

CS50 Section 2

Somewhere in Between

Annaleah Ernst, TF

The Agenda...

Housekeeping

- ▶ Introductions!
- ▶ About section
- ▶ Grading
- ▶ Resources
- ▶ Office Hours

..and then down to business

- ▶ Quick recap of super-section
- ▶ Debugging
- ▶ Arrays
- ▶ Functions
- ▶ Command line arguments
- ▶ Pset 2 - ASCII and modulo (%)

Introductions!

- ▶ Me! (Your TF)
 - ▶ Annaleah Ernst (feel free to call me by my first name)
 - ▶ Email: annaleahernst@college.harvard.edu
 - ▶ Phone: <redacted>
- ▶ And now you guys...
 - ▶ Name, Hometown, and...
 - ▶ If you could be any mythical creature what would you be?

About section

- ▶ Hands on experience
- ▶ Time for questions/clarifications on lecture material
- ▶ Meet me halfway! Prepare by...
 - ▶ Watching the lectures
 - ▶ Read the pset spec & think about the problems
- ▶ Pencil/paper practice recommended

Grading

- ▶ Submit everything (and try everything!)
 - ▶ 9 psets, 2 quizzes, and final project
- ▶ $\text{grade} = \text{scope} * (3 * \text{correctness} + 2 * \text{design} + 1 * \text{style})$
- ▶ Grading Breakdown:
 - ▶ Problem sets: 50%, Quizzes: 40%, Final project: 10%
- ▶ **ULTIMATELY, COMMENTS ARE WHAT MATTER**
- ▶ “...what ultimately matters in this course is not so much where you end up relative to your classmates but where you, in Week 12, end up relative to yourself in Week 0.” - the syllabus
- ▶ NOT curved, NO predetermined cut offs for final grades, sections normalized

Resources

- ▶ Static resources (ie, on your own)
 - ▶ CS50 Study - study.cs50.net
 - ▶ CS50 Manual - manual.cs50.net
 - ▶ Reference50 - reference.cs50.net
 - ▶ Style Guide - manual.cs50.net/style/
 - ▶ Walkthroughs && Shorts
 - ▶ `help50`, `check50`, `style50`, `debug50`
- ▶ Dynamic Resources (ie, involving interaction with others)
 - ▶ CS50 Discuss - cs50.harvard.edu/discuss
 - ▶ Office hours
 - ▶ Classmates
 - ▶ Me!

Office Hours

- ▶ Wednesday: 9 - 11, Widener
- ▶ Thursday: 9 - 11, Widener or Northwest
- ▶ Sunday: 3 - 5, SOCH or Northwest
- ▶ 7 days a week at HSA (check the calendar)

Let's get coding!



Review

▶ Linux Commands

- ▶ Common: `pwd`, `cd`, `ls`, `mkdir`, `rm`, `cp`, `mv`
- ▶ Less common: `>`, `<`, `|`, `grep`

▶ Data types

- ▶ `int`, `float`, `double`, `long long`, `short`, `char`, `string (char*)`, `bool`, **pointer**

▶ Math and Logic

- ▶ Operators: `+`, `-`, `*`, `/`, `%`
- ▶ Operator Precedence: PEMDAS
- ▶ Floating point vs Integer math
- ▶ Boolean expressions: `==`, `!=`, `&&`, `||`

Review (cont)

- ▶ Conditionals
 - ▶ Switch statements
 - ▶ Ternary operator
 - ▶ `condition ? (if true do this) : (else if false do this);`
- ▶ Loops
 - ▶ `while`, `for`, `do-while`
- ▶ Functions
 - ▶ `<return_type> <function_name>(<function_parameters>)`
 - ▶ Eg, `int main(void)`
 - ▶ Include prototypes for functions you write above `main!`

Magic Numbers

- ▶ Magic numbers are unexplained numbers in your code
- ▶ Is there a magic number here?

```
for (int i = 0, n = strlen(text); i < n; i++)
```

- ▶ No - starting a counter at 0 makes intuitive sense
- ▶ What about here?

```
for (int i = 5, n = strlen(text); i < n; i++)
```

- ▶ Yes - what does this 5 signify? I have no idea.
- ▶ Two solutions:
 - ▶ Use variables if the value will change
 - ▶ Use `#define <NAME> <value>` for constants
 - ▶ This goes at the top of the file right after you `#includes`, eg
 - ▶ `#define LEN_ALPHA 26`

