CS50 Section

Week 3

Attendance Sign In: https://tinyurl.com/y3xpwk9c

Agenda

- Pset 2 - General + Design Notes
- Week 2 - Remaining Questions
- Week 3
  - Structs
  - Sorting
  - Searching
  - Running Time
  - Recursion

Pset 2 Debrief

REMINDERS

1. Avoid magic numbers - Use **`#define`** directives, global constants, or a more expressive type instead

```
char c = (((plaintext[i] - 65) + key) % 26) + 65;
```

REMINDERS

1. Avoid magic numbers - Use **`#define`** directives, global constants, or a more expressive type instead

```
char c = (((plaintext[i] - 65) + key) % 26) + 65;
```

REMINDERS

1. Avoid magic numbers - Use **`#define`** directives, global constants, or a more expressive type instead

```
char c = (((plaintext[i] - 65) + key) % 26) + 65;
```

Refactor magic numbers to something another programmer reading your code can understand:

```
char c = (((plaintext[i] - 'A') +
key) % 26) + 'A';
```

REMINDERS

2. Know the difference between explicit and implicit casting.

```
#include <stdio.h>

int main(void) {
   int x = 7;
   printf("%i\n", x + (int) 'c');
}
```

REMINDERS

2. Know the difference between explicit and implicit casting.

```
#include <stdio.h>

int main(void) {
    int x = 7;
    printf("%i\n", x + (int) 'c');
}
```

This is an example of explicit casting. However, C does implicit casting for you when working with arithmetic operators:

```
#include <stdio.h>

int main(void) {
    int x = 7;
    printf("%i\n", x + 'c');
}
```

REMINDERS

3. Always perform error checking at the top of your code and explicitly check for the error, *not* for valid input.

```
#include <stdio.h>
int main(void) {
   if (argc == 2)
   {
       // Rest of my code goes here...
       return 0;
   }
   else {
       printf("Incorrect input.\n")
       return 1;
   }
}
```

REMINDERS

3. Always perform error checking at the top of your code and explicitly check for the error, *not* for valid input.

```
#include <stdio.h>
int main(void) {
   if (argc == 2)
   {
       // Rest of my code goes here...
       return 0;
   }
   else {
       printf("Incorrect input.\n")
       return 1;
   }
}
```

This forces you to indent all of the code in your `main` function unnecessarily and makes it unclear what actually triggers an error in your program.

REMINDERS

3. Always perform error checking at the top of your code and explicitly check for the error, *not* for valid input.

```
#include <stdio.h>
int main(void) {
   if (argc != 2)
   {
       printf("Incorrect input.\n")
       return 1;
   }

   // Rest of my code goes here...
   return 0;
}
```

This is preferred! Notice no `else` branch is needed if the input is valid.

REMINDERS

4. Note that `main` will return `0` automatically if you don't specify a non-zero return.

```
#include <stdio.h>

int main(void) {
   if (argc != 2)
   {
       printf("Incorrect input.\n")
   }

   // Rest of my code goes here...
   return 0;
}
```

REMINDERS

4. Note that `main` will return `0` automatically if you don't specify a non-zero return.

```
#include <stdio.h>

int main(void) {
   if (argc != 2)
   {
       printf("Incorrect input.\n")
   }

   // Rest of my code goes here...
}
```

This will return `0`. Also note that without a `return 1` statement, it will run the rest of the code until it hits the end of the main function, at which that point it will return `0`.

Week 2 Review - What Questions do You Have?

- Arrays
- Strings
- Command Line Arguments

Week 3

Today We'll Cover

- Structs
- Recursion
- Searching
- Sorting
- Running Time

Structs

Why use a Struct?

Why use a Struct?

```
typedef struct
{
    int month;
    int day;
    int year;
}
```

Why use a Struct?

