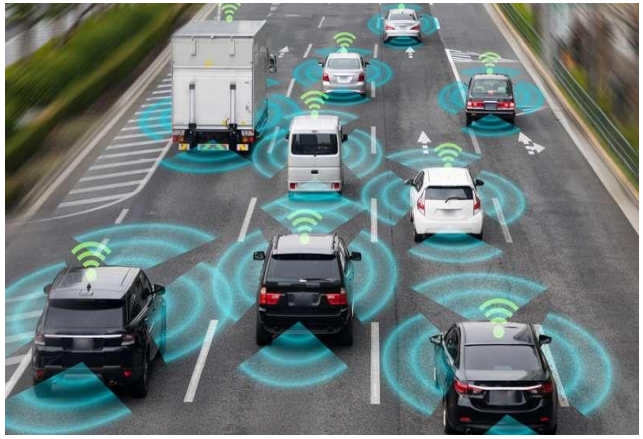
CS103 Unit 0 – Introduction

Mark Redekopp

Welcome to Computer Science



What is Computer Science

- All science is computer science
  - It is very interdisciplinary: Math, Engineering, Medicine, Natural sciences, Art, Linguistics, Social Sciences etc.
- It is more about problem-solving and developing algorithms to solve information-based problems
  - How do I recognize objects in a photograph
  - What patterns can be found in a set of data that would allow me to predict future outcomes
  - Identify the function of this protein given it structure
- Computer science is no more about computers than astronomy is about telescopes*
  - Computers are the primary tool

*This is a famous quote. However, who said it first is unclear.

What Computer Scientist Do…

- Find methods to solve information-based problems (algorithms) **[this is truly CS]**
  - Observe, organize, transform and discover useful information from data
  - Use math and logic to solve problems
  - Work in (cross-discipline) groups
- Convert these methods/algorithms to a form that a computer can execute **[this is programming]**
- **We generally do both**

Computer Science Is…

- A great way to make a living
  - ***Maria Klawe, et. al. - To the age-old question -- "What do you want to do when you grow up?" -- children today give many modern answers: "Help feed hungry families." "Prevent and cure diseases." "Find sources of renewable energy." "Understand the universe." One clear path leads to each of these aspirations: the study of computer science (& engineering).***

http://www.huffingtonpost.com/maria-klawe/computing-our-childrens-f_b_388874.html

CS 103 or CS 102

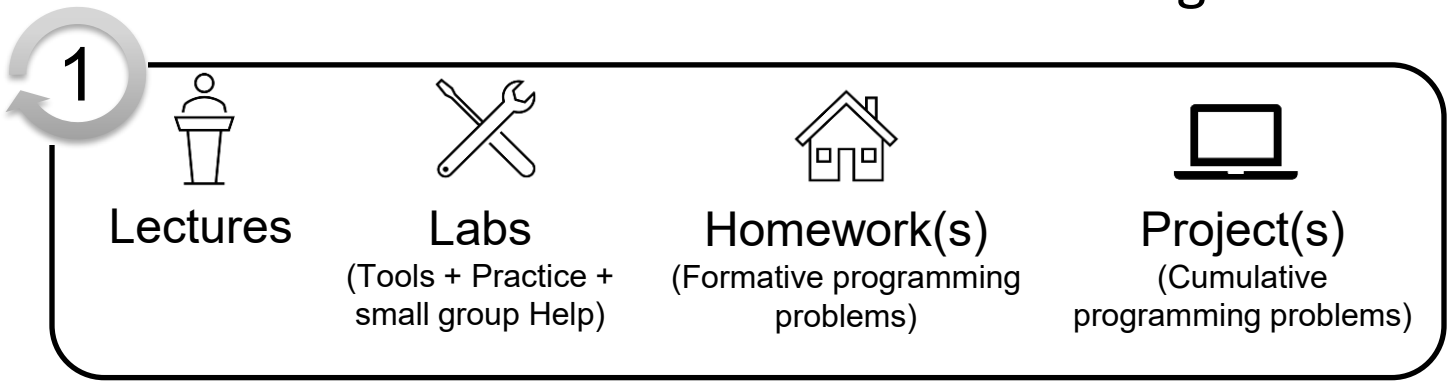
- CS103: Introduction to Programming
- **Introduction** but requires some prior programming experience
  - Must have passed AP CS exam and or the CS 102 placement exam
    - Should know one of Java, Python, or C/C++
  - Those that meet the above criteria, but don't feel confident may consider CS 102 as a slower-paced on ramp to programming
- **Programming**
  - We'll try to teach good coding practices and how to find efficient solutions (not just any solution)
  - We'll focus on concepts present in most languages using C/C++ as the primary language (not Java nor Python)

http://climbingla.blogspot.com/2010/05/walk-6-hermon-and-highland-park.html

http://upload.wikimedia.org/wikipedia/commons/2/2d/Diamond_Interchanges

Course Structure

- The course is broken into 6 units each consisting of:



1 **C++ Language Syntax**

2 **Pointers and Memory**

3 **Objects 1**

4 **Managing Data**

5 **Objects 2**

6 **Recursion**

Exams and Grading

- The course will utilize 3 exams during our Quiz section

  Midterm 1 – TBD

  Midterm 2 – TBD

  Final – Tuesday, May 12 4:30 - 6:30 PM

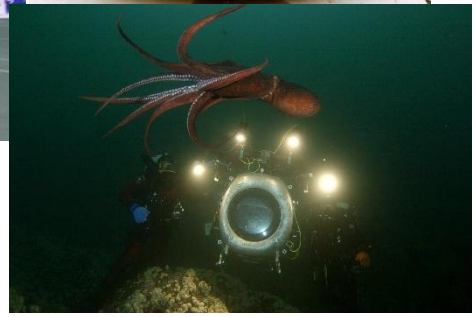
Grading will be as follows:

Labs	10%
Homeworks	15%
Projects	15%
Lowest Exam	15%
Median Exam	20%
Highest Exam	25%
Total	100%

Syllabus

Expectations

- Attend lectures & be engaged
  - Ask questions
  - We're a team…I need you!
  - I'll give you my best. Try to give me yours!
- Catch the wave!
  - Start assignments early, complete them early
  - Study and review after each lecture
- Go deep!
  - Read more on your own or ask questions
  - Don't be satisfied with partial understanding
  - Build the mental muscles needed to solve errors and bugs on your own

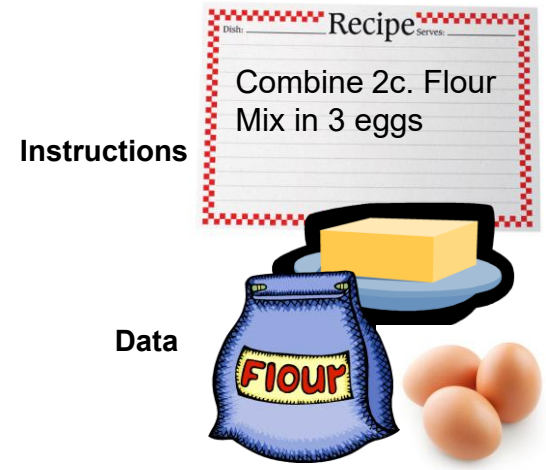


Programming Languages

- Imperative/Structured Languages
  - Describe the what (data) and how (instructions/algorithm)
  - Examples: C/C++, Java, Javascript, Python
  - The focus of most programming courses
  - Programs are like a recipe for how to operate on data

Quadratic Equation Solver

```
1   import math
2
3   a = int(input("Enter a: "))
4   b = int(input("Enter b: "))
5   c = int(input("Enter c: "))
6
7   det = b*b - 4*a*c
8   if(det >= 0):
9       r1 = (-b - math.sqrt(det)) / (2*a)
10      r2 = (-b + math.sqrt(det)) / (2*a)
11      print(f'Roots are {r1} and {r2}')
12  else:
13      print("Imaginary roots")
```

