

Education

BS & MS EE, Stanford, 1973.

MBA, Boston University, 1987.

Background

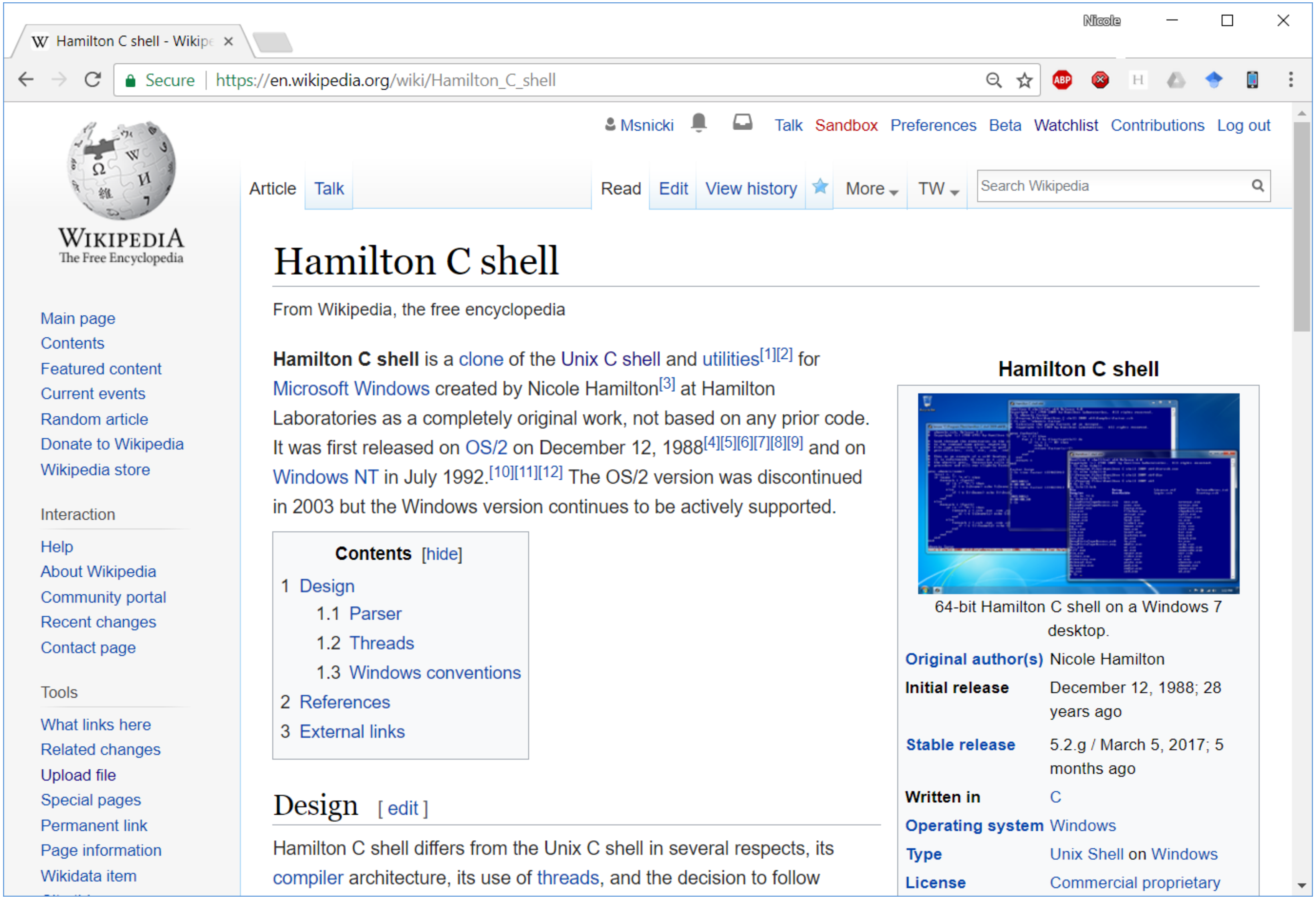
This is my fourth year at UM.

Started my career doing hardware design at IBM but quickly moved into software.

Spent most of my career as an entrepreneur selling a C shell I wrote for Windows.

When the dot-com collapse hit, I went to Microsoft, where I worked on the first release of Bing.

Thought I was retired 8 years ago when I volunteered to advise some Capstone teams of seniors in EE at University of Washington Bothell. Turned out it paid, I loved it, one thing led to another and here I am.



Hamilton C shell

From Wikipedia, the free encyclopedia

Hamilton C shell is a clone of the Unix C shell and utilities[1][2] for Microsoft Windows created by Nicole Hamilton[3] at Hamilton Laboratories as a completely original work, not based on any prior code. It was first released on OS/2 on December 12, 1988[4][5][6][7][8][9] and on Windows NT in July 1992.[10][11][12] The OS/2 version was discontinued in 2003 but the Windows version continues to be actively supported.

Hamilton C shell

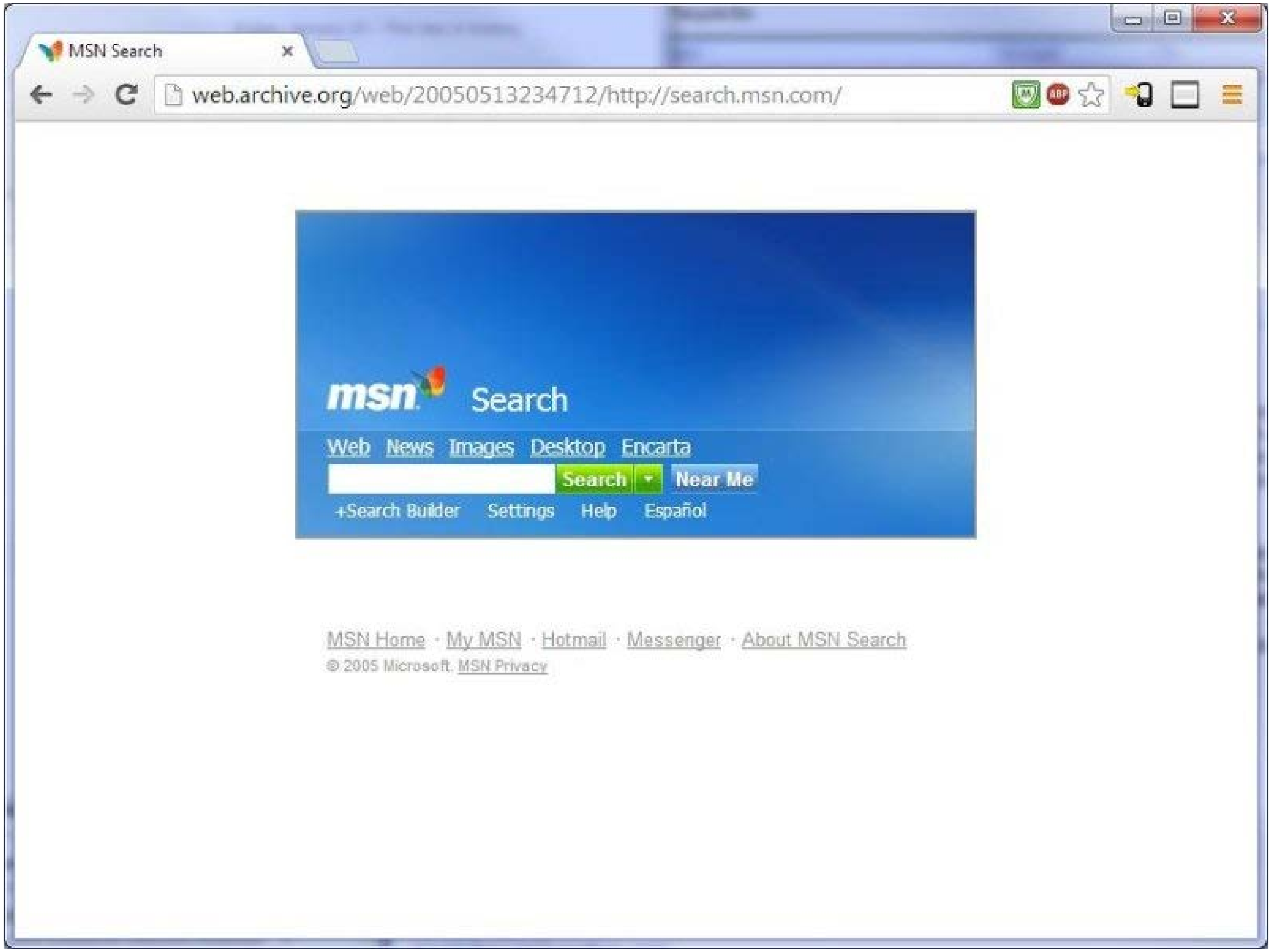
64-bit Hamilton C shell on a Windows 7 desktop.

Original author(s)	Nicole Hamilton
Initial release	December 12, 1988; 28 years ago
Stable release	5.2.g / March 5, 2017; 5 months ago
Written in	C
Operating system	Windows
Type	Unix Shell on Windows
License	Commercial proprietary

Contents [hide]

Design [edit]

Hamilton C shell differs from the Unix C shell in several respects, its compiler architecture, its use of threads, and the decision to follow


MSN Search
web.archive.org/web/20050513234712/http://search.msn.com/
msn Search
Web News Images Desktop Encarta
Search
Near Me
+Search Builder Settings Help Español
MSN Home · My MSN · Hotmail · Messenger · About MSN Search
© 2005 Microsoft. MSN Privacy

MSN Search in early 2005.