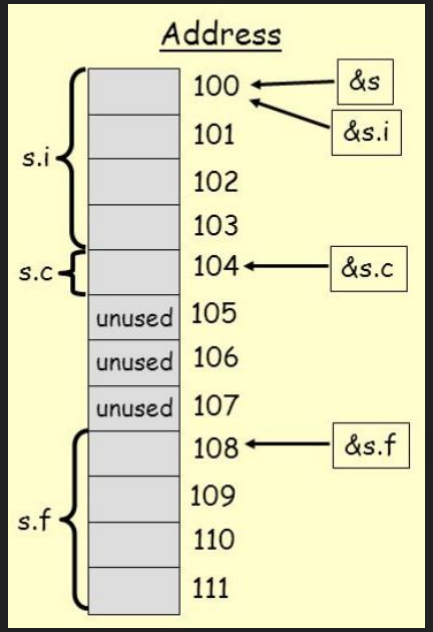
```
typedef struct
{
    int month;
    int day;
    int year;
}
```

```
typedef struct
{
    int id;
    string name;
    float gpa;
}
student;
```

How are Structs stored in Memory?

```
typedef struct{
  int i;
  char c;
  float f;
} Simple;

Simple s;
```



Searching

linear search

linear search

```
For i from 0 to n-1
    If i'th element is 50
        Return true
Return false
```

binary search

Recursive Example

Binary search

```
If middle item is 50
    Return true
```

Base Case

```
Else if 50 < middle item
    Search left half
Else if 50 > middle item
    Search right half
```

Iterative Case

```
If no items
    Return false
```

Pair Exercise #1: Fibonnaci

Write a recursive function fib that computes the nth Fibonacci number.
The 0th Fibonacci number is 0, the 1st Fibonacci number is 1, and every subsequent Fibonacci number is sum of the two preceding Fibonacci numbers:

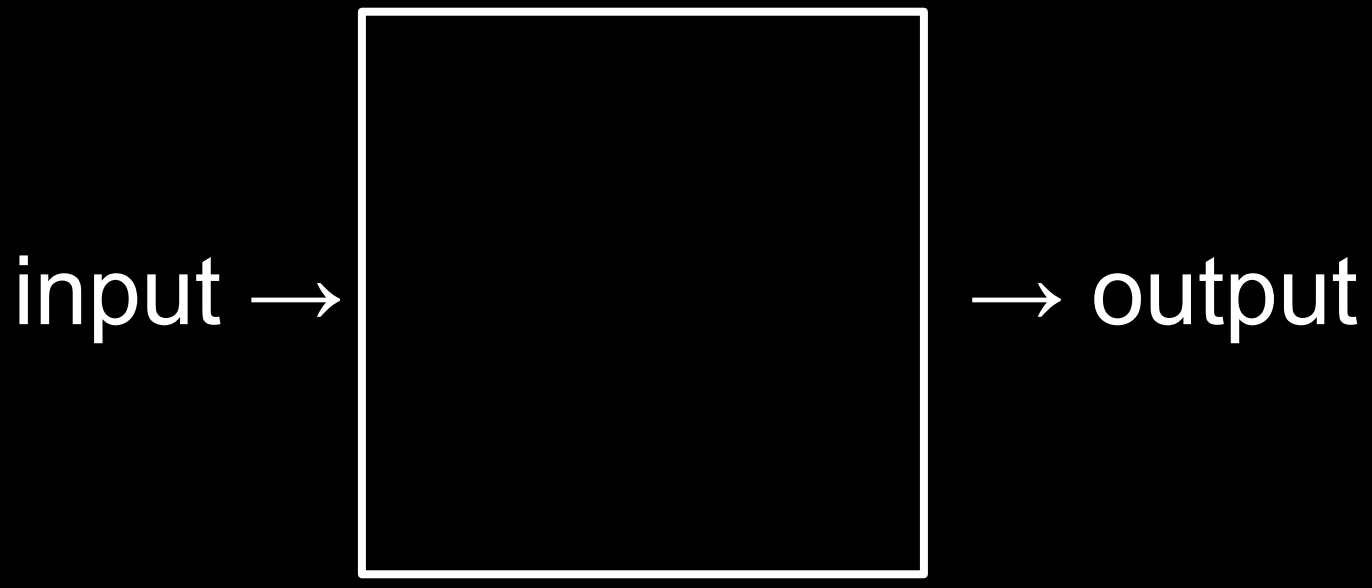
0, 1, 1, 2, 3, 5, 8, 13, 21...