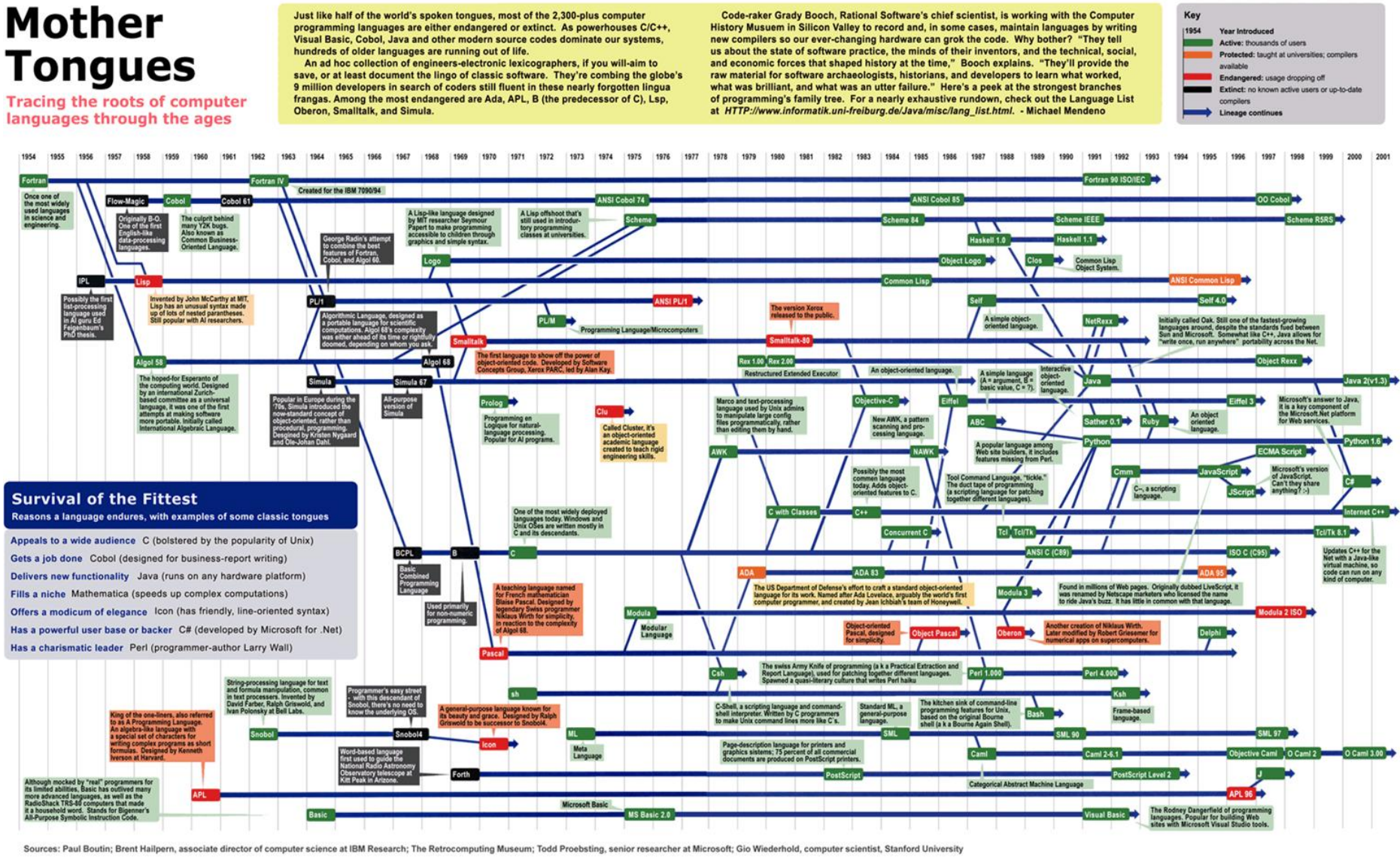


Computer (Reads instructions, operates on data)

Why C/C++

- We will teach C++ in this course because…
  - C/C++ is *still* a very popular language in industry
  - C/C++ is ubiquitous
    - Used everywhere, even to implement other programming languages (i.e. Python, Matlab, etc.)
  - C/C++ is close to the actual hardware
    - Makes it fast & flexible (Near direct control of the HW=Hardware)
    - Makes it dangerous (Near direct control of the HW)
    - Most common in embedded devices (your phone, car, wireless router, etc.)
  - Principles and concepts of C/C++ will allow you to quickly learn other programming languages
    - C/C++ is extremely broad and allows you to learn other languages easily
  - Not Java

High Level Languages



http://www.digibarn.com/collections/posters/tongues/ComputerLanguagesChart-med.png

History of C and C++

Language	Originator	Year
Algol	International Group	1960
Basic Combined Programming Language (BCPL)	Martin Richards	1967
B	Ken Thompson	1970
C	Dennis Ritchie	1972
K&R C	Brian Kernighan & Dennis Ritchie	1978
C++	Bjarne Stroustrup	1980
C++-98 and 03	ISO/IEC 14882:1998/2003	1998, 2003
C++-11,14,17,20	ISO/IEC 14882:2011	2011 and onward

https://www.scaler.com/topics/cpp/origin-and-philosophy-of-cpp/

Philosophy of C++

- Compatibility
  - The language should be compatible with C.
  - The transition from C to C++ should not be difficult. *(yeah right!)*
- Choice (and Efficiency)
  - The language should give the option to programmers to make their own choice, even if it increases the possibility that a programmer will choose incorrectly.
  - The language should not slow down a program or consume space (overhead) for the features not used in the code.

- **Summary: Minimalist**
  - C and C++ force nothing extra upon you (even protections)
  - If you want "extra", implement it yourself or use a library

The Design and Evolution of C++, Bjarne Stroustrup

Quotes from Bjarne

- *"There are only two kinds of languages: the ones people complain about and the ones nobody uses".*
- *"C++ is designed to allow you to express ideas, but if you don't have ideas or don't have any clue about how to express them, C++ doesn't offer much help."*

- Bjarne Stroustrup (Quotes)

Fact 1: Everything is a number

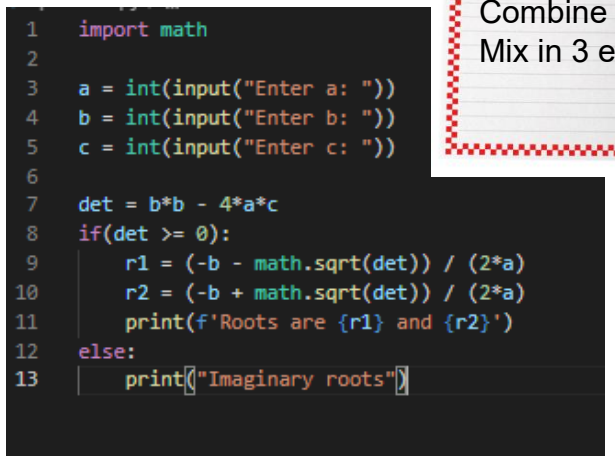
FACT 1 AND FACT 2

It's A Numbers Game

- Fact 1: Everything in a computer is a **number**
  - Sure. Things like 103 and 3.9 are numbers
  - But what about text and images and sound?
  - Everything!

- Fact 2: Computer operations can only work with or "see" 1 or 2 numbers at a time (i.e. they can only do 1 thing at a time)

- Humans process information differently
  - Therein lies some of the difficultly of learning programming



Example (1)

- What do you see?
  - The letter 'a'!

a

- What does the computer see?
  - A number; each text character is coded to a number
    - Example: Character map / Insert symbol

97

Text Representation

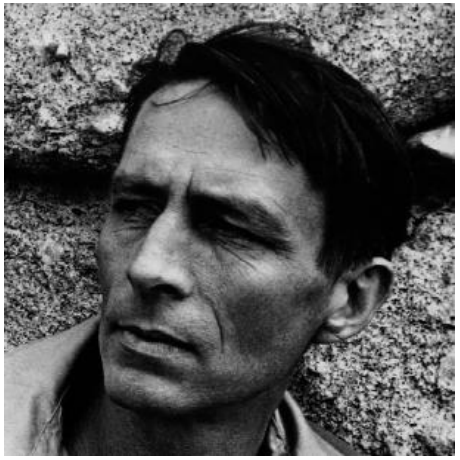
- Most common character code is ASCII (UTF-8)
- Every character, even non-printing, characters have a corresponding numbers
  - Decimal (base 10) / Hexadecimal (base 16)

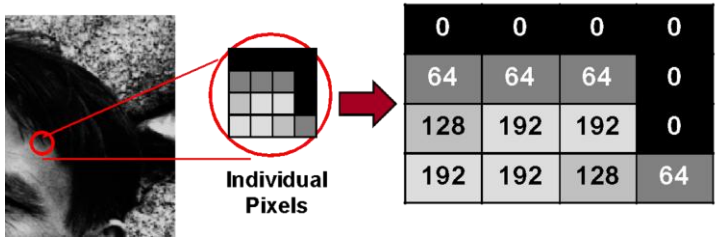
Dec	Hex	Name	Char	Ctrl-char	Dec	Hex	Char	Dec	Hex	Char	Dec	Hex	Char
0	0	Null	NUL	CTRL-@	32	20	Space	64	40	@	96	60	`
1	1	Start of heading	SOH	CTRL-A	33	21	!	65	41	A	97	61	a
2	2	Start of text	STX	CTRL-B	34	22	"	66	42	B	98	62	b
3	3	End of text	ETX	CTRL-C	35	23	#	67	43	C	99	63	c
4	4	End of xmit	EOT	CTRL-D	36	24	$	68	44	D	100	64	d
5	5	Enquiry	ENQ	CTRL-E	37	25	%	69	45	E	101	65	e
6	6	Acknowledge	ACK	CTRL-F	38	26	&	70	46	F	102	66	f
7	7	Bell	BEL	CTRL-G	39	27	'	71	47	G	103	67	g
8	8	Backspace	BS	CTRL-H	40	28	(	72	48	H	104	68	h
9	9	Horizontal tab	HT	CTRL-I	41	29	)	73	49	I	105	69	i
10	0A	Line feed	LF	CTRL-J	42	2A	*	74	4A	J	106	6A	j
11	0B	Vertical tab	VT	CTRL-K	43	2B	+	75	4B	K	107	6B	k
12	0C	Form feed	FF	CTRL-L	44	2C	,	76	4C	L	108	6C	l
13	0D	Carriage feed	CR	CTRL-M	45	2D	-	77	4D	M	109	6D	m
14	0E	Shift out	SO	CTRL-N	46	2E	.	78	4E	N	110	6E	n
15	0F	Shift in	SI	CTRL-O	47	2F	/	79	4F	O	111	6F	o
16	10	Data line escape	DLE	CTRL-P	48	30	0	80	50	P	112	70	p
17	11	Device control 1	DC1	CTRL-Q	49	31	1	81	51	Q	113	71	q
18	12	Device control 2	DC2	CTRL-R	50	32	2	82	52	R	114	72	r
19	13	Device control 3	DC3	CTRL-S	51	33	3	83	53	S	115	73	s
20	14	Device control 4	DC4	CTRL-T	52	34	4	84	54	T	116	74	t
21	15	Neg acknowledge	NAK	CTRL-U	53	35	5	85	55	U	117	75	u
22	16	Synchronous idle	SYN	CTRL-V	54	36	6	86	56	V	118	76	v
23	17	End of xmit block	ETB	CTRL-W	55	37	7	87	57	W	119	77	w
24	18	Cancel	CAN	CTRL-X	56	38	8	88	58	X	120	78	x
25	19	End of medium	EM	CTRL-Y	57	39	9	89	59	Y	121	79	y
26	1A	Substitute	SUB	CTRL-Z	58	3A	:	90	5A	Z	122	7A	z
27	1B	Escape	ESC	CTRL-[	59	3B	;	91	5B	[	123	7B	{
28	1C	File separator	FS	CTRL-\	60	3C	<	92	5C	\	124	7C	|
29	1D	Group separator	GS	CTRL-]	61	3D	=	93	5D	]	125	7D	}
30	1E	Record separator	RS	CTRL-^	62	3E	>	94	5E	^	126	7E	~
31	1F	Unit separator	US	CTRL-_	63	3F	?	95	5F	_	127	7F	DEL