MSN Search
web.archive.org/web/20050513234712/http://search.msn.com/
msn Search
Web News Images Desktop Encarta
Search
Near Me
+Search Builder Settings Help Español
MSN Home · My MSN · Hotmail · Messenger · About MSN Search
© 2005 Microsoft MSN Privacy

Joined the team in July 2003 as the ninth member.

The ranker was the last major piece no one had taken.

Wrote the ranker and the query language for the first release in January 2005.

Almost 30 KLOC in C++.

This is the fourth time I’m teaching this class, the first time under its permanent number EECS 440.

You can read the supporting statement I had to submit to the college to get it approved on Canvas in the documents folder.

I'm still working to improve it.

This semester

1. Will release all the HW for the entire semester.
2. New HTML parser and Bloom filter HW assignments.
3. All the HW except the first few will be groups of 2 or 3.
4. Will try to make it easier to meet teammates with some speed-dating breakouts in the labs, and an online matching site.
5. Adding more structure to the labs to help you through the HW and how to apply it to your engine.

Week	Dates	Monday Lecture	Wednesday Lecture	Lab Topic	Assignment
1	Wed Jan 20 to Sun Jan 24		Introduction	Introduction and speed-dating breakouts	1//24 HW 1: Most positive subsequence
2	Mon Jan 25 to Sun Jan 31	Search engine basics	Project planning	Planning the project, what's most important, speed dating breakouts	1/31 HW 2: Personal goals and Gantt chart for graduate school
3	Mon Feb 1 to Sun Feb 7	HTML, Utf8, HTTP and redirects	TCP/IP, DNS, sockets	Getting started with AWS	2/7 HW 3: HTML parser
4	**Add/Drop deadline Feb 8** Mon Feb 8 to Sun Feb 14	Listen, marshalling data, SSL	Intro to the filesystem	Read an HTTPS webpage.	2/8 G-HW 1: Group photos 2/14 G-HW 2: Project plan
5	Mon Feb 15 to Sun Feb 21	Mapped files	Hashing and hashfiles	WC a directory using mapped files	Group meeting with Professor Hamilton 2/21 HW 4: Read an HTTPS webpage 2/21 G-HW 3: String and vector
6	Mon Feb 22 to Sun Feb 28	Processes and threads	No lecture. Well-being break	Hash table and hash blob	2/28 HW 5: WC a directory using mapped files
7	Mon Mar 1 to Sun Mar 7	Locks, RAII, and producer-consumer relationships	Tiny web server	Multithreaded server	3/7 HW 6: Memory-mapped hash table 3/7 HW 7: Memory-mapped hash blob
8	Mon Mar 8 to Sun Mar 14	Midterm 3:00-5:00pm	The frontier	Bloom filter	3/14 HW 8: Multithreaded server
9	Mon Mar 15 to Sun Mar 21	The index	The constraint solver	Distributing your engine on the cloud	3/21 HW-9: Bloom filter
10	Mon Mar 22 to Sun Mar 28	Top-down recursive descent	The query compiler	Simple expression parser	
11	Mon Mar 29 to Sun Apr 4	Ranking	JSON and a simple web page	Presenting your search results	4/4 HW 10: Simple expression parser
12	Mon Apr 5 to Sun Apr 11	Duplicates and shingling	Beyond basic search	No labs. Staff meetings with teams.	
13	Mon Apr 12 to Sun Apr 18	Ethics	Course debrief	No labs. Staff meetings with teams.	G-HW 4: Presentation slides
14	Mon Apr 19 to Wed Apr 21	Group presentations	Group presentations	No labs.	G-HW 5: Final reports and code snapshots Group meeting and demo with Professor Hamilton
15	Mon Apr 26	Final Exam 10:30am-12:30pm			

Lab instructors

Alex Erf
alexerf@umich.edu

Fridays
10:30 to 11:30 am
Zoom
https://umich.zoom.us/j/97763265848

Alex Jalkanen
alexjalk@umich.edu

Thursdays
10:30 to 11:30 am
Zoom
https://umich.zoom.us/j/94438882408

Daniel Hoekwater
dhoek@umich.edu

Thursdays
3:00 to 4:00 pm
Zoom
https://umich.zoom.us/j/99742264598

You can attend any lab you like.

Course organization

1. You don’t need to know anything more than 281.
2. I will teach you everything else you need to know.
3. First half will focus operating system topics needed to build the engine.
4. Second half will focus building a complete working internet search engine.
5. You’ll have homework nearly every week to help you learn the material. You’ll do most of it in groups.

Grading

Group project	
Group performance	25%
Individual contribution	25%
Homework	20%
Midterm	15%
Optional final	15%

If you skip the final, I'll use your midterm score.

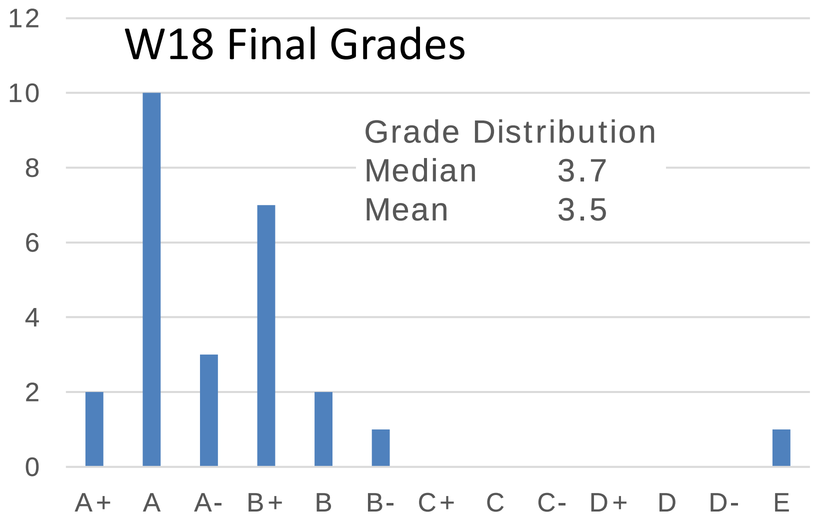
Some of the grading will be done by surveys.

Past curves

In the syllabus, I tell you to expect that most students will fall between 2.8 and 4.0 with median around 3.2 or a little higher.

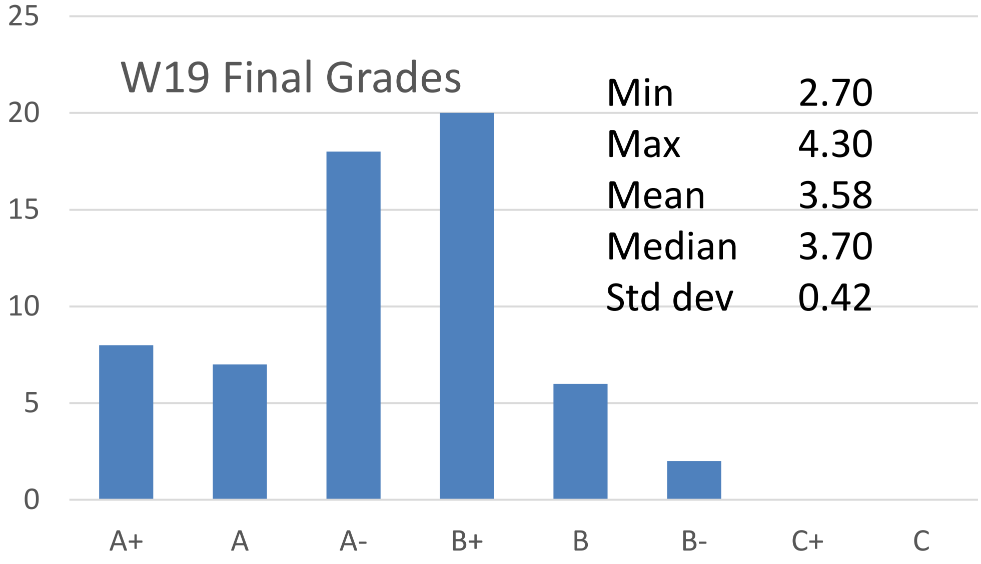
This is what I've actually done.

The past is no guarantee about the future.