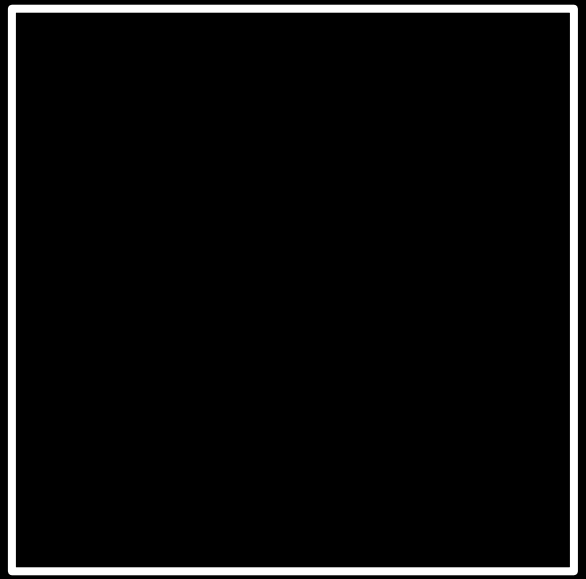
Starter Template: http://bit.ly/2mujKk7

Pair Exercise #1 - Solution

```
int fib(int n)
{
    if (n == 0)
        return 0;
    if (n == 1)
        return 1;
    return fib(n - 1) + fib(n - 2);
}
```

Sorting


input →
→ output

unsorted →  → sorted

7 2 1 6 3 4 50 → [ ] → 1 2 3 4 6 7 50


Insertion
Selection
Bubble
Shell
Merge
Heap
Quick
Quick3
Random
Nearly Sorted
Reversed
Few Unique

bubble sort

bubble sort

```
Repeat n-1 times
    For i from 0 to n-2
        If i'th and i+1'th elements out of order
            Swap them
```

bubble sort

```
Repeat n-1 times
    For i from 0 to n-2
        If i'th and i+1'th elements out of order
            Swap them
```

Try It! **Pair Exercise #2**

Starter Code: http://bit.ly/2mQR6Kh

Pair Exercise #2 - Solution

```
for (int i = 0; i < n - 1; i++)
{
    for (int j = 0; j < n - 1; j++)
    {
        if (values[j] > values[j + 1])
        {
            int temp = values[j];
            values[j] = values[j + 1];
            values[j + 1] = temp;
        }
    }
  }
```

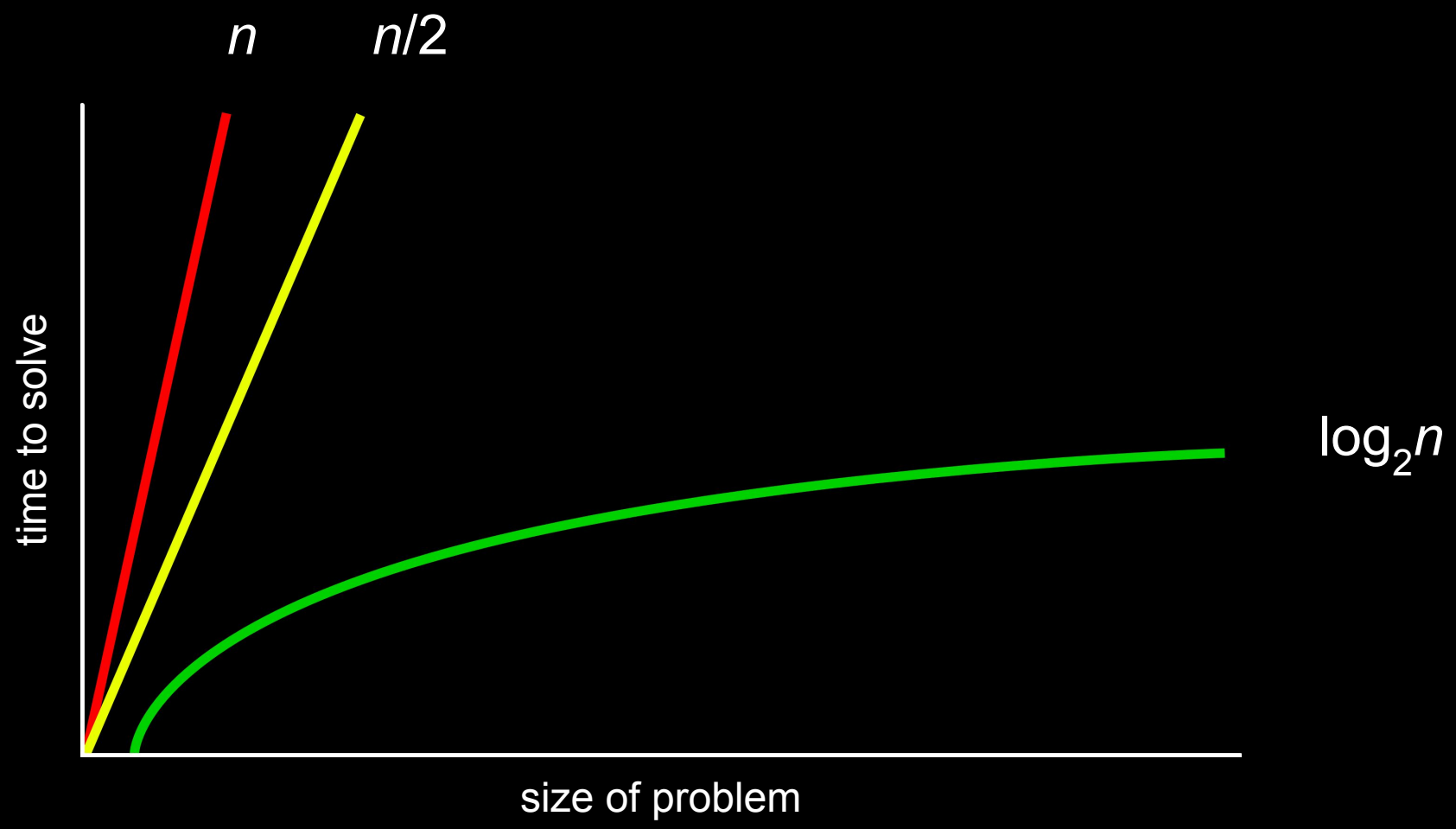
Pair Exercise #2 - Further Optimizations