https://www.commfront.com/pages/ascii-chart

Example 2

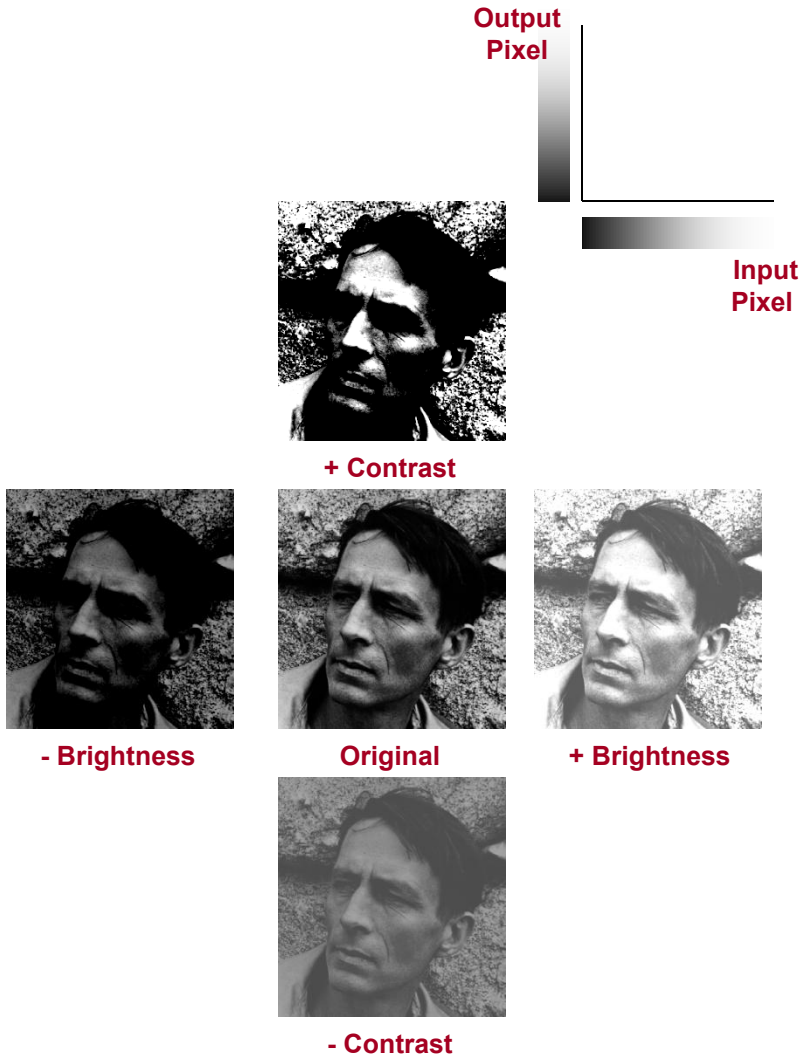
- What do you see?
  - A face!
- What does the computer see?
  - Individual pixels whose value indicates the shade of gray (or color)





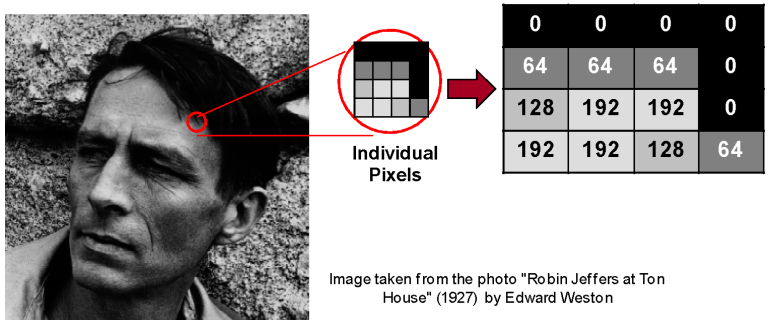
The Connection with Mathematics

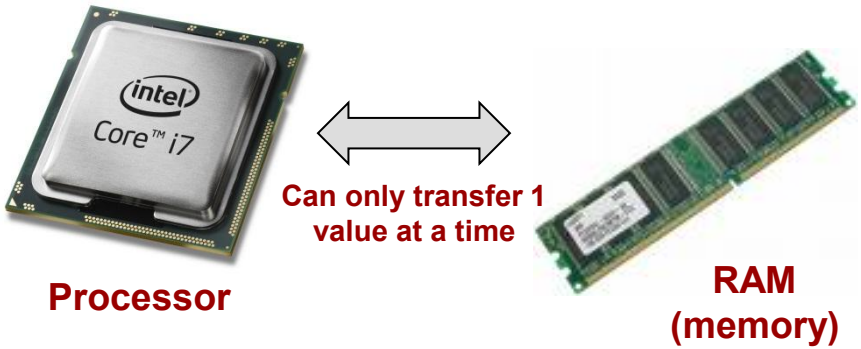
- Brightness
  - Each pixel value is increased/decreased by a constant amount
  - $P_{new} = P_{old} + B$
    - B > 0 = brighter
    - B < 0 = less bright
- Contrast
  - Each pixel value is multiplied by a constant amount
  - $P_{new} = C*P_{old} + k$
    - C > 1 = more contrast
    - 0 < C < 1 = less contrast
- Same operations performed on all pixels



But Why?

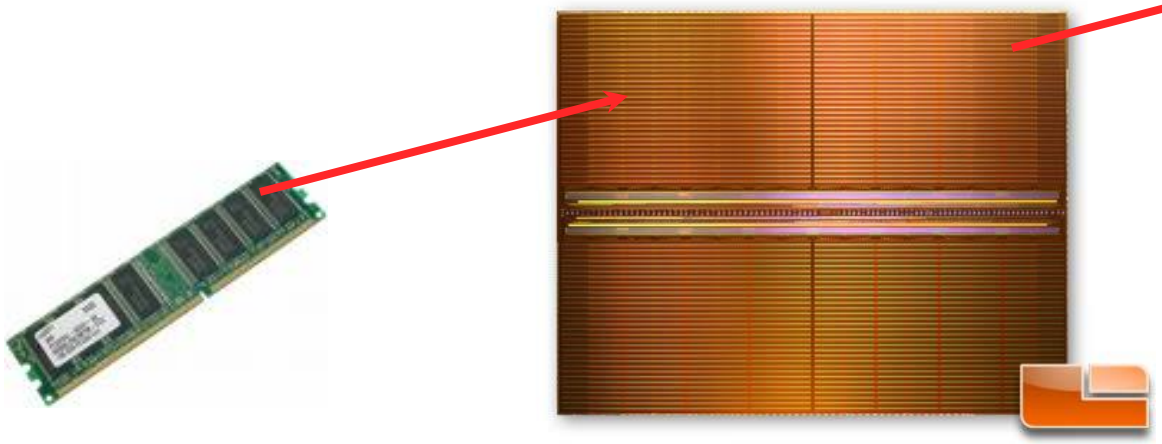
- Why can't computer just "look" at the image
  - Computers store information as numbers (e.g. each pixel of an image is a separate number)
  - These numbers are stored as units of 8-, 32- or 64-bits in the computer's RAM (memory)
  - The computer processor must retrieve them from memory 1 at a time





Memory

- Set of cells that each store a group of bits (usually, 1 byte = 8 bits)
- Unique address assigned to each cell
  - Used to reference the value in that location

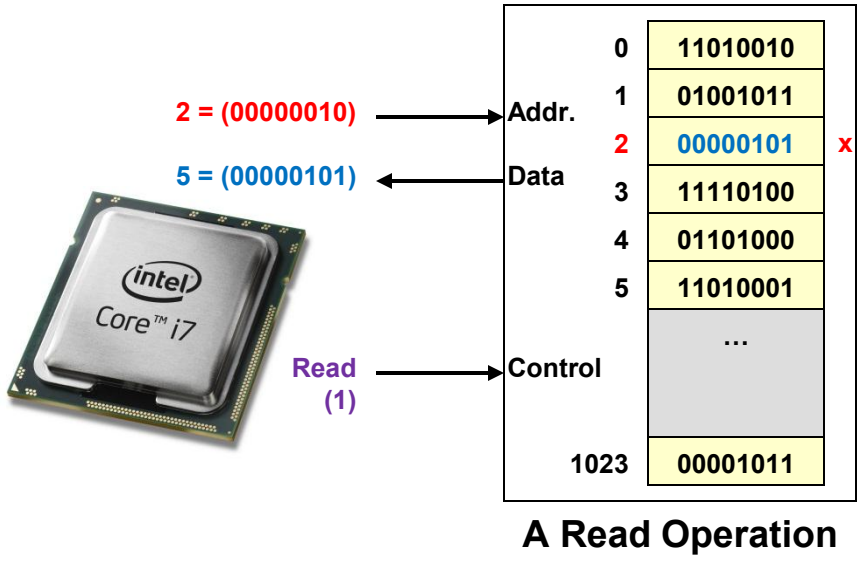


Address	Data
0	11010010
1	01001011
2	10010000
3	11110100
4	01101000
5	11010001
	…
1023	00001011

Memory Device

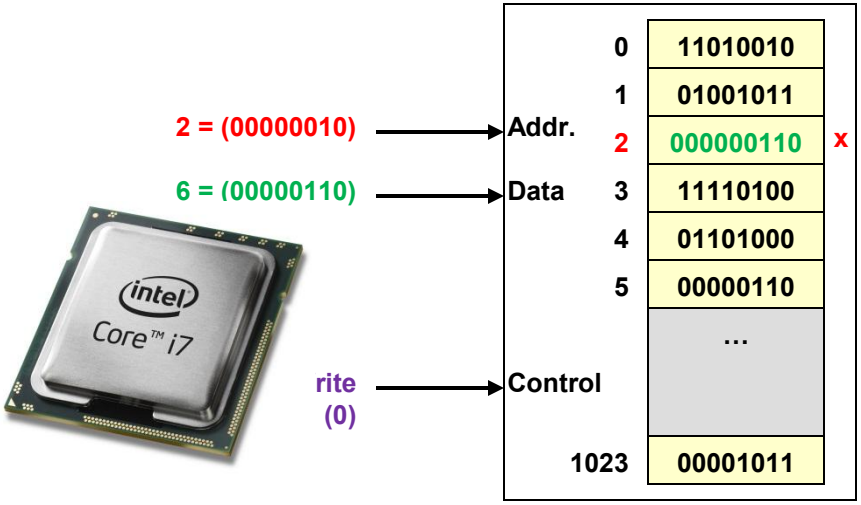
Memory Operations

- Processor can access 1 location at a time, either reading or writing that location
  - **Read**: retrieves data value in a particular location (specified using the address)
  - **Write**: changes data in a location to a new value