Grade Distribution

Median	3.7
Mean	3.5

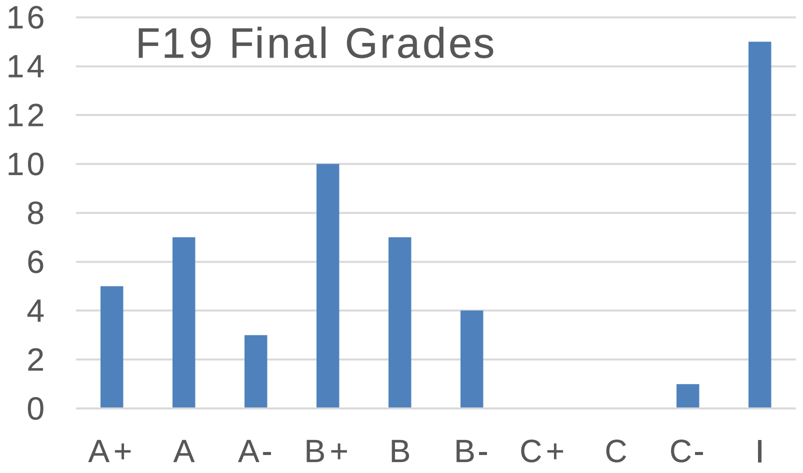


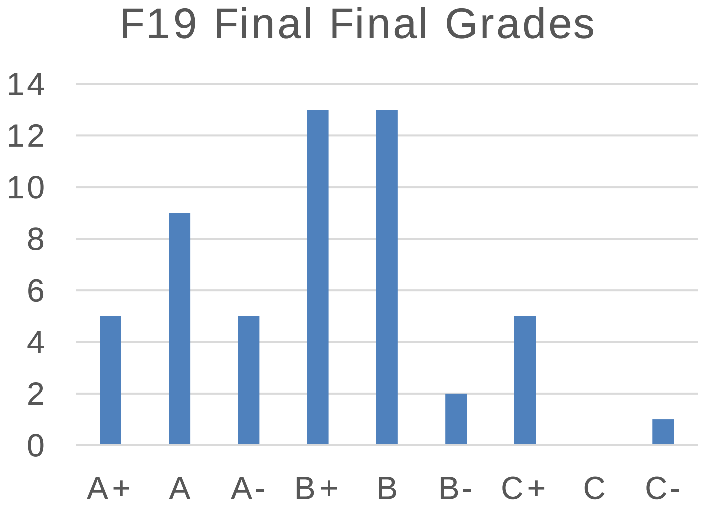
Min	2.70
Max	4.30
Mean	3.58
Median	3.70
Std dev	0.42

F19 was surprising

Initially, lots of incompletes from teams that needed to demo in January 2020.

The midterm break comes too late in the fall.





All the work must be your own

1. Copying answers from another student or off the internet, omitting attribution, submitting work that’s not your own or attempting to deceive me will be reported for academic misconduct.

2. I’m good at spotting misconduct and very good at reporting it.

3. I do not give warnings. I report everything.

Where you’ll find stuff

Canvas for announcements, files, individual and group homework assignments and grades.

Google docs for links to Zoom lecture recordings.

The autograder to test code for basic correctness.

Piazza for questions and discussion.

We expect to have a course website soon.

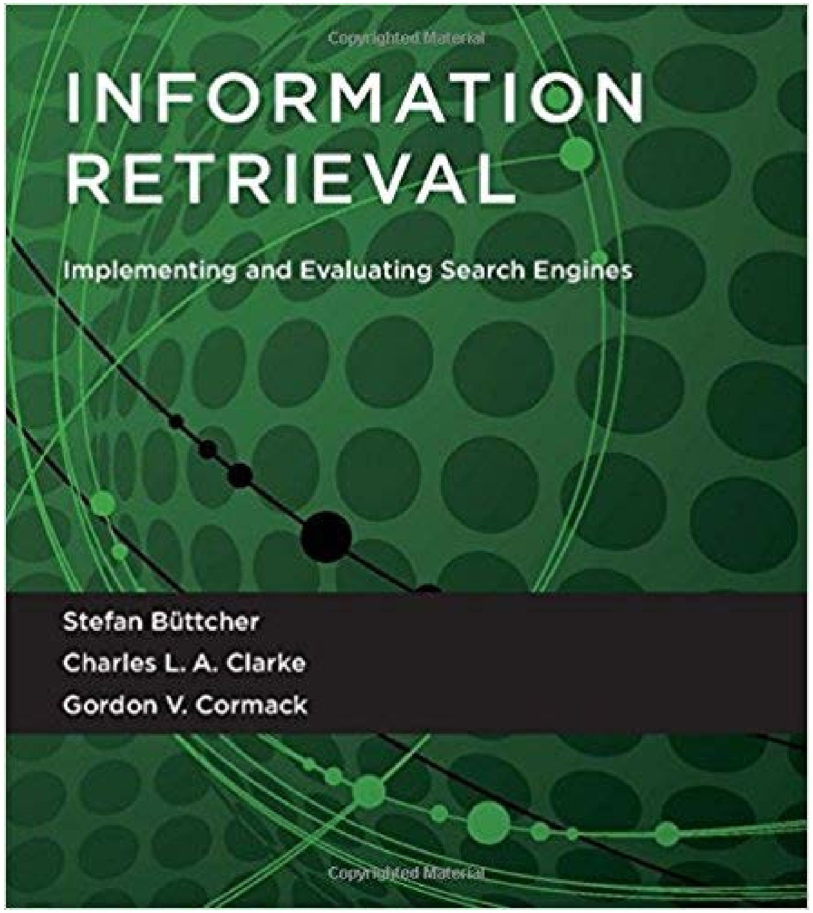
Learning objectives

1. Ability to work on a team to design a large software project you don’t already know how to do.
2. Learn how a search engine works.
3. Appreciation of software as art.
4. Ability to create elegant, reliable code in C++.
5. Learn how a large system is decomposed into objects that talk to each other.
6. Appreciation of how an application interacts with the operating system and the user.
7. Familiarity with simple projecting planning tools and concepts, Gantt charts, software metrics, LOC.
8. Appreciation of the what it means to be an ethical engineer.

Interwoven themes

1. What is system design?
2. What is a beautiful design?
3. How do you tackle a large project?
4. How do you work as a team?
5. How does a search engine work?
6. How do you use the operating system?

Textbooks



Information Retrieval: Implementing and Evaluating Search Engines
Reprint edition (February 12, 2016)
Stefan Büttcher, Charles L.A. Clarke,
Gordon V. Cormack
The MIT Press
ISBN 978-0262528870

Required.
Read the first 6 chapters.

Characteristics of system design projects

1. There’s an important domain-specific part: You need to learn something new about an interesting problem you’ve never seen before.
2. You need to invent a solution with lots of moving parts.
3. It’s usually “close to the metal”.
4. They're usually team efforts because they're too big to do any other way.
5. You get to build your part from scratch and it feels good.

A search engine hits on every bit of what a system design project is to me.
I also intend for it to relatable to family and friends and recruiters.
I want you to get jobs.

Project

You are to self-select into teams of 6 to design and build an end-to-end search engine completely from scratch in C++, assigning your own roles.

Past engines in LOC

	Project total			Individual		
	w18	w19	f19	w18	w19	f19
High	14,271	26,887	19,264	4,096	8,646	7,826
Median	5,300	13,000	11,973	1,006	1,750	1,572
Low	4,170	9,414	4,575	0	444	120

Past index sizes in documents

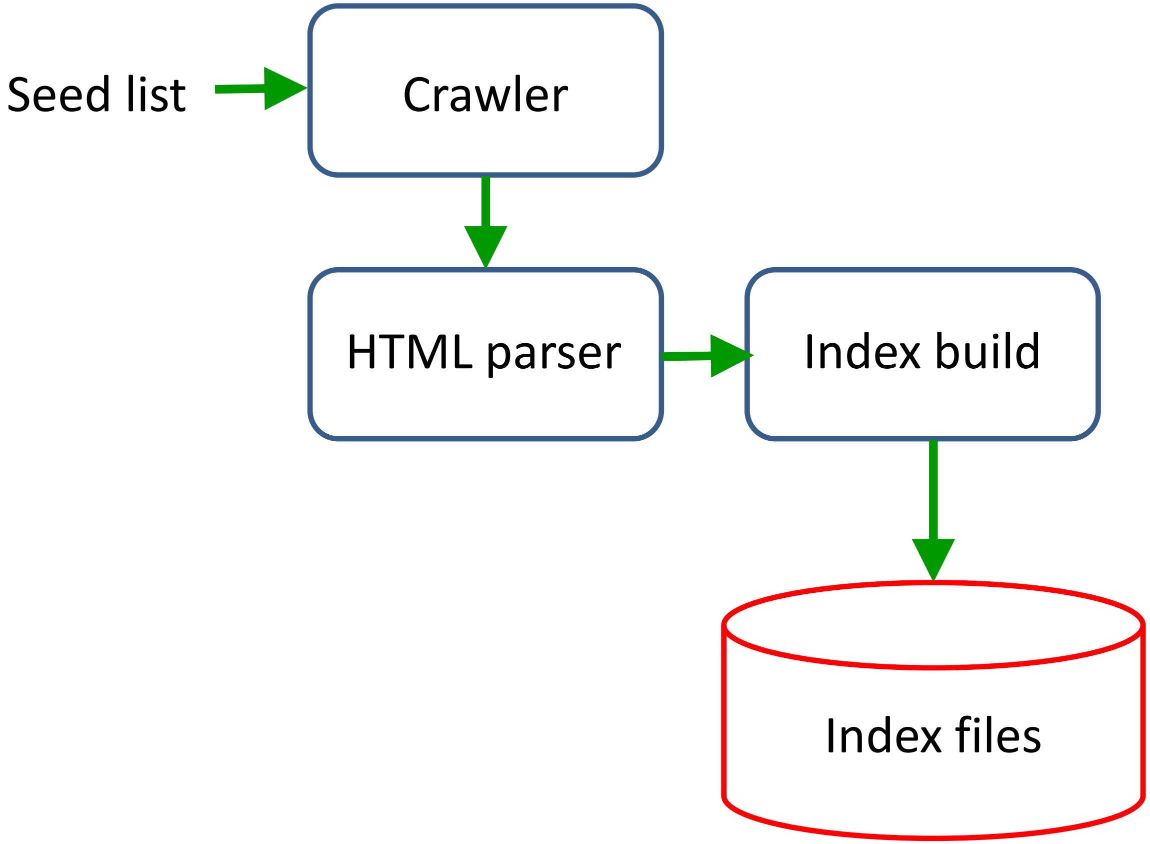
	w18	w19	f19
High	13.4M	150M	586M
Median	8,000	4.65M	397M
Low	4,816	1.9M	207M

Project

These are the basic pieces you will need.