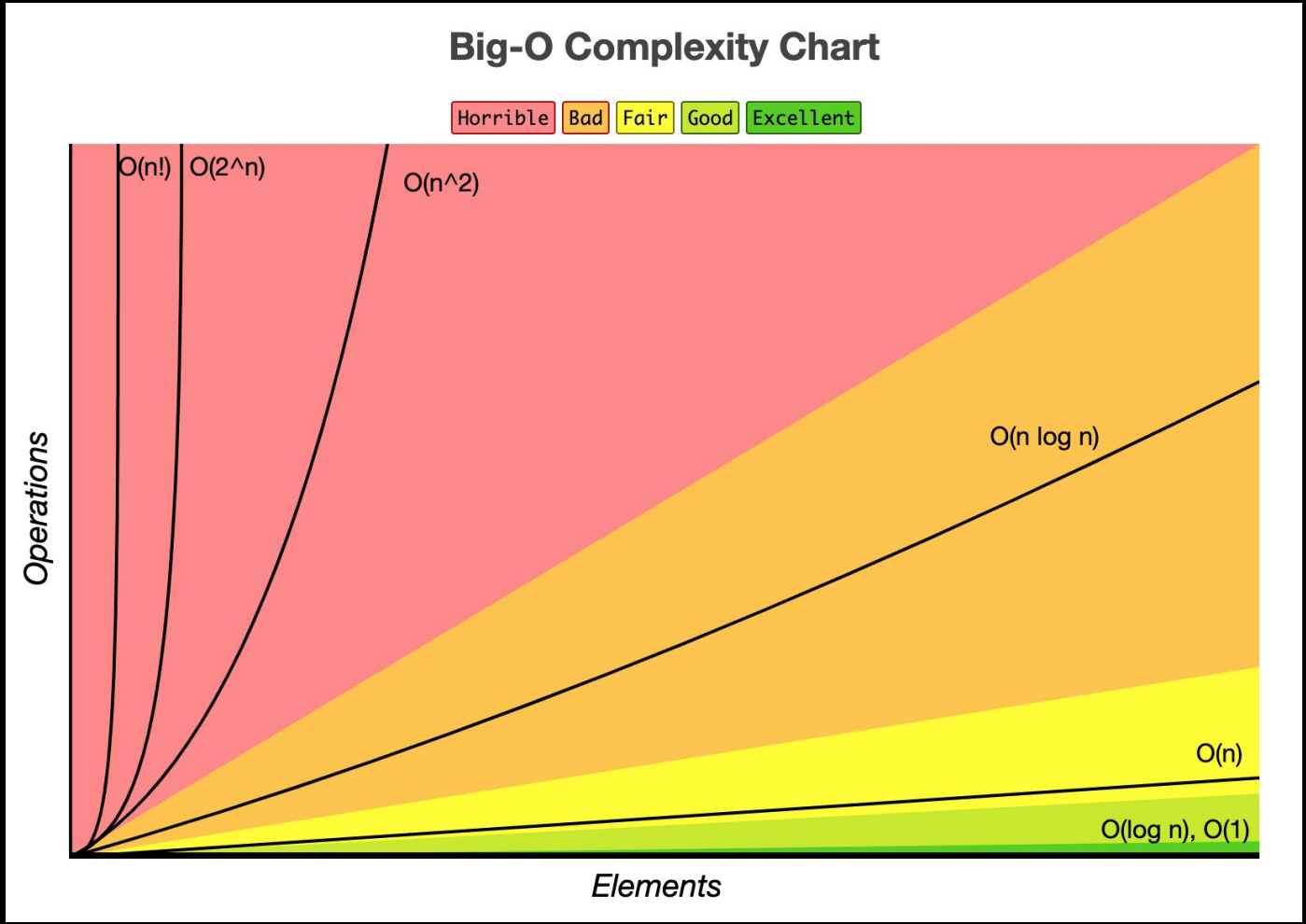
- **Optimization #1:** An optimization can be made by realizing that the inner loop need only loop `n - 1 - i` times instead of `n - 1` times, since each time the outer loop runs, the next highest number will be in its correct position.
- **Optimization #2:** Another optimization can be made by checking, on each iteration of the outer loop, if any swaps were made. If no swaps were made, then the list is sorted, and the loop can exit even if it hasn't run all `n - 1` times.