A Read Operation

- High level operation: x=x+1
  - Read x's current value from memory to the processor
  - Add 1
  - Write x's new value back to memory



A Write Operation

How many steps does it take a computer to perform the following tasks

1 OR MANY

One Operation or Many (1)

```
#include <iostream>
#include <string>
using namespace std;

// Execution starts at main()
int main()
{
  int x = 1, y = 2;

  x = y;

  return 0;
}
```

One / Many

```
#include <iostream>
#include <string>
using namespace std;

// Execution starts at main()
int main()
{
  string x = "abc", y = "defg";

  x = y;

  return 0;
}
```

One / Many

One Operation or Many (1)

```
#include <iostream>
#include <string>
using namespace std;

// Execution starts at main()
int main()
{
  int x = 1, y = 2;

  x = y;

  return 0;
}
```

One

```
#include <iostream>
#include <string>
using namespace std;

// Execution starts at main()
int main()
{
  string x = "abc", y = "defg";

  x = y;

  return 0;
}
```

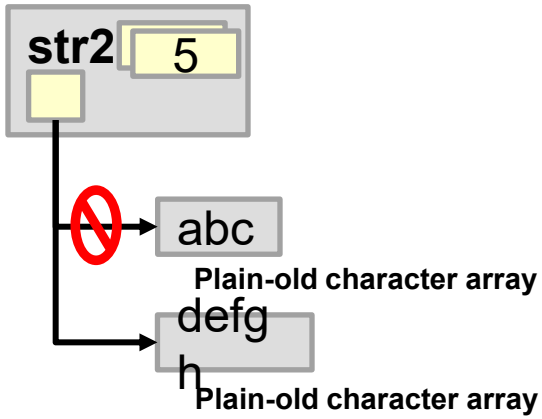
Many

`ints` are single objects to a computer
`strings` are composite objects (i.e. an array) to a computer

Strings Overview

- `strings` simply abstract character arrays
- Behind the scenes strings are just creating and manipulating character arrays but giving you a simplified set of operators and functions
- When the contents of the string grow too large, a new array is allocated behind the scenes (potentially copying the contents of the old string)

```
#include <iostream>
#include <string>
using namespace std;

int main()
{
  string str2 = "abc";
  // str2 stores 3 chars. = "abc"

  str2 = "defgh";
  // now str2 stores 5 characters

}
```



One Operation or Many (2)

```
#include <iostream>
#include <string>
using namespace std;

int main()
{
  char c = 'a', d = 'b';

  if(c == d) {
   ...
  }
  return 0;
}
```

One / Many

```
#include <iostream>
#include <string>
using namespace std;

int main()
{
  string x = "abc", y = "acb";

  if(x == y) {
   ...
  }
  return 0;
}
```

One / Many

One Operation or Many (2)

```
#include <iostream>
#include <string>
using namespace std;

int main()
{
  char c = 'a', d = 'b';

  if(c == d) {
   ...
  }
  return 0;
}
```

One

```
#include <iostream>
#include <string>
using namespace std;

int main()
{
  string x = "abc", y = "acb";

  if(x == y) {
   ...
  }
  return 0;
}
```

Many

`chars, doubles, ints` can be compared in a single operation
`strings` are arrays an must be compared character by character

One Operation or Many (3)

```
#include <iostream>
#include <string>
using namespace std;

int main()
{
  // Mimics an integer array
  vector<int> y = {1,2,3};

  y.push_back(4); // append 1 int

  return 0;
}
```

One / Many

One Operation or Many (3)

```
#include <iostream>
#include <string>
using namespace std;

int main()
{
  // Mimics an integer array
  vector<int> y = {1,2,3};

  y.push_back(4); // append 1 int

  return 0;
}
```

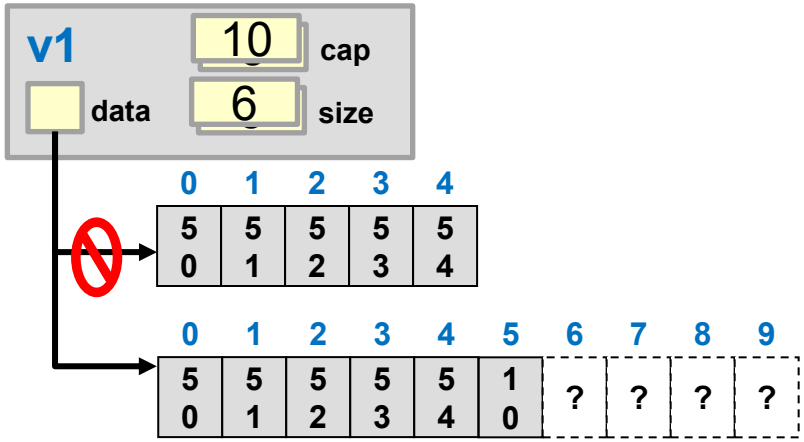
Many

Growing an array (which is the underlying structure of a vector) may require reallocating a larger array and copying the old contents

What Happens Behind the Scenes

- Vectors abstract arrays
  - Behind the scenes vectors are just creating and manipulating arrays but giving you a simplified set of operators and functions

```
#include <iostream>
#include <vector>
using namespace std;
int main()
{
  vector<int> v1(5);
  for(int i=0; i < 5; i++){
    v1[i] = i+50;
  }
  v1.push_back(10);
  // causes a resize behind the scenes
```



Who Am I

- Newest Teaching faculty in CS
- Undergrad at Cal Poly
- Grad at UC Riverside
  - PhD in VR
- Worked in software Eng. Before PhD
- Previously Visiting Faculty at
  Harvey Mudd College
- Guitarist, Banjoist.

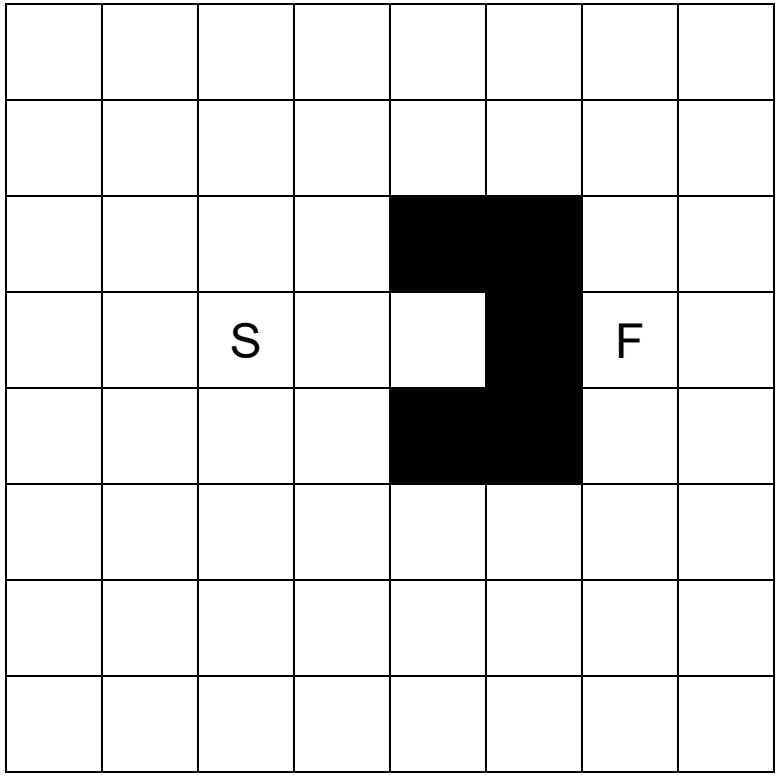
- Tabletop game connoisseur.

