Open your IDE and create an *int main(void)* function. Then, create an *int increment(int n)* function outside of *int main(void)*, which returns *n + 1*. Call *increment* on 3, 5, and 7 in *int main(void)* and print your answers, each on a separate line.

Write your name on the paper. If you have free time, draw your favorite Pokémon ☺

We'll begin at ~~12:08pm!~~

3:08 pm

You can wait to start until 3 if you'd like.

CODE.CS50.IO

IDE

CS50 Section 1.5

Introduction

Fun Fact: This is one of like 2 photos of me in existence where I'm smiling!

Hi! I'm Luca – here are a couple of facts about me!

- Sophomore concentrating in Applied Math/Economics
- CIA (Chinese-Italian-American) – hence the last name
- Like books, tennis, sports, economics, politics, philosophy, and other stuff too

Tutorials + Contact Info

Every week, I'll host 3 hours of tutorials:

- Wednesday, 5-6 PM @ Pfoho Wolbach
- Saturday, 2-4 PM @ Pfoho Wolbach

I actually live in Lowell, but I've somehow been convinced by Phyllis (Head CA and creator of these slides) to somehow host my tutorials all the way in the wilderness of the Quad with her.

If you want to reach me outside of tutorials, feel free to email me at ldamicowong@college.harvard.edu! I'm happy to answer any questions you might have, and I will try my best to respond as promptly as possible!

806-543-3462

Section Overview

- Section will be held here at Northwest B105 every Tuesday from 3-5 PM.
- We'll go over content from lecture in more depth as well as complete a lab that will help prepare you for the problem set.
- Attendance is required and will be recorded, and please try and participate as best you can in section!

- C
- Compiling
- Strings
- Variables
- Types
- Loops
- Conditions
- Imprecision
- Overflow
- Extras

Machine Code

- CPUs understand **machine code**. These are the zeroes and ones that tell the machine what to do. Machine code might look like this: 01111111 01000101 01001100 01000110 00000010 00000001 00000001 00000000. Can you read this?

Machine Code

- CPUs understand **machine code**. These are the zeroes and ones that tell the machine what to do. Machine code might look like this: 01111111 01000101 01001100 01000110 00000010 00000001 00000001 00000000. Can you read this?

Assembly Code

- Assembly code includes more english-like syntax. Assembly code is an example of source code, syntax than can be translated to machine code.

- Some sequences of characters in assembly code include these: ***movl***, ***addq***, ***popq***, and ***callq***, which we might be able to assign meaning to. What do you think ***addq*** does?

Machine Code

- CPUs understand **machine code**. These are the zeroes and ones that tell the machine what to do. Machine code might look like this: 01111111 01000101 01001100 01000110 00000010 00000001 00000001 00000000. Can you read this?

Assembly Code

- Assembly code includes more english-like syntax. Assembly code is an example of source code, syntax than can be translated to machine code.

- Some sequences of characters in assembly code include these: ***movl***, ***addq***, ***popq***, and ***callq***, which we might be able to assign meaning to. For example, perhaps ***addq*** means to add or ***callq*** means to call a function. What values are we doing these operations on?

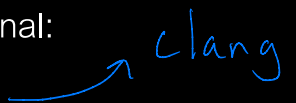
- The smallest unit of useful memory is called a **register**. These are the smallest units that we can do some operation on. These registers have names, and we can find them in assembly code as well, such as ***%ecx, %eax, %rsp,*** and ***%rsb***.

Compilers are pieces of software that know both how to understand source code and the patterns of zeroes and ones in machine code and can translate one language to another.

When we code in c, it sufficient to write this in the terminal:

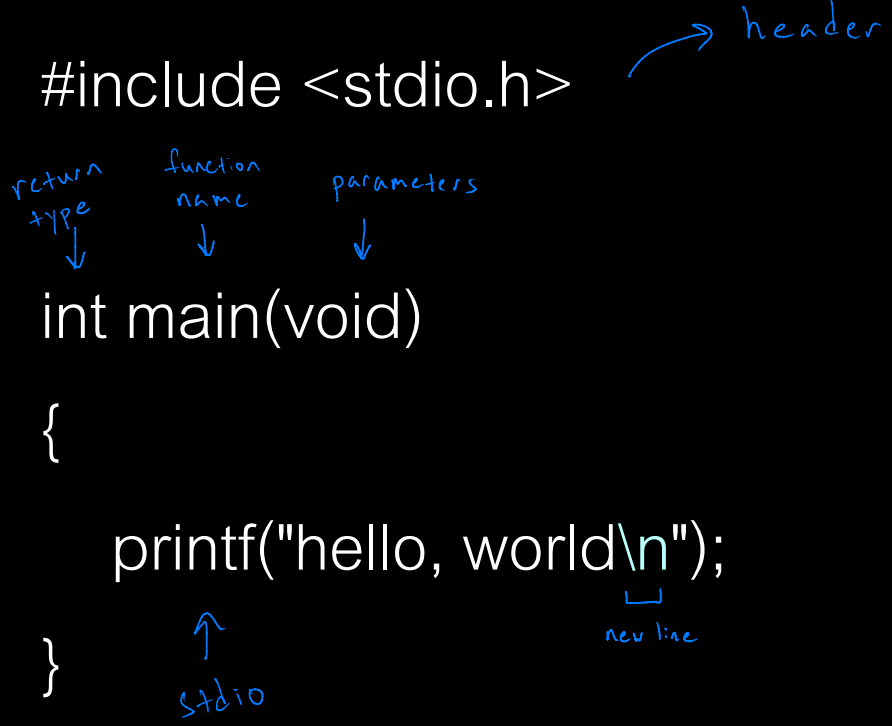
make file to generate the machine readable file

./file to execute the machine readable file



- C
- Compiling
- Strings
- Variables
- Types
- Loops
- Conditions
- Imprecision
- Overflow
- Extras

```c
#include <stdio.h>

int main(void)
{
    printf("hello, world\n");
}
```



```c
#include <stdio.h>

int main(void)
{
    printf("hello, world\n");
}
```

Escape Sequence!

Functions

Getting inputs (import cs50)	Printing (with printf)
get_char	%c
get_double	%f
get_float	%f
get_int	%i
get_long	%li
get_string	%s

'C'

3.0

3.0

3

1000000000

"hello"

Syntax

```
#include <cs50.h>
#include <stdio.h>

int main()
{
    int a = get_int("Input an integer: 3");
    printf("Value: %i \n", a);
}
```

int a = 3

Value : 3

Functions

Getting inputs (import cs50)	Printing (with printf)
get_char	%c
get_double	%f
get_float	%f
get_int	%i
get_long	%li
get_string	%s

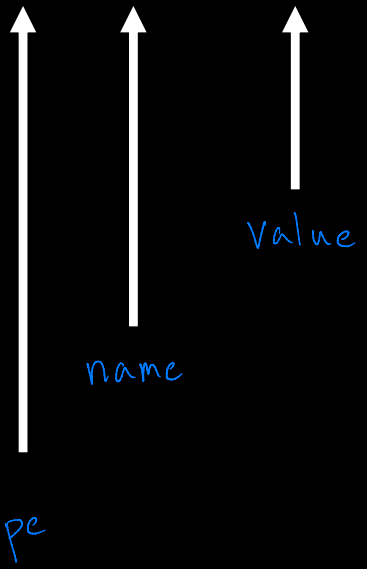
Syntax

```
#include <cs50.h>
#include <stdio.h>

int main()
{
        int a = get_int("Input an integer: ");
        printf("Value: %i \n", a);
}
```

%i is a placeholder for our variable a

Variables

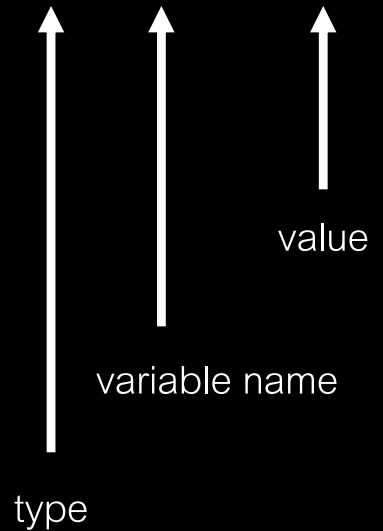
```
int x = 50;
```




```
char x = 'c'
double q = 4.32
```

Variables

```
int x = 50;
```



- C
- Compiling
- Strings
- Variables
- Types
- Loops
- Conditions
- Imprecision
- Overflow
- Extras

Conditions

```
if (x > 0)
{
        printf("x is positive\n");
}
else if (x < 0)
{
        printf("x is negative\n");
}
else if (x == 0)
{
        printf("x is 0\n");
}
```

What is the general structure of an if statement?

if (condition) {
    do stuff
}

equality

1 equals sign - assignment

Loops

```
for (int i = 0; i < 10; i++)
{
        printf("%i\n", i);
}
```

Output	i
0	0
1	1

```
0
1
2
:
9
```

What is the general structure of a loop?

```
for (int i=_; condition; updating i) {
    do stuff
}
```