1. HTML parser.
2. Crawler.
3. Index.
4. Constraint solver.
5. Query compiler.
6. Ranker.
7. Front end.

The index build side



Seed list

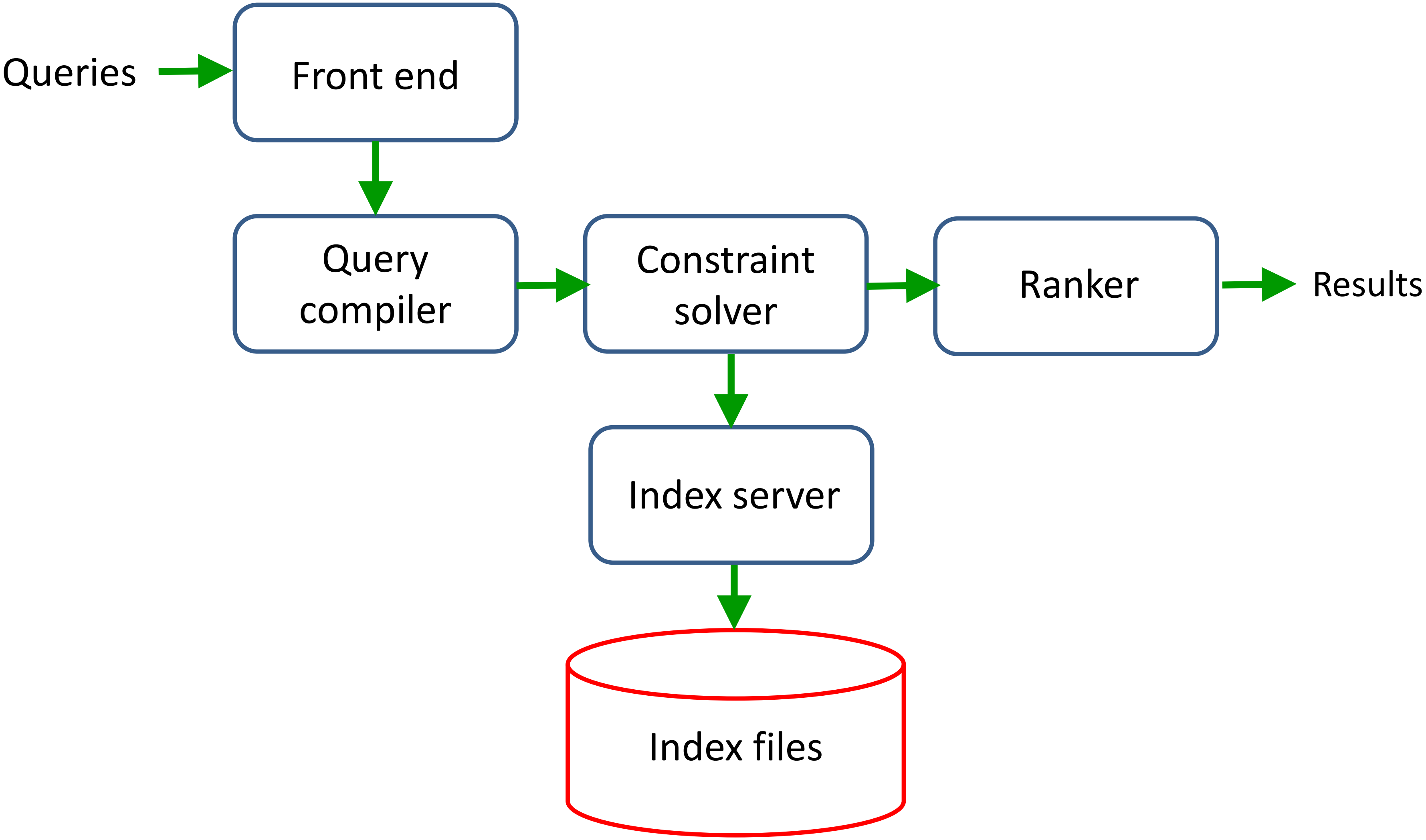
Crawler

HTML parser

Index build

Index files

The query serve side



Your search engine

1. You may deliver your engine on any platform but everyone always chooses Linux.
2. I will arrange Amazon AWS accounts for all of you.
3. All of your work will be in C++ and all of it must be yours.
4. All your code must conform to my stylesheet.
5. I care less about test cases than I do about getting things working.
6. You are discouraged but not prohibited from using STL anywhere the details of the implementation matter.
7. You will meet with me and the course staff periodically as a team to discuss your plans and progress.
8. You will have a final review, demo, submit a paper and an archive of your code.
9. You will give a short presentation.

Levels of functionality

0. Create a plan.
1. Parse text files into a hash table.
2. Build a crawler.
3. Build a reverse word index.
4. Create a user interface.
5. Build a constraint solver and query parser.
6. Build a ranker.
7. Advanced functionality, e.g., distributed processing.

What are the most important determinants of all your outcomes in life?

I suggest it’s all the other people in your life. Sometimes, you get to choose.

Observations

On the best-performing teams:

1. They like each other.
2. They buy into the rules and set out to win at them.
3. Their plans are filled with a lot of brain-storming about the problems they need to solve and how they'll do it.
4. There's a lot of brain-storming in their execution as well.
5. They come to my office frequently.

Observations

On teams having difficulties:

1. Their plans are often thin on detail, treating much of the design as TBD once we got to the appropriate lecture.
2. They check off some “all of the above” boxes, e.g., “all of level 7”, without much analysis of what that would entail.
3. There are questions about who is in charge. Decisions don’t get made. No one is available to meet at the same time.
4. They don’t do a lot of brainstorming and they don’t resolve technical questions with technical arguments.
5. Interfaces between the components and what each component does are unclear.
6. They complain about the rules.

Observations

What makes for a great teammate:

1. They step up and get stuff done.
2. It’s less about what they know and more about what they’re willing to do.
3. They have lots of ideas but they don’t insist on their own.
4. They respect boundaries.
5. They’re available and supportive.

Your first group activity will be form a group and submit a group photo.

We’ll do “speed dating” breakouts in the labs this week to help you get to know others in the class.

You may also find Alex’s https://group-finder.com helpful.