Fact 2: A computer operation can only process 1 or 2 values at a time

THINK LIKE A COMPUTER

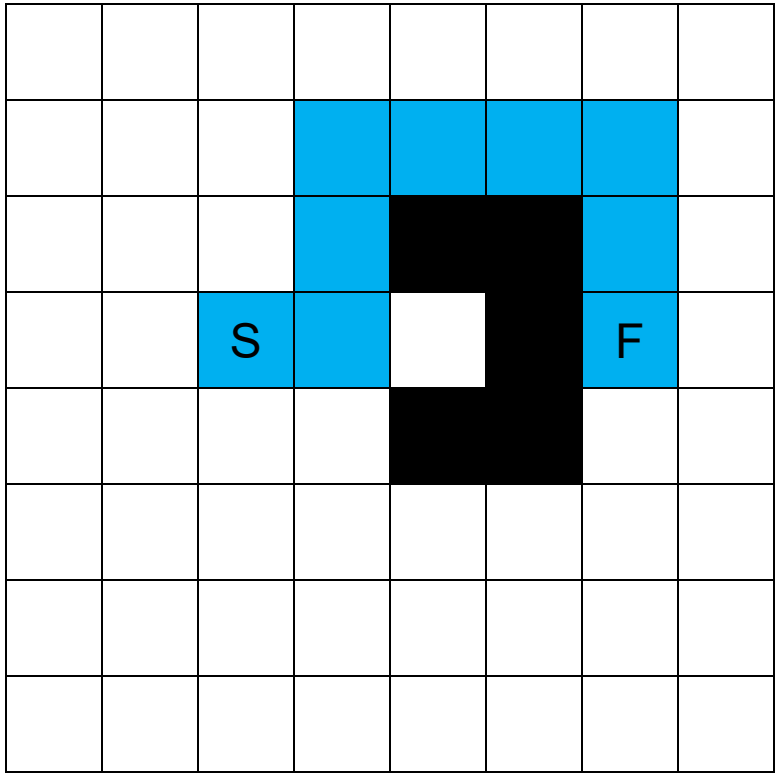
Path Planning

- Find shortest path from S to F



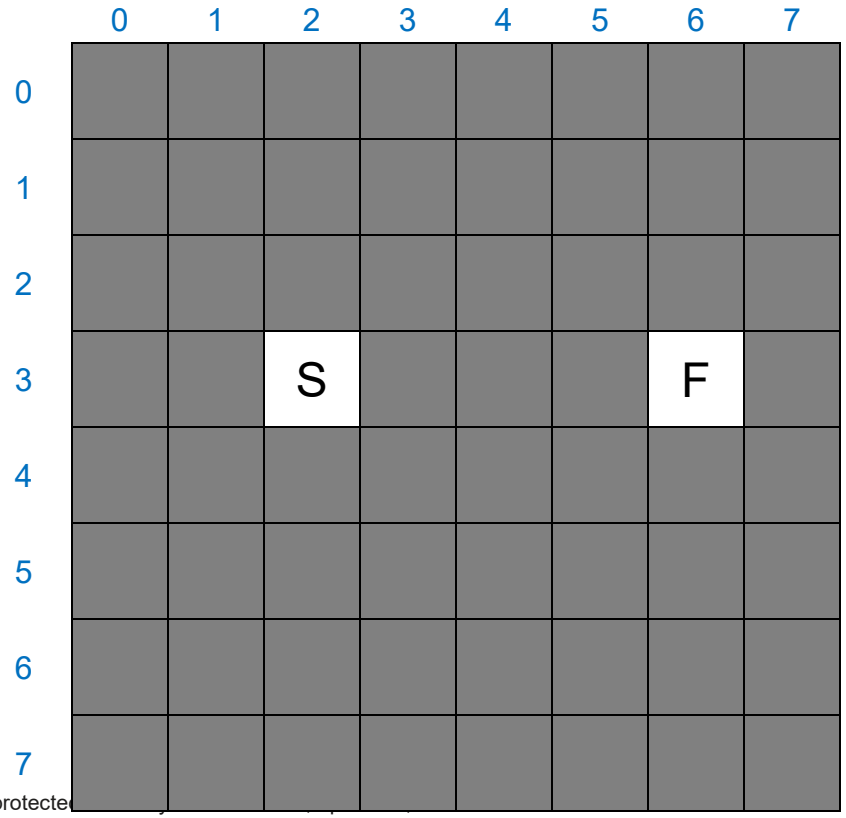
Path Planning

- Find shortest path from S to F



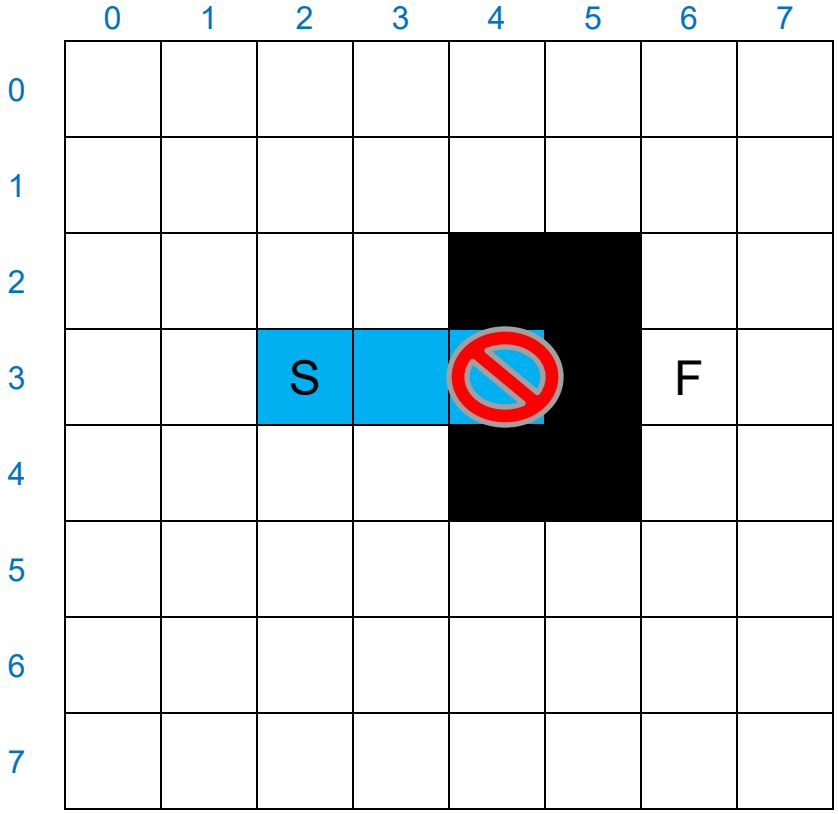
Path Planning

- Let's say you are a maps application and must find the appropriate path
- **Fact 2** (from earlier): A computer usually can only **process** (or "see") **one or two** data items (a square) at a time

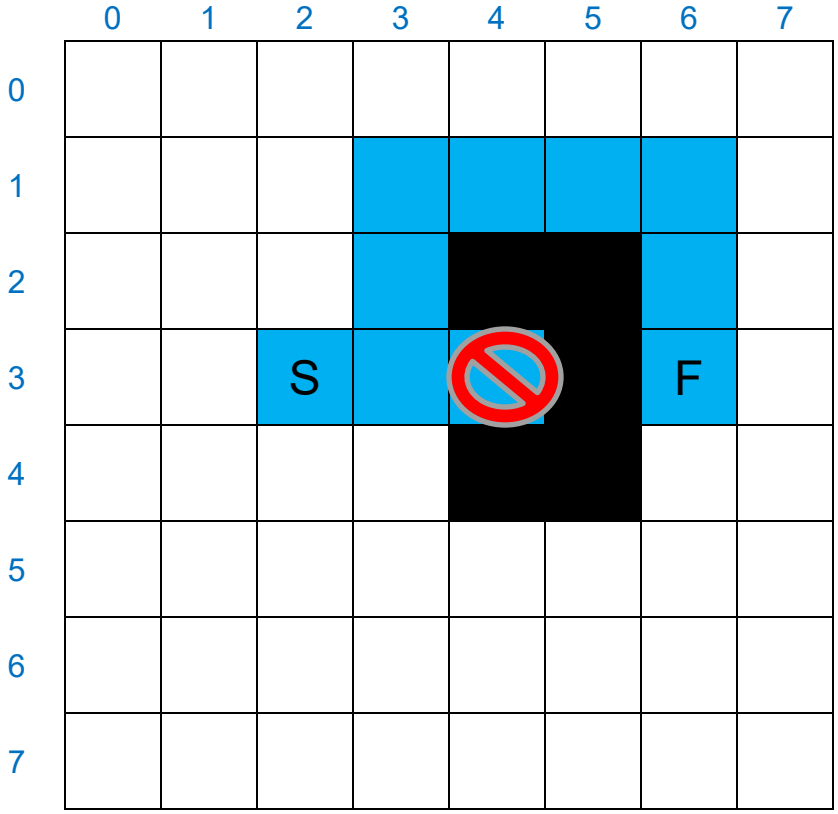


May just compute a straight line path from 'S' to 'F'