Pair Exercise #2 - Optimized Solution

```
for (int i = 0; i < n - 1; i++)
{
    bool swaps = false;
    for (int j = 0; j < n - 1 - i; j++)
    {
        if (values[j] > values[j + 1])
        {
            swaps = true;
            int temp = values[j];
            values[j] = values[j + 1];
            values[j + 1] = temp;
        }
    }
    if (swaps == false)
        break;
}
```

Runtime


n
n/2
log2n
time to solve
size of problem



CS50 Fall 2019. Adapted from Various Sources.

Array Sorting Algorithms

Algorithm	Time Complexity			Space Complexity
	Best	Average	Worst	Worst
Quicksort	Ω(n log(n))	Θ(n log(n))	O(n^2)	O(log(n))
Mergesort	Ω(n log(n))	Θ(n log(n))	O(n log(n))	O(n)
Timsort	Ω(n)	Θ(n log(n))	O(n log(n))	O(n)
Heapsort	Ω(n log(n))	Θ(n log(n))	O(n log(n))	O(1)
Bubble Sort	Ω(n)	Θ(n^2)	O(n^2)	O(1)
Insertion Sort	Ω(n)	Θ(n^2)	O(n^2)	O(1)
Selection Sort	Ω(n^2)	Θ(n^2)	O(n^2)	O(1)

CS50 Fall 2019. Adapted from Various Sources.


Insertion
Selection
Bubble
Shell
Merge
Heap
Quick
Quick3
Random
Nearly Sorted
Reversed
Few Unique

Calculating Runtime Complexity: BubbleSort

```
Repeat n-1 times
    For i from 0 to n-2
        If i'th and i+1'th elements out of order
            Swap them
```

Runtime?

```
Repeat n-1 times
    For i from 0 to n-2
        If i'th and i+1'th elements out of order
            Swap them
```

$(n - 1) \times (n - 2)$

$n^2 - 2n - 1n + 2$

$n^2 - 3n + 2$

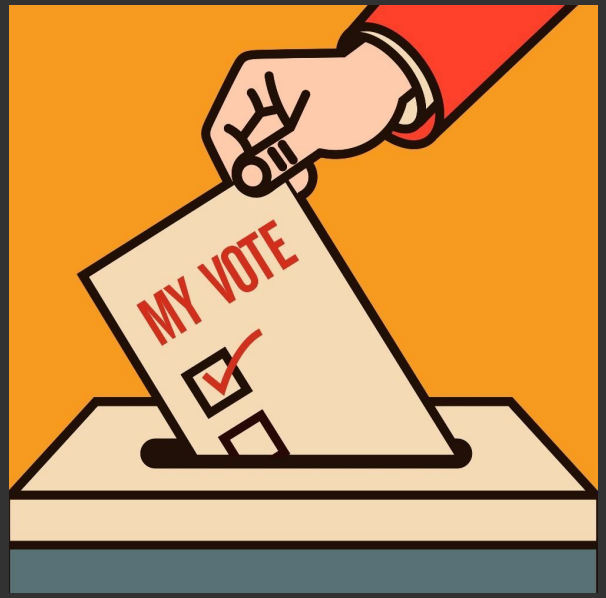
$O(n^2)$

PROBLEM SET 3 PREVIEW

Due Sun 9/29 @ 11:59pm

You will need to complete:

- **`Plurality`**
- `One of` **`Runoff`** `or` **`Tideman`**



Section Feedback: https://tinyurl.com/cs50rwfeedback

Appendix

Note on Strings

ASCII

Hex	Value	Hex	Value	Hex	Value	Hex	Value	Hex	Value	Hex	Value	Hex	Value	Hex	Value
00	NUL	10	DLE	20	SP	30	0	40	@	50	P	60	`	70	p
01	SOH	11	DC1	21	!	31	1	41	A	51	Q	61	a	71	q
02	STX	12	DC2	22	"	32	2	42	B	52	R	62	b	72	r
03	ETX	13	DC3	23	#	33	3	43	C	53	S	63	c	73	s
04	EOT	14	DC4	24	$	34	4	44	D	54	T	64	d	74	t
05	ENQ	15	NAK	25	%	35	5	45	E	55	U	65	e	75	u
06	ACK	16	SYN	26	&	36	6	46	F	56	V	66	f	76	v
07	BEL	17	ETB	27	'	37	7	47	G	57	W	67	g	77	w
08	BS	18	CAN	28	(	38	8	48	H	58	X	68	h	78	x
09	HT	19	EM	29	)	39	9	49	I	59	Y	69	i	79	y
0A	LF	1A	SUB	2A	*	3A	:	4A	J	5A	Z	6A	j	7A	z
0B	VT	1B	ESC	2B	+	3B	;	4B	K	5B	[	6B	k	7B	{
0C	FF	1C	FS	2C	,	3C	<	4C	L	5C	\	6C	l	7C	|
0D	CR	1D	GS	2D	-	3D	=	4D	M	5D	]	6D	m	7D	}
0E	SO	1E	RS	2E	.	3E	>	4E	N	5E	^	6E	n	7E	~
0F	SI	1F	US	2F	/	3F	?	4F	O	5F	_	6F	o	7F	DEL

Character Representation

Displayed	C	S	5	0	\0
In ASCII	43	53	35	30	0

Note on Command-Line Arguments

Command-Line Arguments

```
int main(int argc, string argv[])
{



}
```

COMMAND LINE ARGUMENTS

We can use command line arguments to pass arguments into our program:

```
int main(int argc, string argv[]) {


}
```

`argc` represents the number of arguments we've passed via the command line

`argv` is an array of strings with the different command line arguments

```
./caesar 2
```

argv[0] argv[1]

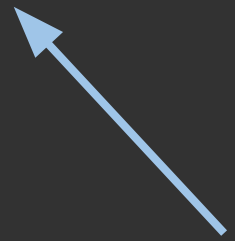
COMMAND LINE ARGUMENTS

```
$ ./ main "x" "y" "z"
```

COMMAND LINE ARGUMENTS

```
$ ./ main "x" "y" "z"
```

`argc` would be 4



argv[0]	argv[1]	argv[2]	argv[3]
“main”	“x”	“y”	“z”

COMMAND LINE ARGUMENTS

```
$ ./ main "x" "y" "z"
```

`argc` would be 4

`argv[0]` is always the name of the program

argv[0]	argv[1]	argv[2]	argv[3]
“main”	“x”	“y”	“z”

COMMAND LINE ARGUMENTS - SOME NOTES

- **`argv[]`** gives us an array of strings
  - If you want command line arguments that are processed as integers, use **`atoi(<string>)`**
  - Likewise, you can use **`atof(<string>)`** for doubles and various other functions

COMMAND LINE ARGUMENTS - SOME NOTES

Because they're strings, we can treat them as a multidimensional array. How?

```
$ ./ main "bob" "gloria" "suzy"
```

COMMAND LINE ARGUMENTS - SOME NOTES

Because they're strings, we can treat them as a multidimensional array. How?

```
$ ./ main "bob" "gloria" "suzy"
```

`argv[1][1]` **would give us "o".**