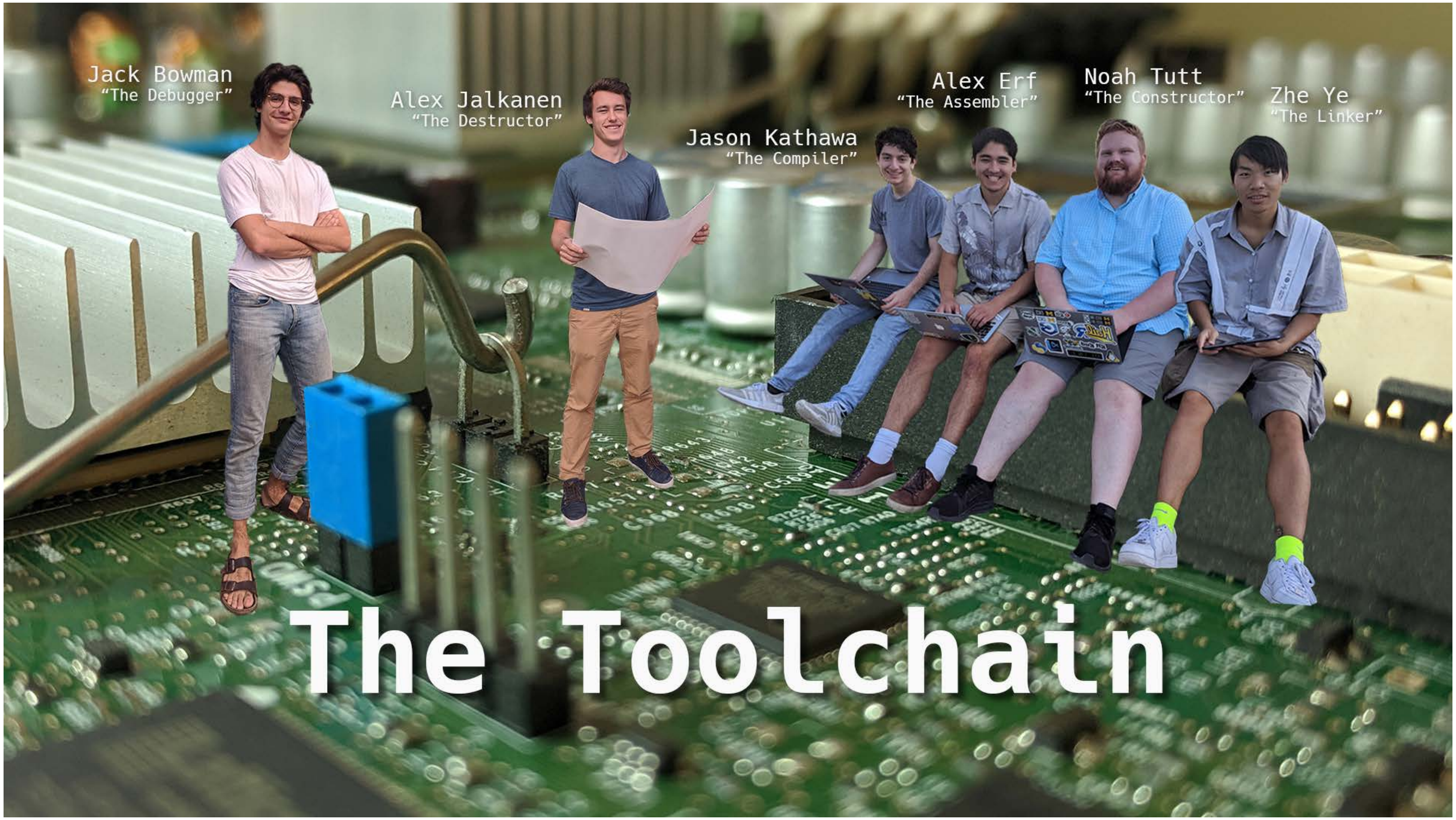
Group photo

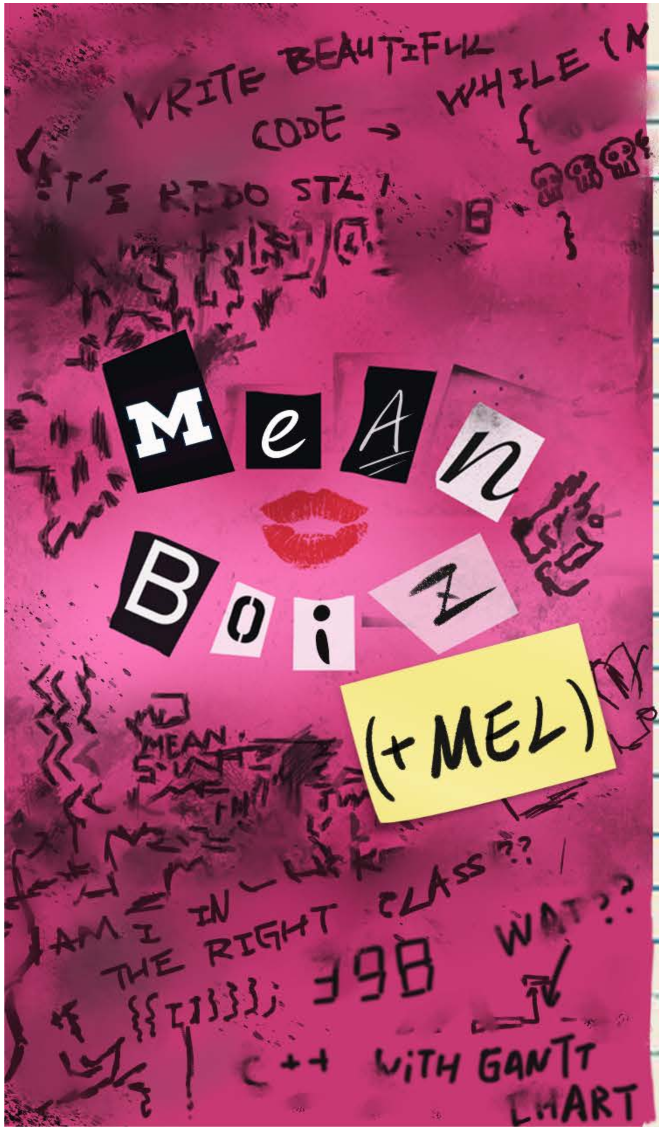
1. You are to form groups of 6, choose a name for your group and submit a photo of yourselves.
2. You must all appear in the photo. Faces have to be clearly visible and easily recognized.
3. Each person must be named in the photo.
4. Your group name must appear in the photo.
5. You may not spend any money on this.
6. You can edit the image.
7. You may not steal copyrighted artwork.
8. You may use public domain images.

1. This assignment is competitive based on creativity and execution.
2. Your grades will be determined by a vote.
3. Please submit only one copy for the entire team.

Here were some past submissions.

MAXIMAL MUNCH
BRANDON KAYES
RYAN WUNDERLY
DANIEL HOEKWATER
ALEX RAISTRICK
AUSTIN KIEKINTVELD
ADOLFO APOLLONI


Jack Bowman
"The Debugger"
Alex Jalkanen
"The Destructor"
Jason Kathawa
"The Compiler"
Alex Erf
"The Assembler"
Noah Tutt
"The Constructor"
Zhe Ye
"The Linker"
The Toolchain



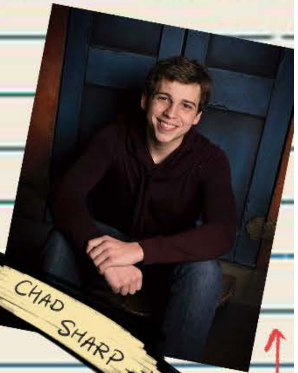
CODY MORTERUD

Uses One letter variable names

Also doesn't write comments

Actual High School Mathlete

MELISSA GEORGE



CHAD SHARP

Uses nested tenary operators

COLTEN WILLIAMS

Goes into a CS research not knowing struct

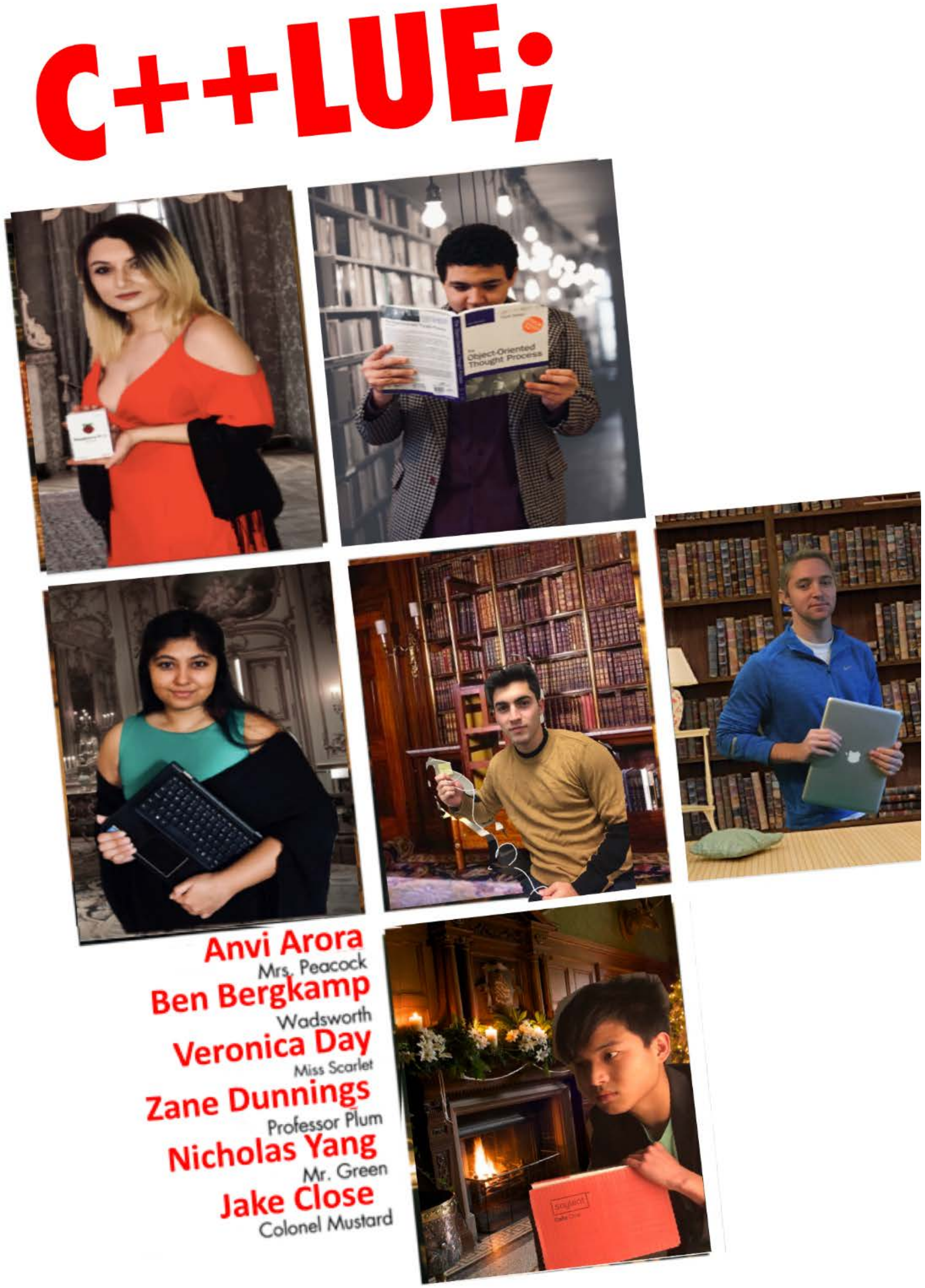
plays hype music when submitting to AG

RAHUL AGRAWAL BEJARANO

GABRIEL LUO

Indents with 3 spaces non-ironically

KEYBOARD SURFERS
SHESANTH RAMAKRISHNAN
THOMAS BARTLETT
ERAY MITRANI
ENGIN AKDEMIR
CAROLYN BUSCH



C++LUE;