Path Planning

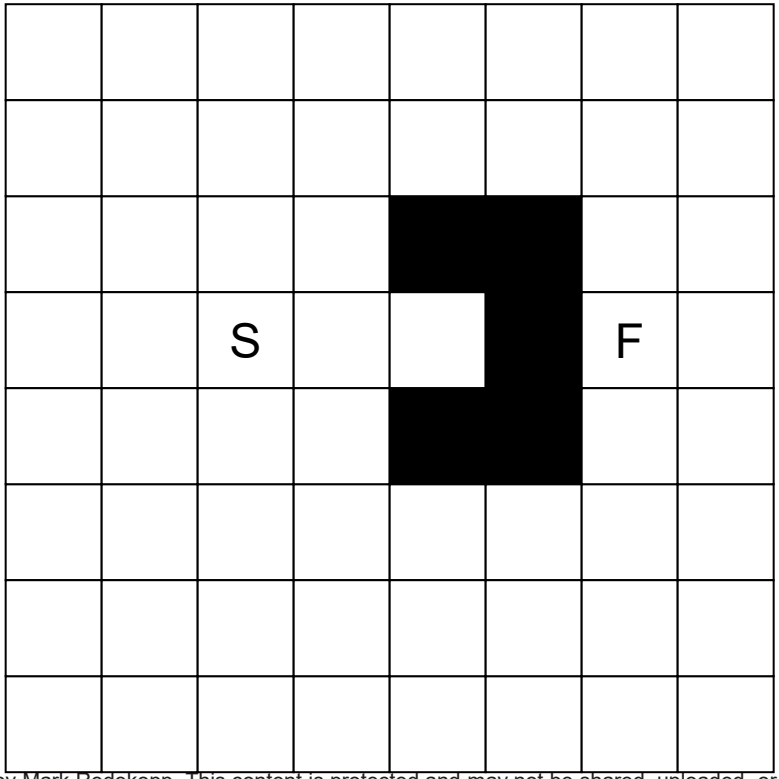


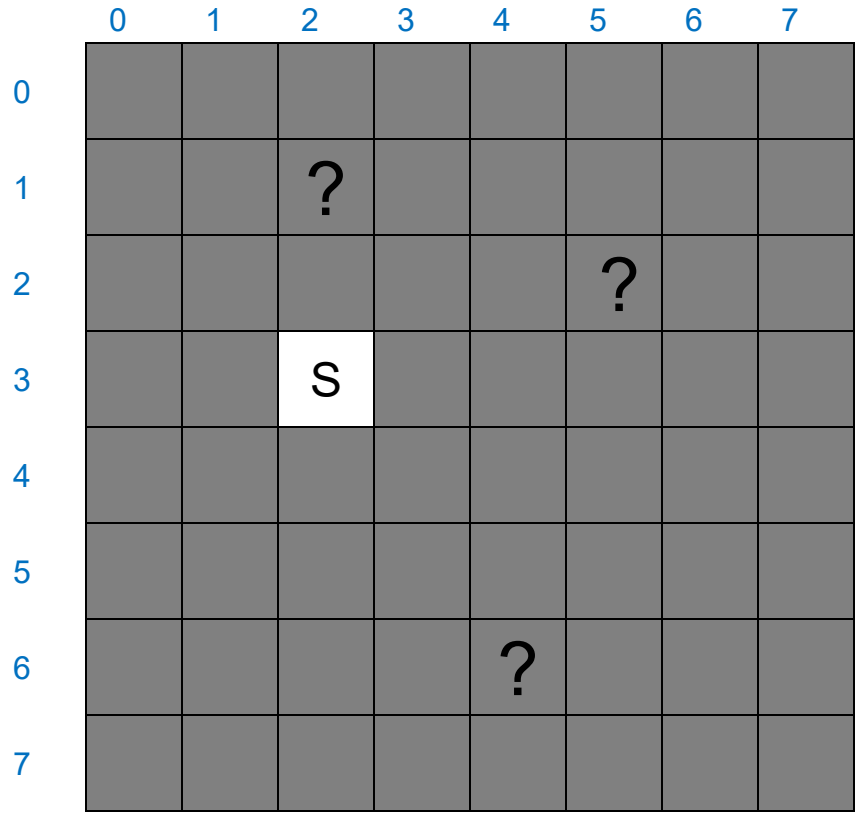
Path Planning



Path Planning

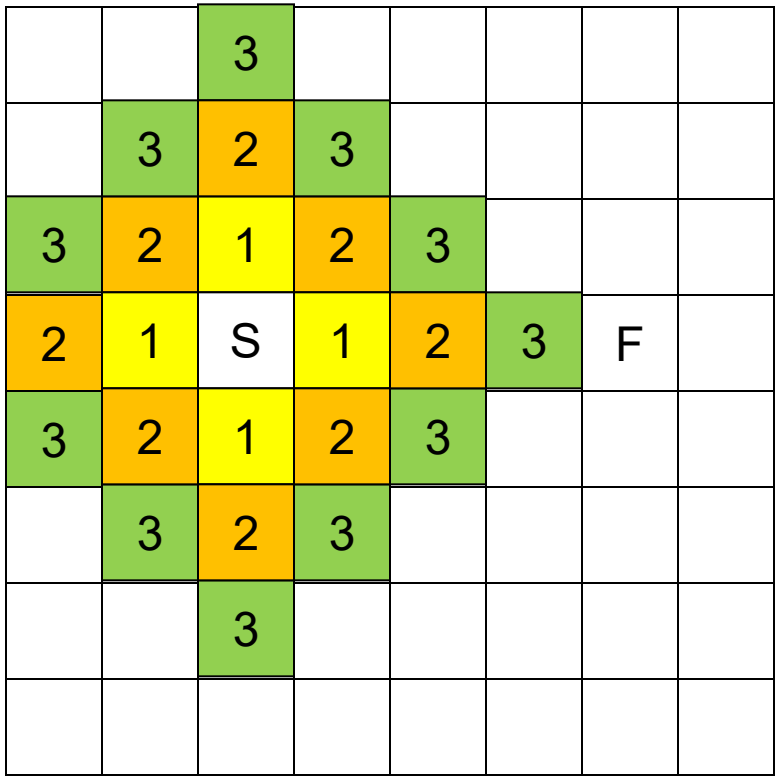
- What if I don't know where the Finish square is? You can examine any square in any order (no longer a robot) one at a time.
- In what order would you examine the locations to find the shortest path to the goal location?





Path Planning

- Examine all closer squares one at a time before progressing to further squares.



If you don't know where F is and want to find the shortest path, you should examine all closer locations before examining further-away locations.

Uninformed search for shortest path:

Breadth-first

A quick high-level view before we dive into the details…

OVERVIEW OF C++ PROGRAM STRUCTURE AND COMPILATION

Coming From Other Languages

Java

```java
import java.lang.Math.*;


class Hello {
  public static void printName(String name) {
    System.out.println(name + " Trojan");
  }

  // Execution starts here
  public static void main(String[] args)
  {
    System.out.println("Hello: ");
    printName("Tommy");
    printName("Tina");
    double e_x = Math.exp(1.0);
    System.out.println("e is " + e_x);
  }
}
```

```
$ javac Hello.java
$ java Hello
Hello:
Tommy Trojan
Tina Trojan
e is 2.71828
```

Python

```python
import math


def printName(name):
    print(name, "Trojan")


def main():
    print("Hello: ")
    printName("Tommy")
    printName("Tina")
    e_x = math.exp(1.0)
    print("e is", e_x)

# Execution starts here (weird)
if __name__ == "main":
    main()
```

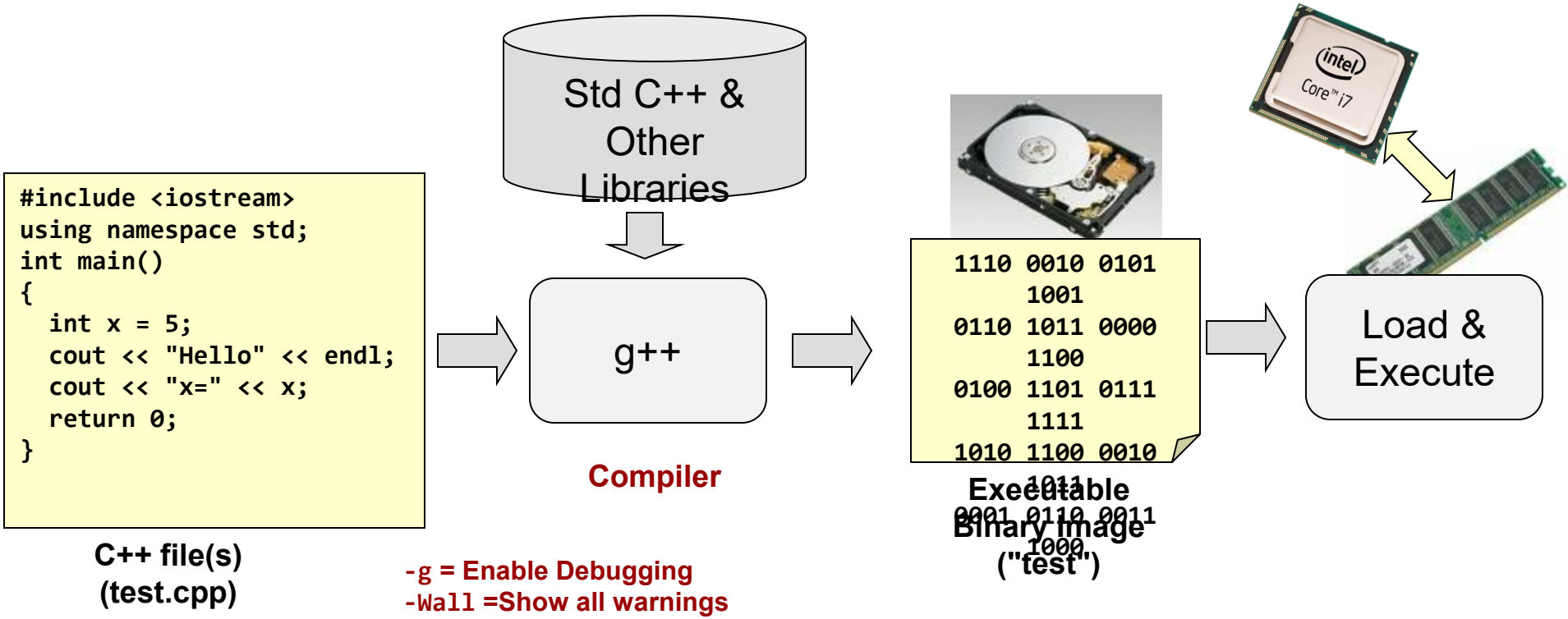
```
$ python3 hello.py
Hello:
Tommy Trojan
Tina Trojan
e is 2.71828
```

C++

```cpp
#include <iostream>
#include <cmath>
using namespace std;

void printName(string name)
{
  cout << name << " Trojan" << endl;
}
// Execution starts at main()
int main()
{
  cout << "Hello: " << endl;
  printName("Tommy");
  printName("Tina");
  double e_x = exp(1.0);
  cout << "e is " << e_x << endl;
  return 0;
}
```

```
$ g++ hello.cpp –o hello
$ ./hello
Hello:
Tommy Trojan
Tina Trojan
e is 2.71828
```

Compilation & Execution Process



-g = Enable Debugging
-Wall =Show all warnings
-o test = Specify Output executable name

```
$ g++ -g -Wall test.cpp -o test
```

```
$ g++ -g -Wall test.cpp -o test
$ ./test
```

1. Edit & write code
2. Compile & fix compiler errors
3. Load & run the executable program

C/C++ Program Format/Structure

- Comments
  - C-Style => "/*" and "*/"
  - C++ Style => "//"
- Compiler Directives
  - #includes tell compiler what other library functions you plan on using
  - 'using namespace std;' -- Just do it for now!
- main() function
  - Starting point of execution for the program
  - **All code/statements in C/C++ must be inside a function**
  - Statements execute sequentially (one line after the next) and **end with a semicolon (;)**
  - Ends with a 'return 0;' statement
- Other functions
  - Functions (and later, classes) are the primary unit of code organization (and problem decomposition/abstraction)
  - Enclosed by { and } (aka curly braces)

