caesar

TODO

- ☐ get the key
- ☐ get the plaintext
- ☐ encipher
- ☐ print ciphertext

key = 2

plaintext	A	B	C	...	W	X	Y	Z
ciphertext	C	D	E	...	Y	Z	A	B

examples

```
$ ./caesar 2
ABCDEFGHIJKL
CDEFGHIJKLMN

$ ./caesar 2
This is CS50!
Vjku ku EU50!
```

TODO

- ☐ get the key
  - ▣ 2nd command line argument
  - ▣ `atoi`
- ☐ get the plaintext
- ☐ encipher
- ☐ print ciphertext

argc, argv

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

- **argc**
  - int
  - the number of arguments passed
- **argv**
  - array of strings
  - the list of arguments passed

argc, argv

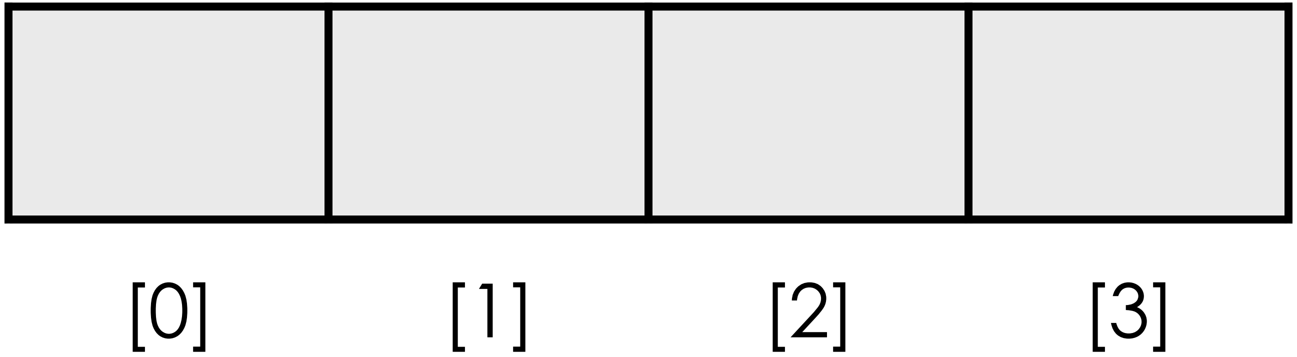
```
./blastoff Team Rocket
```

- `argc → 3`

./blastoff	Team	Rocket
argv[0]	argv[1]	argv[2]

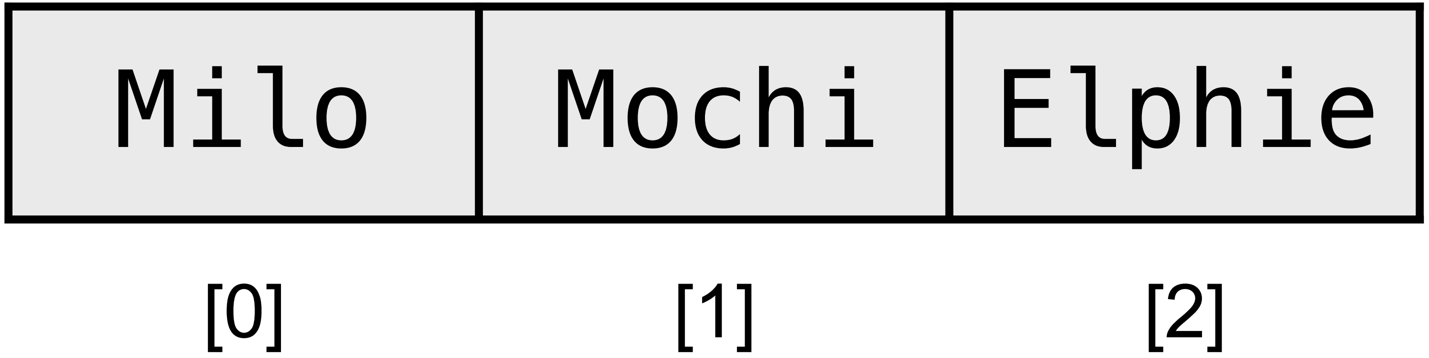
arrays

- data structures that hold multiple values of the same type
- entries are zero-indexed



creating an array

```
string dogs[3];
dogs[0] = "Milo";
dogs[1] = "Mochi";
dogs[2] = "Elphie";
```



correct usage

- **argc**: the number of arguments passed
  - argc must be 2

- **string argv[]**
  - argv[1] is a string
  - convert to int

`atoi`: from string to integer

```
string num = "50";
int i = atoi(num);
```

TODO

- ☑ get the key
- ☐ get the plaintext
  - ▣ `get_string`
- ☐ encipher
- ☐ print ciphertext