Anvi Arora
Mrs. Peacock
Ben Bergkamp
Wadsworth
Veronica Day
Miss Scarlet
Zane Dunnings
Professor Plum
Nicholas Yang
Mr. Green
Jake Close
Colonel Mustard

Internet Explorers
Pranav Tadikonda, Anthony Jonikas, Andrew Shea, Federico Kulyckyj, Julia Atayde, Chris Bahr

CodeKeepers

THE EXCEPTIONS
"BAT_EXCEPTION"
YUAN YAO
"INVALID_ARGUMENT"
YICHANGLE ZHAO
"BAD_ALLOC"
SHANG ZHOU XIA
"BAD_WEAK_PTR"
KONSTANTIN GOLOBOKOV
"RUNTIME_ERROR"
GRAHAM PALMER

DEVESH MODI
CHRIS HOANG
SACH VAIDYA
OSAMA SAEED
SAMIUR KHAN
MINH NGUYEN
MIHIR BALA
THE SEEKERS
OUT OF 14,000,605 DOCUMENTS, ONLY 1 WAS RELEVANT

Background image from https://img.buzzfeed.com/buzzfeed-static/static/2014-11/5/20/enhanced/webdr09/enhanced-1071-1415238999[illegible].jpg

Produced By: Graham Eger, Max Guthmann,
Bradley Huang, Dennis Li, Jason Setting, and Jacob Crouch

Injured in a browser crash?

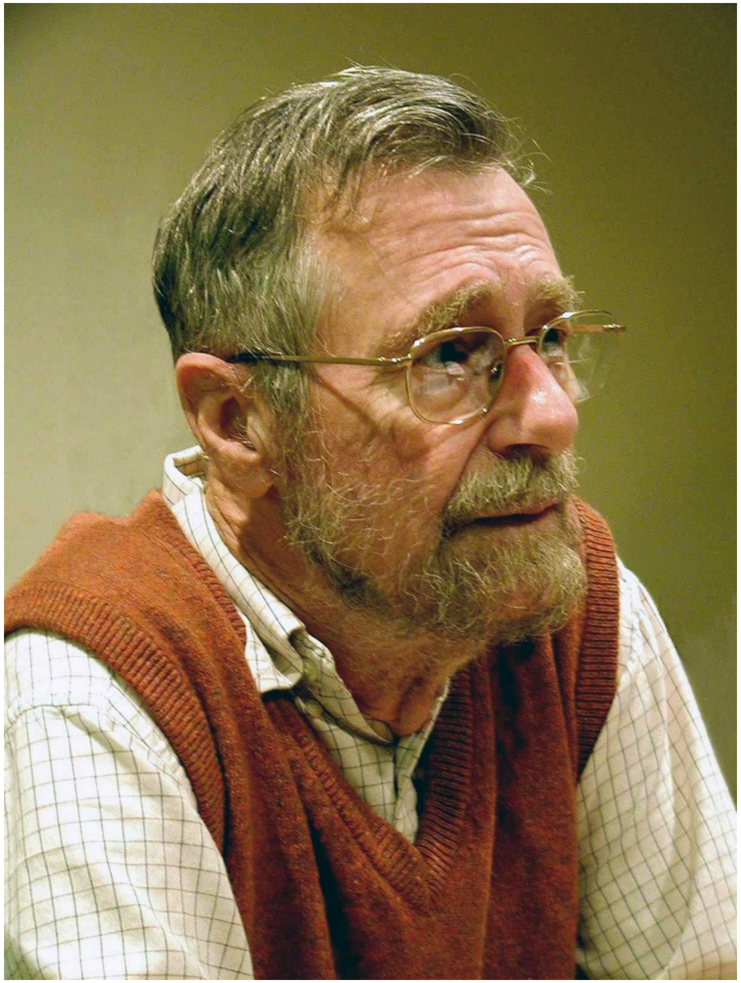
Your search ends here. **Call: 1-800-GETBREAD**

Office of Easton&Jin&Lau&Nagel&Zhong&Zhong

My usual experience

A team’s performance on the simple task of submitting a creative photo of themselves is often predictive of how they’ll do on everything else.

Please don’t do something lame.

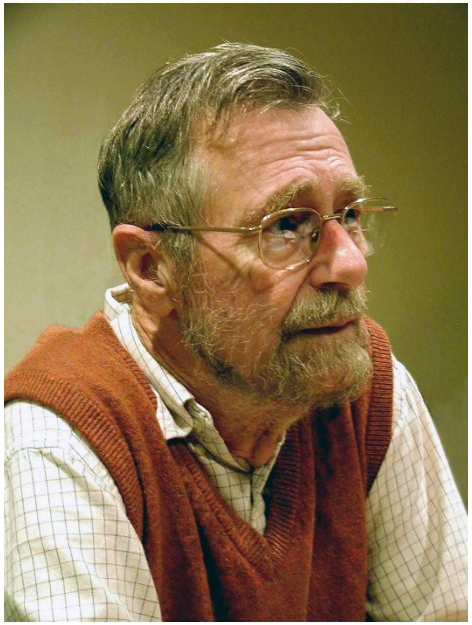


Who is this?

Edsger Dijkstra
(1930 – 2002)

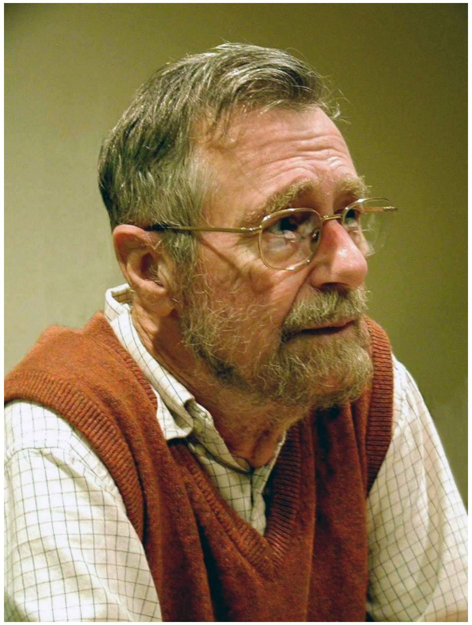
Dutch computer scientist, coined "structured programming", known for his work on mutual exclusion, winner of the first ACM Turing Award in 1972.

Edsger Dijkstra



Big believer in the importance of simplicity and highly critical of baroque programming languages with lots of features, especially PL/I, the C++ of the day.

Edsger Dijkstra



"I absolutely fail to see how we can keep our growing programs firmly within our intellectual grip when by its sheer baroqueness the programming language – our basic tool, mind you! – already escapes our intellectual control."

"The Humble Programmer", CACM, October 1972.

Why I’m not a fan of STL

Algorithmic and tuning choices, performance and size tradeoffs and other issues are much *harder to spot* when hidden inside an opaque template, especially *where the whole point* of the template is that *you’re not supposed to care how it works* inside.

How big is an STL string?

How big is an STL string?

It depends on the compiler and OS.

With the MS compiler, an empty string takes 36 bytes = 28 bytes on the stack + 8 bytes on the heap. New'ing a string, it's the same two chunks but both go on the heap. Adding "hello" does not require additional memory. Adding another 28 characters requires another 48 bytes.

With Cygwin g++, an empty string takes 8 bytes whether it's on the heap or the stack. But the moment you add "hello" to it, it jumps to 38 bytes. Adding another 28 characters requires another 58 bytes allocated and 30 freed.

With g++ under WSL, an empty string takes 32 bytes whether it's on the heap or the stack. Adding "hello" does not require additional memory. Adding another 28 characters requires another 34 bytes.

Your team grade on the project

Competitive, based on ranking your engine's functionality, *performance*, e.g., time and size to *crawl, create or search* a *multi-terabyte index*, its *features* and *quality* against the engines created by the *other teams* on the same and different platforms.

Your individual grade on the project

Based primarily on the *code you contributed*, including:

1. The number of *lines of code* you wrote,
2. The *complexity of the tasks* it performs,
3. Its *performance*,
4. The *creativity* and the *overall elegance*.

I will also consider other contributions you made to your team as measured by survey of your teammates.

HW1 MostPositiveSubsequence( )

Write a C++ function that can scan a sequence of N integers in an array A, returning the sum and the left and right indices of the most positive subsequence.

For example, for the sequence

{ -1, 3, 5, 6, -2, -4, 1, 7, -15, 12, 7, -5 }

0 1 2 3 4 5 6 7 8 9 10 11

the best sum = 20, left = 1, right = 10.