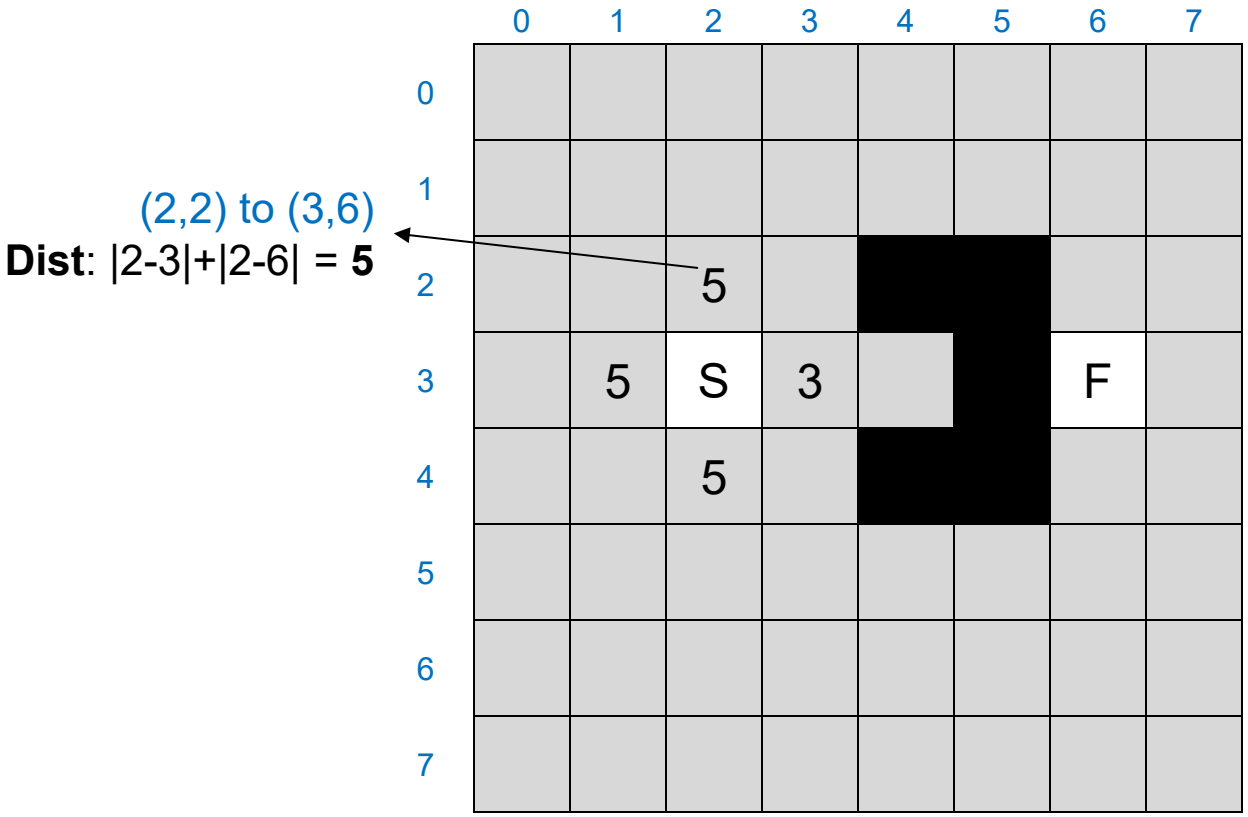
```
/* Anything between slash-star and
star-slash is ignored even across
multiple lines of text or code */

// Anything after "//" is ignored on a single line

// #includes allow access to library functions
#include <iostream> // cout and endl
#include <cmath>    // exp
using namespace std;

void printName(string name)
{
  cout << name << " Trojan" << endl;
}

// Execution always starts at the main() function
int main()
{
  cout << "Hello: " << endl;
  printName("Tommy");
  printName("Tina");
  double e_x = exp(1.0);
  cout << "e is " << e_x << endl;
  return 0;
}
```

```
$ g++ hello.cpp -o hello  # compile
$ ./hello                 # execute
Hello:
Tommy Trojan
Tina Trojan
e is 2.71828
```

BACKUP

Path Planning

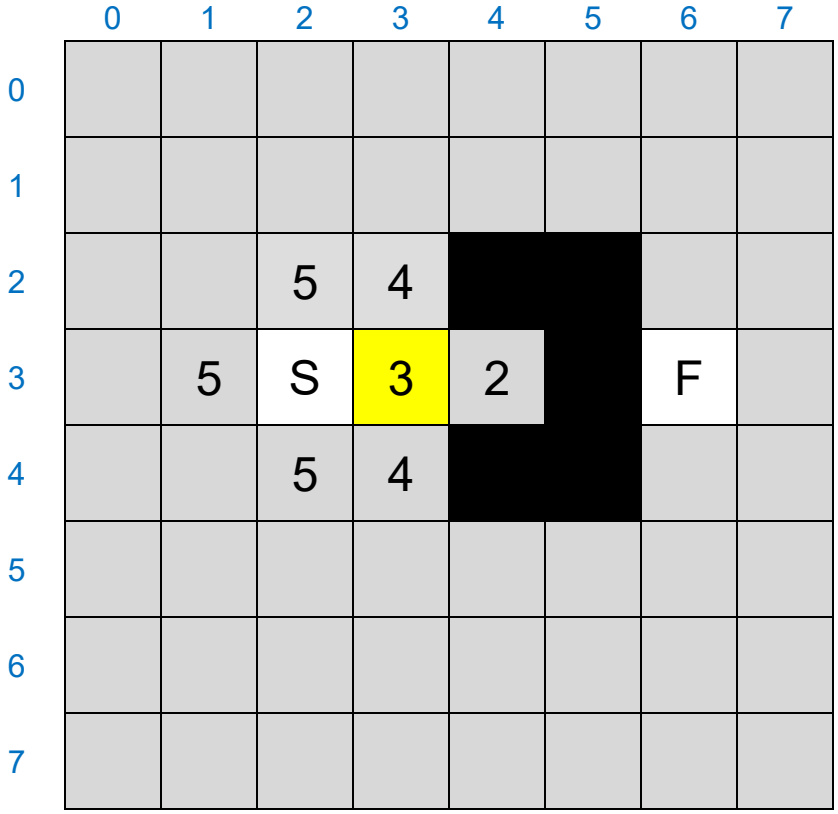
- You can't see obstacles until you are directly neighboring them.
- Now I tell you where the finish, F, location is. Can that help you reduce the number of squares explored?



Select a square to explore with minimum distance to the finish

Path Planning

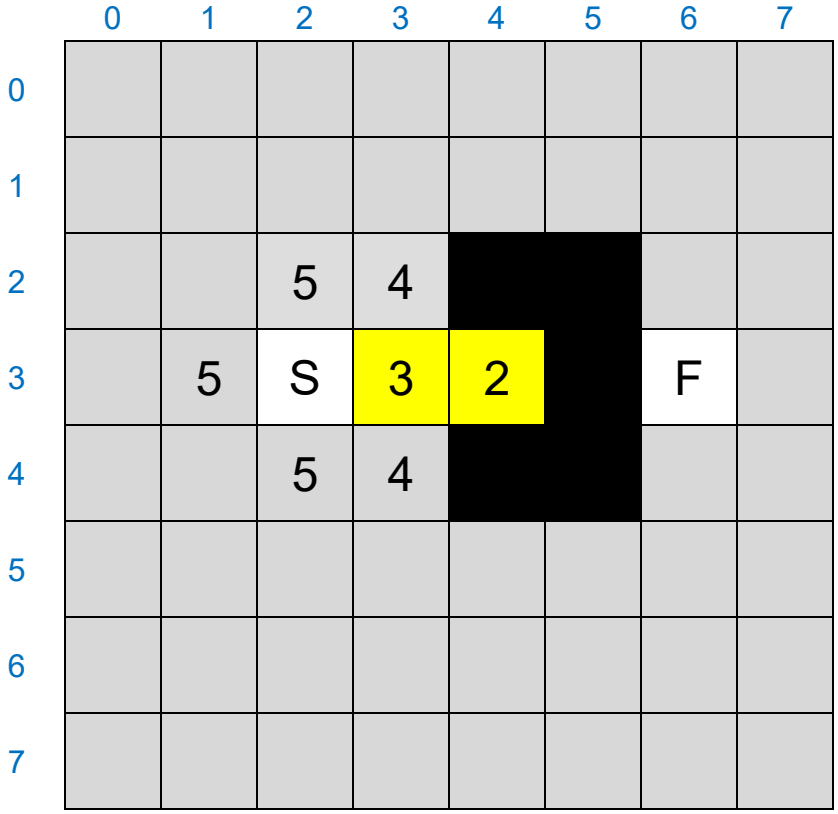
- You can't see obstacles until you are directly neighboring them.
- Now I tell you where the finish, F, location is. Can that help you reduce the number of squares explored?



Select a square to explore with minimum distance to the finish

Path Planning

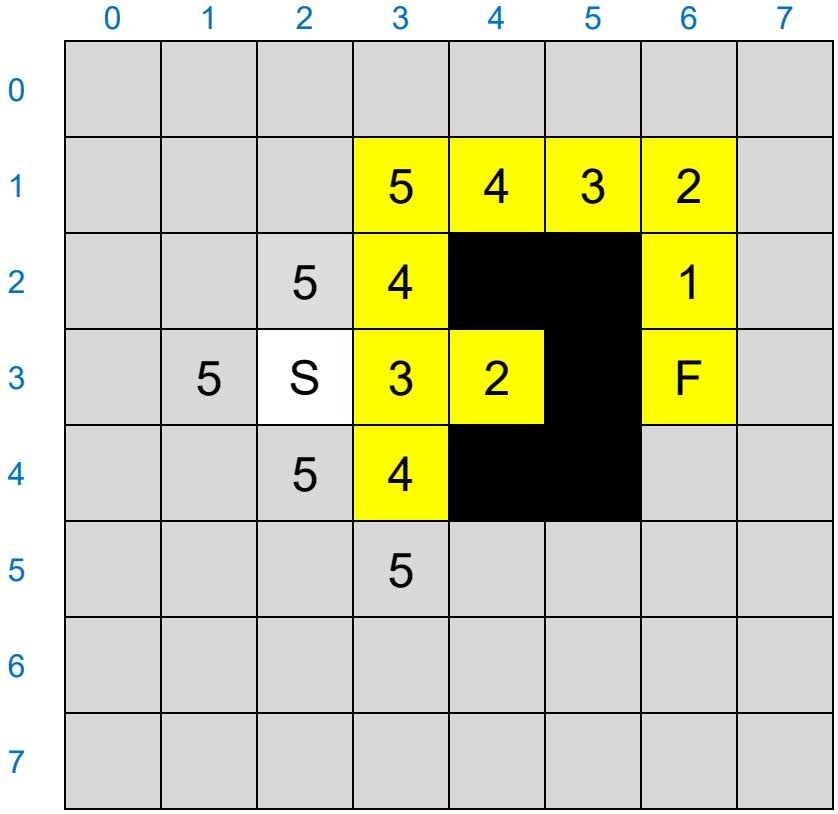
- But what if we run into a blockage?
  - Now we would pick the best among the remaining, unchosen locations.