Debugging: help50

- ▶ step 0: help50
 - ▶ convert cryptic output into something more user friendly
 - ▶ great for compiler errors
 - ▶ EX: make debug_output

```
clang -ggdb3 -O0 -std=c11 -Wall -Werror -Wshadow debug_output.c  
-lcs50 -lm -o debug_output  
debug_output.c:1:10: fatal error: 'studio.h' file not found  
#include <studio.h>  
      ^  
1 error generated.  
make: *** [debug_output] Error 1
```

- ▶ help50 make debug_output adds

```
Looks like you're trying to `#include` a file (`studio.h`) on line 1 of `debug_output.c`  
which does not exist. Did you mean to `#include <stdio.h>` (without the `u`)?
```

Debugging: eprintf

- ▶ eprintf

- ▶ included in the cs50 library (`#include "cs50.h"`)
- ▶ print information while debugging
 - ▶ we do this while we're trying to figure out what's wrong with a problem set
- ▶ provides automatic context - which file, which line of code
- ▶ calls to eprintf should be removed before pset submission
- ▶ EX: Greedy
 - ▶ check50 is returning all green...except for this case

```
:( input of 4.2 yields output of 18
  \ expected output, but not "22\n"
```

- ▶ Now what? Let's walk through it in the ide

Debugging: eprintf, cont

- ▶ Let's use eprintf verify that we're getting the inputs we expect

```
...  
  
float dollars;  
do  
{  
    printf("Ohai! How much change is owed?\n");  
    dollars = get_float();  
}  
while (dollars < 0);  
  
int cents = dollars * DOLLARS_TO_CENTS;  
  
// add a temporary debug statement to verify inputs  
eprintf("dollars: %f\n", dollars);  
eprintf("cents: %i\n", cents);  
  
...
```

- ▶ What does this output?

```
Ohai! How much change is owed?  
4.2  
greedy:greedy.c:24: dollars: 4.200000  
greedy:greedy.c:25: cents: 419  
22
```

- ▶ cents is 419 when it should be 420! we must be dealing with floating point errors

Debugging: debug50

- ▶ debug50
 - ▶ run program line by line to pinpoint bugs
 - ▶ set breakpoints in you .c file
 - ▶ lines of code you want to stop on to look at variables, test values, etc
 - ▶ run debug50 with `debug50 ./my_program`
 - ▶ step into
 - ▶ brings you out of main an into a function
 - ▶ step over
 - ▶ treats the function as a black box and moves to the next line
 - ▶ mouse-over variables to see their values
 - ▶ EX: buggy.c

Debugging: Duck Debugging

- ▶ Explain your code to a rubber duck
 - ▶ No, but actually
 - ▶ Often, talking through your code helps you find bugs



Arrays

- ▶ Data structure
 - ▶ Stores data in one place in memory
 - ▶ Can store pieces of data of the same type
- ▶ Declaration
 - ▶ `<datatype> <name>[<size>]`
 - ▶ Eg, `char alpha[26]`
 - ▶ Question: How could I declare an int array?
- ▶ Initialization

Arrays - Declaration

- ▶ `<datatype> <name>[<size>]`
- ▶ How would I declare an array...
 - ▶ called scores of three integers?
 - ▶ `int scores[3];`
 - ▶ called floaty of six floats?
 - ▶ `float floaty[6];`
 - ▶ called alpha of twenty six chars?
 - ▶ `char alpha[26];`

Array - initializing

```
int scores[3];  
scores[0] = 6;  
scores[1] = 5;  
scores[2] = 4;
```

- ▶ What happens if I try to put something at scores[3]?
- ▶ Alternative method for initializing:

```
int scores[] = {6, 5, 4};  
// OR  
int scores[3] = {6, 5, 4};
```

Arrays - iterating

```
// what's wrong with this code?  
int scores[3] = {6, 5, 4};  
for (int i = 0; i <= 3; i++)  
{  
    printf("%i\n", scores[i]);  
}
```

- ▶ How do we fix this?