TODO

- ☑ get the key
- ☑ get the plaintext
- ☐ encipher
  - ▫ one character
  - ▫ entire plaintext
- ☐ print ciphertext

pseudocode

```
get key from command line argument
turn key into integer
prompt for plaintext
for each character in the plaintext string
    if alphabetic
        preserve case
        shift plaintext character by key
print ciphertext
```

shift letters only: `isalpha`
preserve capitalization: `isupper, islower`

`isalpha('Z')` → true
`isupper('Z')` → true
`islower('Z')` → false

```
char letter = 'Z';
if (isupper(letter))
{
    printf(letter);
}
```

key = 2

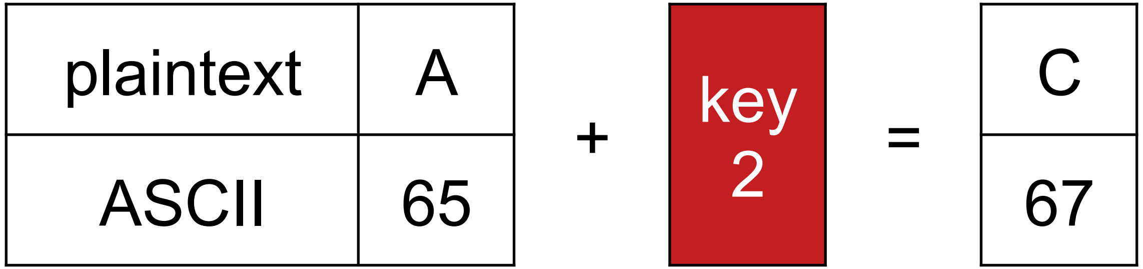
plaintext	A	B	C	...	W	X	Y	Z
ciphertext	C	D	E	...	Y	Z	A	B

ASCII Chart

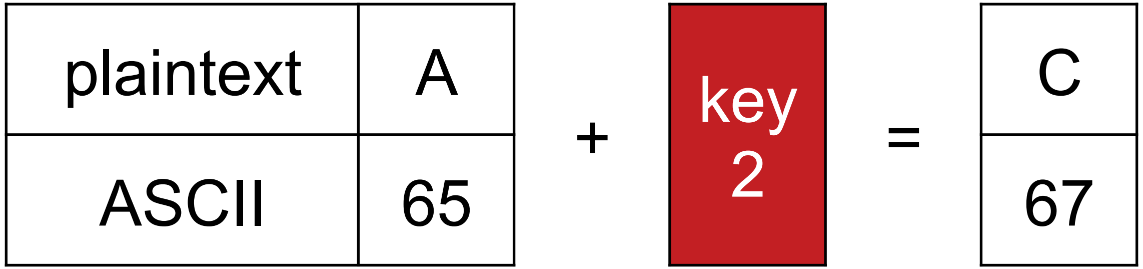
A	B	C	D	E	F	G	H	I	...
65	66	67	68	69	70	71	72	73	...

a	b	c	d	e	f	g	h	i	...
97	98	99	100	101	102	103	104	105	...

key = 2



key = 2



$$c_i = (p_i + k) \% 26$$

- c_i : ith ciphertext letter
- p_i : ith plaintext letter
- k : key
- % 26: remainder after dividing by 26

'A' + 2 = 'C' ?

ASCII Values

('A' + 2) % 26

= (65 + 2) % 26
= 67 % 26
= 15

C = 67

ASCII vs. alphabetical index

character	A	B	C	D	E	F	G	
ASCII	65	66	67	68	69	70	71	…
alphabetical	0	1	2	3	4	5	6	…

'A' + 2 = 'C' ?

ASCII Values	alphabetical Index
('A' + 2) % 26	A : 0
= (65 + 2) % 26	(0 + 2) % 26
= 67 % 26	= 2 % 26
= 15	= 2
C = 67	C = 2

'Y' + 2 = 'A' ?

ASCII Values	alphabetical Index
('Y' + 2) % 26	Y : 24
= (89 + 2) % 26	(24 + 2) % 26
= 91 % 26	= 26 % 26
= 13	= 0
A = 65	A = 0

alphabet wraparound

- start with: ASCII values
- encipher: alphabetical index
- print: ASCII values

ASCII → alphabetical?

alphabetical → ASCII?

ASCII vs. alphabetical index

character	A	B	C	D	E	F	G	
ASCII	65	66	67	68	69	70	71	…
alphabetical	0	1	2	3	4	5	6	…

TODO

- ☑ get the key
- ☑ get the plaintext
- ☐ encipher
  - ☑ one character
  - ▣ entire plaintext
- ☐ print ciphertext

strings

- a string is just an array of characters

```
string text = "This is CS50";
text[0] → 'T', text[1] → 'h'

strlen(text) → 12
```

pseudocode

```
get key from command line argument
turn key into integer
prompt for plaintext
for each character in the plaintext string
    preserve case
    shift plaintext character by key
print ciphertext
```

TODO

- ☑ get the key
- ☑ get the plaintext
- ☑ encipher
- ☑ print ciphertext

this was caesar