Select a square to explore with minimum distance to the finish

Path Planning

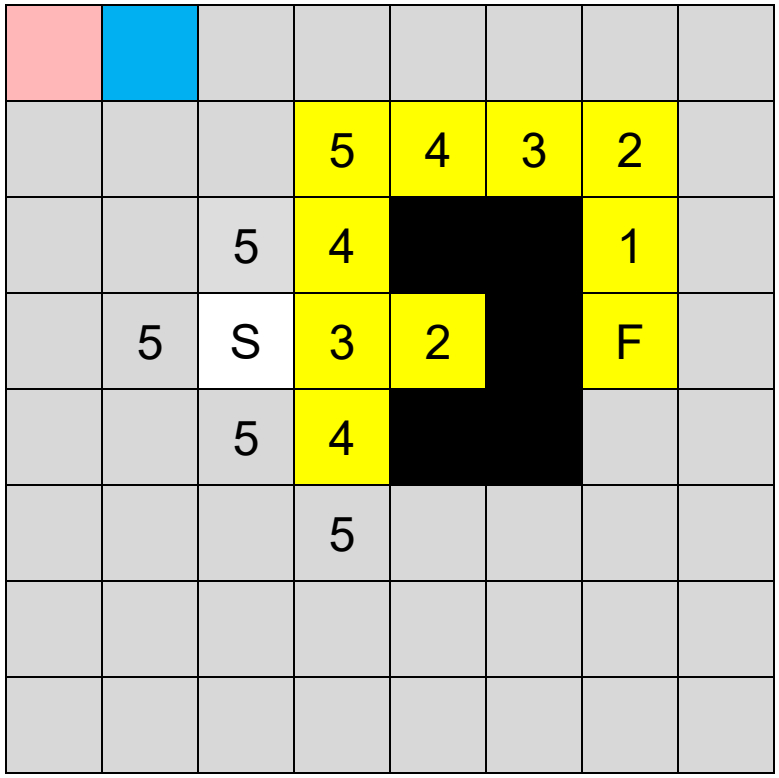
- But what if we run into a blockage?
  - Now we would pick the best among the remainder.

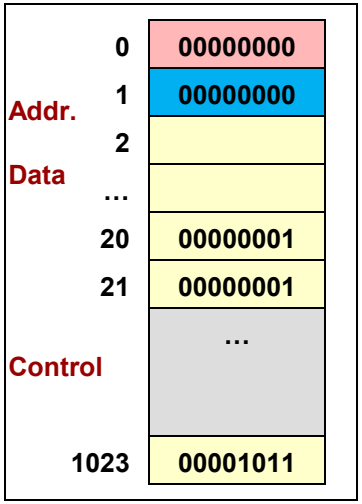


Select a square to explore with minimum distance to the finish

Programming vs. Algorithms

- Programming entails converting an algorithm into a specific process that a computer can execute





Another Example: Image Compression

- Images are just 2-D arrays (matrices) of numbers
- Each number corresponds to the color or a pixel in that location
- Image store those numbers in some way

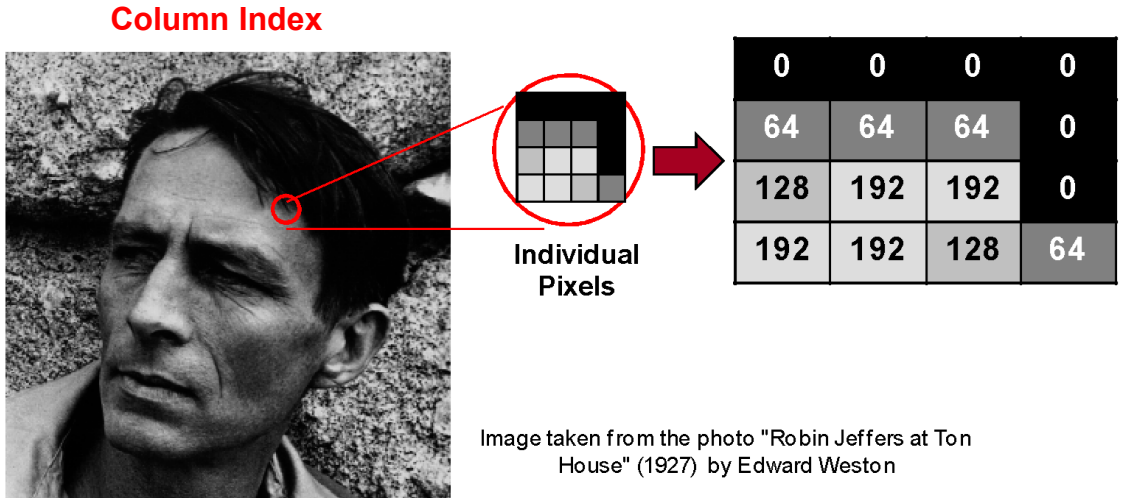
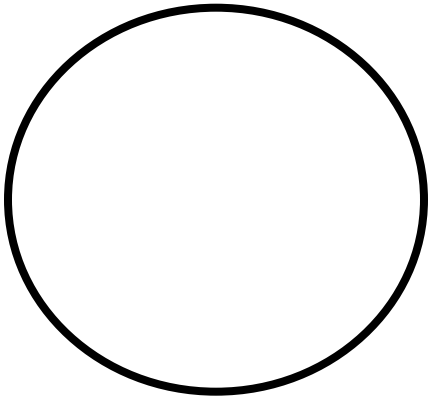
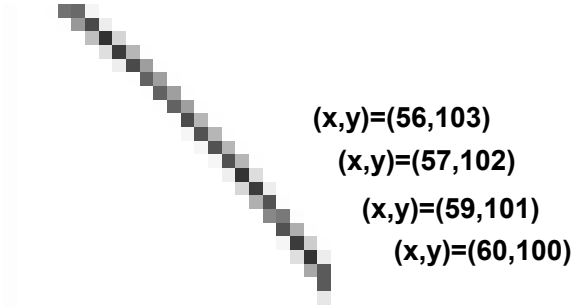
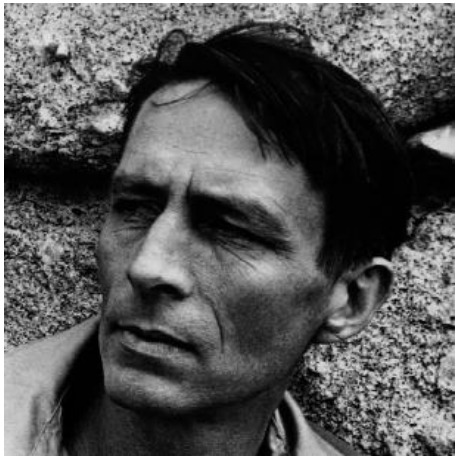


Image taken from the photo "Robin Jeffers at Ton House" (1927) by Edward Weston

Example 2 and 3







- What does the computer see?
  - Coordinate pairs of each "pixel"
  - …or…
  - r = 120; origin = (10, 14)
  - Computer has to enumerate and visit each location and color it black

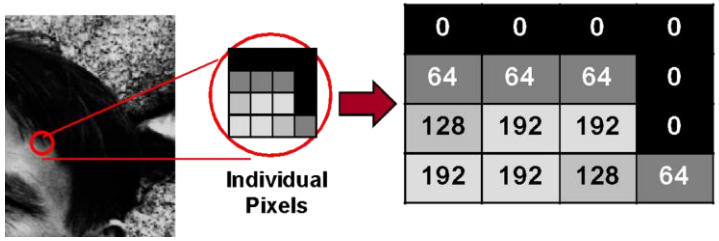
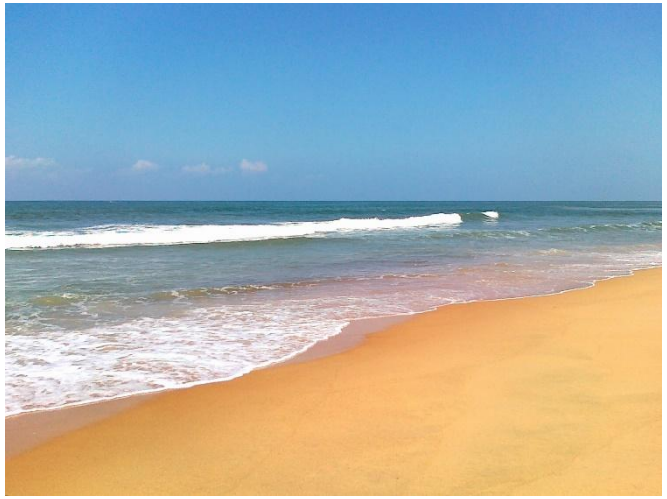


Image Compression



129	131	130	133	132	132	130	129	128	130	131	129
130	130	131	129	131	132	131	133	130	129	129	131
132	131	130	132								
134	132	131	132								
133	131										
156	157										
153	155										
154	152										
207	204										
208	205										

Image Compression

129	131	130	133	132	132	130	129	128	130	131	129
130	130	131	129	131	132	131	133	130	129	129	131
132	131	130	132								
134	132	131	132								
133	131										
156	157										
153	155										
154	152										
207	204										
208	205										

1. Break Image into small blocks of pixels

129	131	130	133
130	130	131	129
132	131	130	132
134	132	131	132

129	2	1	4
2	1	2	0
3	2	1	3
5	3	2	3

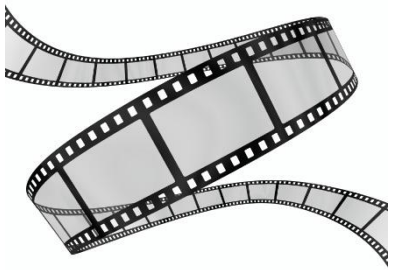
129	2	1	4
2	1	2	0
3	2	1	3
5	3	2	3

129	2	0	4
2	0	2	0
2	2	0	2
4	2	2	2

2. Store the difference of each pixel and the upper left (or some other representative pixel)

3. We can save more space by rounding numbers to a smaller set of options (i.e. only even # differences)

Video Compression



- Video is a sequence of still frames
  - 24-30 frames per second (fps)
- How much difference is expected between frames?
- Idea:
  - Store 1 of every K frames, with other K-1 frames being differences from frame 1 or from previous frame