Arrays - iterating

```
int scores[3] = {6, 5, 4};  
for (int i = 0; i < 3; i++)  
{  
    printf("%i\n", scores[i]);  
}
```

Arrays - your turn!

count.c

- ▶ Write a program that ...
 - ▶ Creates an array containing the integers 1 to 5
 - ▶ Iterates through the array and prints one number per line

Strings

- ▶ These are just special arrays characters!
- ▶ Last box reserved for null
- ▶ `string s = "ohai";`
- ▶ Is equivalent to...
- ▶ `Char s[] = {'o', 'h', 'a', 'i', '\0'};`
- ▶ Example: arrays.c

Your turn: spell.c

- ▶ Write a program that...
 - ▶ Asks the user for a string
 - ▶ Prints out each char on a new line
- ▶ Don't forget to `#include <string.h>`

Functions

- ▶ Black boxes
- ▶ Take things in (parameters)
- ▶ Do something (side effect)
- ▶ Spit something out (return value)
- ▶ Why use functions?
 - ▶ Organization
 - ▶ Simplification
 - ▶ Reusability
- ▶ Remember to declare prototypes above main

Functions

```
<return type> <name>(<parameter list>)  
{  
    <code>  
}
```

Functions: main

```
int main(void)
{
    printf("ohai \n");
    return 0;
}
```

Functions: scope

- ▶ Every variable has scope
- ▶ I.e., Where the variable may be referenced
 - ▶ Eg, i in a for loop
- ▶ “What happens in braces stays in braces.”

```
int a;  
  
int main(void)  
{  
    int a;  
    a = 5;  
}
```

Your turn! - hello.c

- ▶ Write a program in which...
 - ▶ main calls another function that prints out a greeting to the user

Function declaration

```
// protoype - this is what's important
void hello(void);
int main(void)
{
    hello();
}
void hello(void)
{
    printf("hello world!");
}
```

Command-line arguments

- ▶ One way to pass information into a program!
- ▶ `int main(void)` becomes...
- ▶ `int main(int argc, string argv[])`
- ▶ `argc` - “argument count” (# of arguments)
- ▶ `argv[]` - “argument vector” (arguments themselves)
- ▶ Example:
 - ▶ `./ohai cs50 section`
 - ▶ `argc` is 3
 - ▶ `argv[0]` is “ohai”
 - ▶ `argv[1]` is “cs50”
 - ▶ `argv[2]` is “section”
- ▶ `argv[0]` is always the name of the program
- ▶ careful! `argv` is ALWAYS an array of strings

Multidimensional Arrays

- ▶ Arrays of arrays -> rows and columns
- ▶ How does this relate to command line arguments?
 - ▶ Argv is an array of strings...what do we know about strings?
- ▶ Really, argv = array of arrays
- ▶ Back to previous example:
 - ▶ `./ohai cs50 section`
 - ▶ `argv[1]` is “cs50”
 - ▶ `argv[1][2]` is ‘5’

Your turn

- ▶ Modify hello.c such that that...
 - ▶ Takes a user's name as command line args
 - ▶ two and only two names (first and last) may be given to the program
 - ▶ Print out a greeting using that user's first name

Pset 2 - Crypto

- ▶ Caesar cipher
 - ▶ Rotate values in a target word
 - ▶ eg, “Annaleah” rotated by 2 becomes “Cppcngj”
 - ▶ What happens if I rotate by 100?
 - ▶ The solution: use %
- ▶ Vigenere cipher
 - ▶ Use a key to encrypt a word
 - ▶ eg, “Annaleah” encrypted by “hi” becomes “Hvuismhp”
 - ▶ What if the letters in the target word aren’t divisible by the key?
 - ▶ The solution is still to use mod

Pset 2 - considerations

- ▶ Why might we need modulo?
- ▶ What data type are the contents of `argv[]`?
 - ▶ How do we convert them to int?
 - ▶ `atoi()`
- ▶ What do we know about how characters are represented?
 - ▶ ASCII - <http://www.asciitable.com/>
 - ▶ Eg, 'A' is 65, 'a' is 97