HW1 MostPositiveSubsequence( )

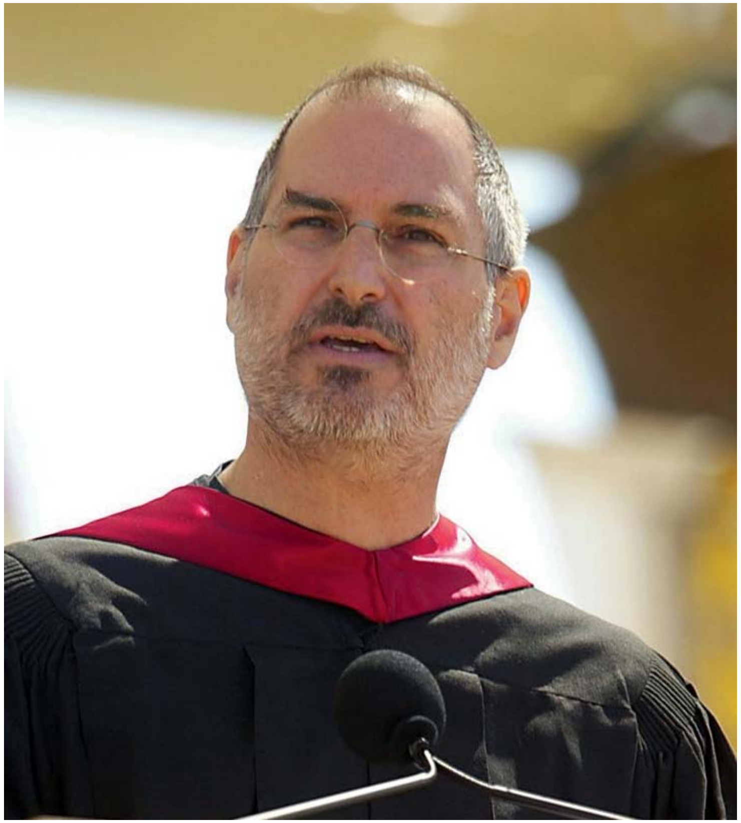
1. Due Sunday, January 24, 2021.
2. Write and submit two C++ files to the autograder.
    a. MostPositiveSubsequence.cpp, containing your implementation of the MostPositiveSubSequence( ) function.

    b. BestSubsequence.cpp, containing a simple main( ) routine that takes a sequence on the command line an argv and reports the results.
3. You must also demonstrate that you can build and debug it *with a graphical debugger* by uploading a screenshot showing you stepping through your code.

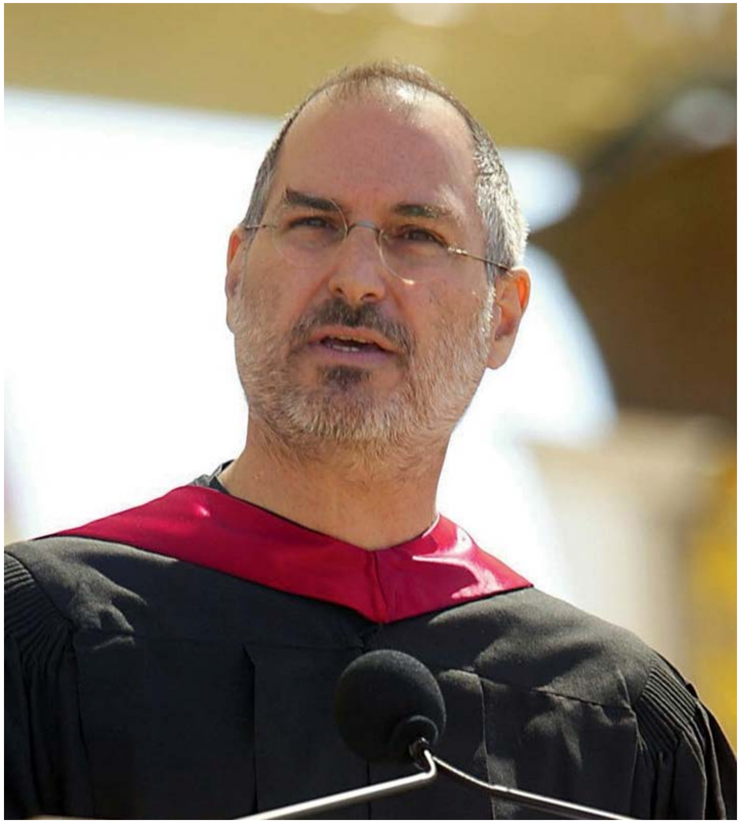
General rules for homework

Unless otherwise specified:

1. You must solve it on your own or as a registered group.
2. You are to make good choices anywhere you find the spec ambiguous and you will be judged on them.
3. You may not discuss it with others or use the internet or other resources to help.



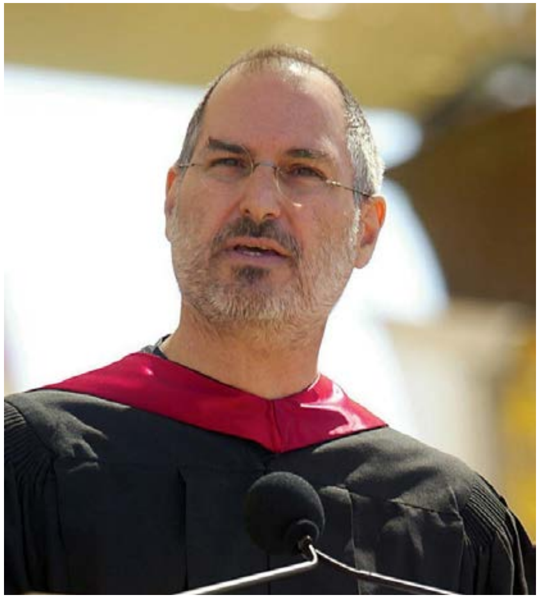
Who is this?



Steve Jobs (1995—2011)

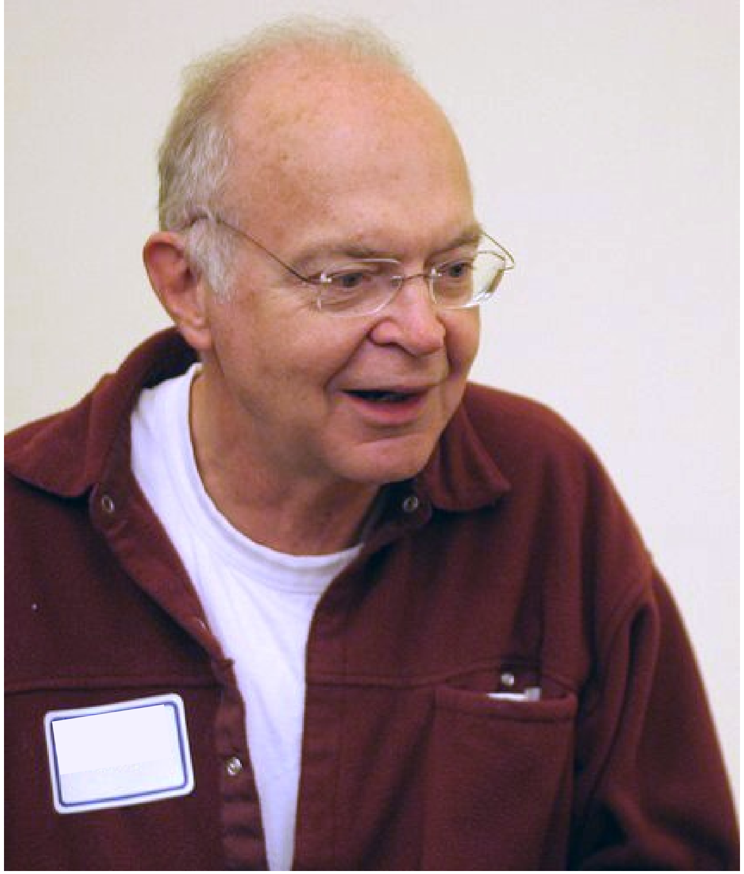
Shown here giving his commencement address at Stanford, 2005.

Image source: http://bullandbearmcgill.com/wp-content/uploads/2014/04/img_steve.jpg



"You've got to *find what you love.* And that is as true for your work as it is for your lovers. Your work is going to fill a large part of your life, and the only way to be truly satisfied is to do what you believe is great work. And the only way to do great work is to *love what you do.* If you haven't found it yet, keep looking. *Don't settle.*"
-- Steve Jobs

Source: http://news.stanford.edu/2005/06/14/jobs-061505/



Who is this?

Don Knuth

Professor Emeritus at Stanford

Famous for his *Art of Computer Programming* textbooks. (Seven planned, only 3 written.)

He was my professor 45 years ago for CS 144A and B, introduction to algorithms and data structures.

Image source: https://en.wikipedia.org/wiki/File:KnuthAtOpenContentAlliance.jpg

Don Knuth on programming as art

"When I speak about computer programming as an art, I am thinking primarily of it as an art *form*, in an aesthetic sense. The chief goal of my work as educator and author is to help people learn how to write *beautiful programs*."

-- Don Knuth, CACM, December 1974

Source: http://www.paulgraham.com/knuth.html

“My feeling is that when we prepare a program, it can be *like composing poetry or music*; … programming can give us both *intellectual and emotional satisfaction*, because it is a *real achievement* to master complexity and to establish a system of consistent rules.”

So what is beautiful code?

Beautiful code

“I may not know art but I know what I like.”

“If they can’t tell me how they would improve it if they could do it over, I won’t hire them.”