ASCII

- ▶ characters can be represented as numbers

Dec	Hx	Oct	Char	Dec	Hx	Oct	Html	Chr	Dec	Hx	Oct	Html	Chr	Dec	Hx	Oct	Html	Chr
0	0	000	NUL (null)	32	20	040	 	Space	64	40	100	@	@	96	60	140	`	`
1	1	001	SOH (start of heading)	33	21	041	!	!	65	41	101	A	A	97	61	141	a	a
2	2	002	STX (start of text)	34	22	042	"	"	66	42	102	B	B	98	62	142	b	b
3	3	003	ETX (end of text)	35	23	043	#	#	67	43	103	C	C	99	63	143	c	c
4	4	004	EOT (end of transmission)	36	24	044	$	\$	68	44	104	D	D	100	64	144	d	d
5	5	005	ENQ (enquiry)	37	25	045	%	%	69	45	105	E	E	101	65	145	e	e
6	6	006	ACK (acknowledge)	38	26	046	&	&	70	46	106	F	F	102	66	146	f	f
7	7	007	BEL (bell)	39	27	047	'	'	71	47	107	G	G	103	67	147	g	g
8	8	010	BS (backspace)	40	28	050	((72	48	110	H	H	104	68	150	h	h
9	9	011	TAB (horizontal tab)	41	29	051))	73	49	111	I	I	105	69	151	i	i
10	A	012	LF (NL line feed, new line)	42	2A	052	*	*	74	4A	112	J	J	106	6A	152	j	j
11	B	013	VT (vertical tab)	43	2B	053	+	+	75	4B	113	K	K	107	6B	153	k	k
12	C	014	FF (NP form feed, new page)	44	2C	054	,	,	76	4C	114	L	L	108	6C	154	l	l
13	D	015	CR (carriage return)	45	2D	055	-	-	77	4D	115	M	M	109	6D	155	m	m
14	E	016	SO (shift out)	46	2E	056	.	.	78	4E	116	N	N	110	6E	156	n	n
15	F	017	SI (shift in)	47	2F	057	/	/	79	4F	117	O	O	111	6F	157	o	o
16	10	020	DLE (data link escape)	48	30	060	0	0	80	50	120	P	P	112	70	160	p	p
17	11	021	DC1 (device control 1)	49	31	061	1	1	81	51	121	Q	Q	113	71	161	q	q
18	12	022	DC2 (device control 2)	50	32	062	2	2	82	52	122	R	R	114	72	162	r	r
19	13	023	DC3 (device control 3)	51	33	063	3	3	83	53	123	S	S	115	73	163	s	s
20	14	024	DC4 (device control 4)	52	34	064	4	4	84	54	124	T	T	116	74	164	t	t
21	15	025	NAK (negative acknowledge)	53	35	065	5	5	85	55	125	U	U	117	75	165	u	u
22	16	026	SYN (synchronous idle)	54	36	066	6	6	86	56	126	V	V	118	76	166	v	v
23	17	027	ETB (end of trans. block)	55	37	067	7	7	87	57	127	W	W	119	77	167	w	w
24	18	030	CAN (cancel)	56	38	070	8	8	88	58	130	X	X	120	78	170	x	x
25	19	031	EM (end of medium)	57	39	071	9	9	89	59	131	Y	Y	121	79	171	y	y
26	1A	032	SUB (substitute)	58	3A	072	:	:	90	5A	132	Z	Z	122	7A	172	z	z
27	1B	033	ESC (escape)	59	3B	073	;	;	91	5B	133	[[123	7B	173	{	{
28	1C	034	FS (file separator)	60	3C	074	<	<	92	5C	134	\	\	124	7C	174	|	
29	1D	035	GS (group separator)	61	3D	075	=	=	93	5D	135]]	125	7D	175	}	}
30	1E	036	RS (record separator)	62	3E	076	>	>	94	5E	136	^	^	126	7E	176	~	~
31	1F	037	US (unit separator)	63	3F	077	?	?	95	5F	137	_	_	127	7F	177		DEL

Source: www.LookupTables.com

Last challenge - fruit.c

- ▶ Get an integer from the user via command line arguments. You may assume that it will be positive and will not be greater than `INT_MAX`.
- ▶ Get five pieces of fruit from the user and store them in an array.
- ▶ Use the integer we got via the command line to index into the array (assuming looping) and tell the user what they selected.
 - ▶ How can we use mod to make sure that the index we select is always inside the list?
 - ▶ Eg, if I wanted the 6th index of an array with 5 members, I would get the element at index 0

```
#include <cs50.h>
#include <stdio.h>
#define MAX 5
int main(int argc, string argv[])
{
    if (argc < 2)
    {
        return 1;
    }
    // get user input
    int user_in = atoi(argv[1]);

    // declare and array
    string fruits[MAX];

    // fill array with user input
    for (int i = 0; i < MAX; i++)
    {
        printf("Enter a fruit: ");
        fruits[i] = GetString();
    }
    // print the command line argument's index
    printf("You input was %d.\n", user_in);
    printf("Mod-ing by MAX (%d).\n", MAX);

    // get the final index
    int final_index = (user_in - 1) % MAX;

    printf("The new index is %d.\n", final_index);
    printf("Your fruit is %s\n", fruits[final_index]);
}
```