“Artists are never done with a piece, they just stop working on it.”

Beautiful code

To me, beautiful code should be:

1. Elegant, easy to read, sparse, clear.
2. Stylish.
3. Fast and efficient.

It should make a hard problem seem easy.

To get better, you must be able to criticize your work.

Beautiful code

Consider the problem:

Write a function that determines if a C-string contains doubled letters.

Here is one sample solution.

```
bool has_doubled_letters(const char *str) {
    const char *ptr = str + 1;
    while (*str != '\0' && *ptr != '\0') {
        if (*str == *ptr) {
            return true;
        }
        ++str, ++ptr;
    }
    return false;
}
```

Here is another.

```
bool HasDoubledLetters( const char *s )
   {
   while ( *s && s[ 0 ] != s[ 1 ] )
      s++;
   return *s;
   }
```

Can we say if we like one better than another?

```
bool has_doubled_letters(const char *str) {
    const char *ptr = str + 1;
    while (*str != '\0' && *ptr != '\0') {
        if (*str == *ptr) {
            return true;
        }
        ++str, ++ptr;
    }
    return false;
}
```

```
bool HasDoubledLetters( const char *s )
   {
   while ( *s && s[ 0 ] != s[ 1 ] )
      s++;
   return *s;
   }
```

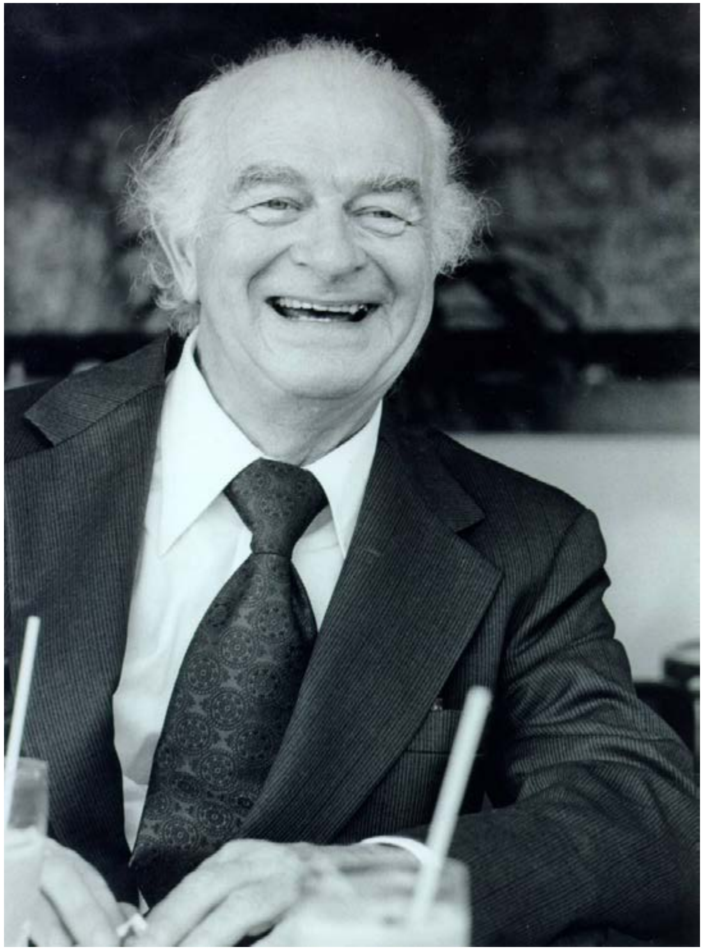
A lot of problem-solving is coming up with ideas of how to do something.

A lot of problem-solving is coming up with ideas of how to do something.

For example, a lot of debugging is coming up with ideas of how you got it wrong and how you’ll find your mistake.

The number 1 problem many people make is stopping with their first idea.

They refuse to ask what’s wrong with it and how it could be made better or if a different idea would be even better.



Who is this?

Linus Pauling (1901-1994)

American chemist, biochemist, peace activist.

Nobel Prizes in Chemistry and Peace.

Image source: http://www.villages-news.com/linus-pauling-and-prostate-cancer/

He was asked, *"How do you get so many good ideas?"*

“If you want to have good ideas *you must have many ideas*. Most of them will be wrong, and what you have to *learn* is *which ones to throw away*.”

-- Linus Pauling

As quoted by Francis Crick in his presentation, “The Impact of Linus Pauling on Molecular Biology” (1995).

Do you think people will steal your new idea?

Ballmer Laughs at iPhone - YouTube

www.youtube.com/watch?v=eywi0h_Y5_U ▾

https://www.youtube.com/watch?v=eywi0h_Y5_U

New ideas

If an idea is *really new*, it’s surprisingly hard to give it away.

If it’s *really good*, most people will think you’re crazy.

Most people have trouble imagining how their life would be different and why they would want that.

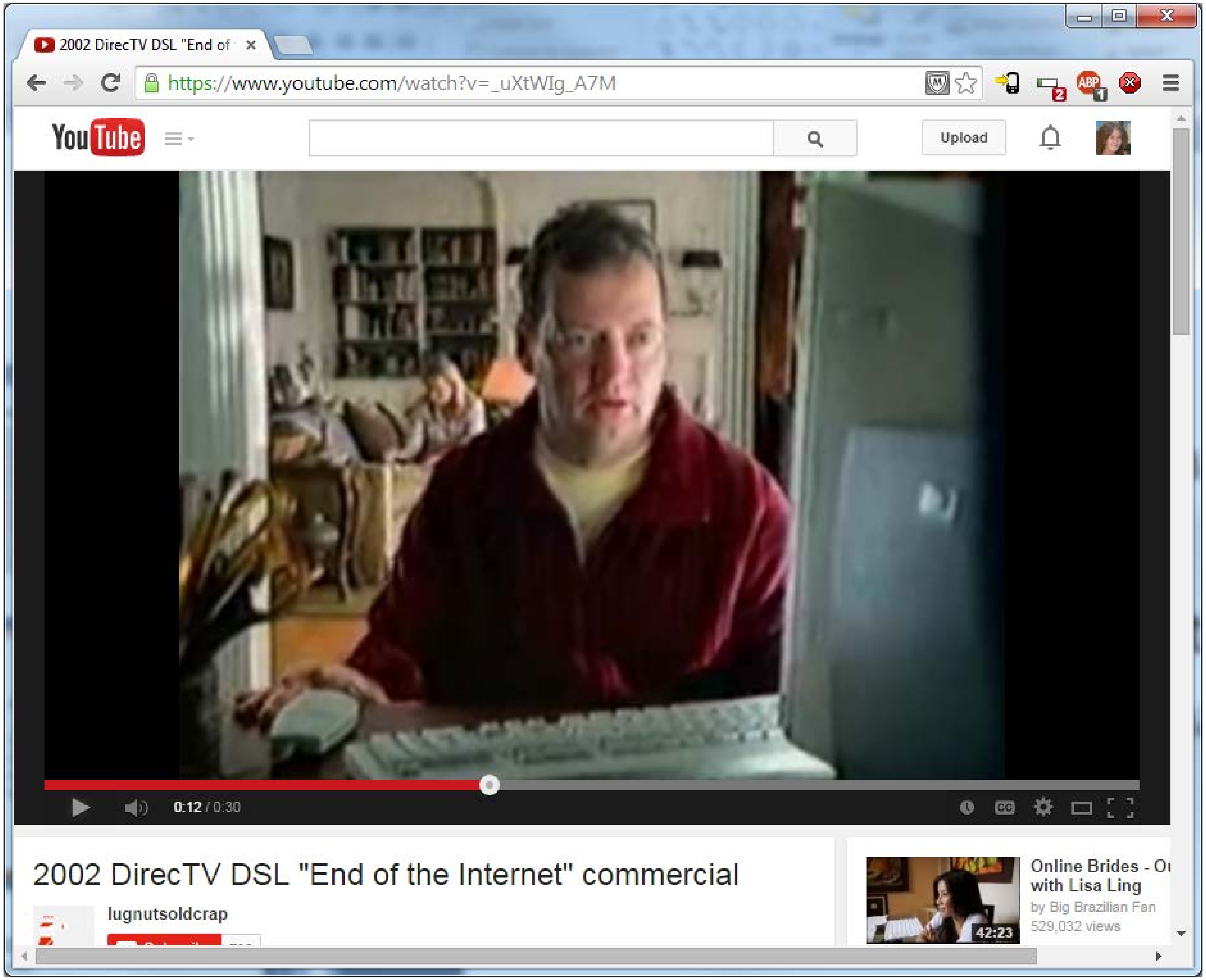
Problem-solving versus problem-choosing

A fabulous solution to a problem nobody cares about is still a solution nobody cares about.

But even a mediocre solution to a really important problem can be important.

Opportunities are created when the world changes.

Something becomes possible that wasn’t before.



2002 DirecTV DSL "End of the Internet" commercial

lugnutsoldcrap

https://www.youtube.com/watch?v=_uXtWIg_A7M

What made search engines possible

The insight was that you could do it.

You could spider the entire web and make and index your own private copy of the whole thing. Processors, bandwidth and storage were so cheap you could do it.

When MSN Search went live in Jan 2005, we had 10 rows of 500 machines, each with a copy of 1/500-th of the entire web = 10 complete copies.