3

Functions

```
int square(int x)
{
        return x * x;
}
```

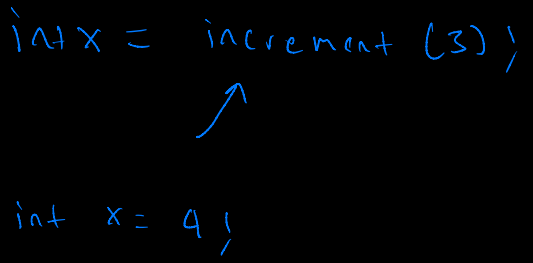
What is the general structure of a function?

return type name(parameters) {
    do stuff;
    return __
}

Return

Writing “return” halts the function! The next lines in the function don’t run!

Writing “return ___” will throw ___ back to the function’s original call.



Return

```
int main(void)
{
        print("%i\n", square(3));
}
int square(int x)
{
        return x * x;
}
```

f

9

return 3*3 = 9

Writing “return ___” will throw ___ back to the function’s original call.

Return

```
int main(void)
{
        print("%i\n", mystery(3));
}
int mystery(int x)
{
        return x * x;
        return x + x;
        return x – x;
}
```

Writing “return” halts the function! The next lines in the function don’t run!

Return

```
int main(void)
{
        int x = get_int(“hallo plz give me int that is not less than 0”);
        if (x < 0)
        {
                return;
        }
        play_game(x)
}
```


Writing “return” halts the function! The next lines in the function don’t run!

- C
- Compiling
- Strings
- Variables
- Types
- Loops
- Conditions
- Imprecision
- Overflow
- Extras

Terminal Functions:

ls: listing of all folders in the directory

rm file.txt: remove a file named “file.txt”

cd dir_name: change directories into dir_name

cd .. : changes directories into the parent directory

cd : changes directories into the root directory

Terminal Functions:

If you try to make a file that is not in your current directory, it will say that the target for your filename is not found. Ensure that you cd into the correct folder.

increment.c

Debugging – Reading Error Messages

Runtime Errors:

- Are you dividing by zero somewhere?
- Are you indexing a negative index or an index > length for an array?
- Do you have an infinite loop? (is your program terminating?) Break – Ctrl + C
- Are you accessing a piece of memory that you have yet to allocate?

Debugging – Reading Error Messages

Runtime Errors:

- Are you dividing by zero somewhere?
- Are you indexing a negative index or an index > length for an array?
- Do you have an infinite loop? (is your program terminating?)
- Are you accessing a piece of memory that you have yet to allocate?

Printing as a form of debugging!

If you don't know where you're getting an error, you can insert random print statements in the code to determine where the program stops running.

help50

debug50

man.cs50.io

Problem

Open your IDE and create an *int main(void)* function. Then, create an *int increment(int n)* function outside of *int main(void)*, which returns *n + 1*. Call *increment* on 3, 5, and 7 in *int main(void)* and print your answers, each on a separate line.

```
#include <stdio.h>
#include <cs50.h>
int increment(int x);

int main(void) {
    printf("%i\n", increment(3));
}

int increment(int x) {
    return x+1;
}
```

```
3
3
```

- Compiling (again!)
- Memory
- Arrays
- Strings
- Command-line Arguments

Compiling

- Preprocessing – headers! What are headers again? #include <stdio.h>

Compiling

- Preprocessing – headers! These are lines such as *#include <cs50.h>*, which tells **clang** to look for this header file first since it contains functions that we want to use in our program!

```c
#include <cs50.h>
#include <stdio.h>

int main(void)
{
    string name = get_string("Name: ");
    printf("hello, %s\n", name);
}
```

Compiling

- Preprocessing – headers! These are lines such as *#include <cs50.h>*, which tells **clang** to look for this header file first since it contains functions that we want to use in our program!

```c
#include <cs50.h>
#include <stdio.h>

int main(void)
{
    string name = get_string("Name: ");
    printf("hello, %s\n", name);
}
```

```c
string get_string(string prompt);
int printf(const char *format, ...);

int main(void)
{
    string name = get_string("Name: ");
    printf("hello, %s\n", name);
}
```

Compiling

- Preprocessing – headers! These are lines such as *#include <cs50.h>*, which tells **clang** to look for this header file first since it contains functions that we want to use in our program!

```c
#include <cs50.h>
#include <stdio.h>

int main(void)
{
    string name = get_string("Name: ");
    printf("hello, %s\n", name);
}
```

```c
string get_string(string prompt);
int printf(const char *format, ...);

int main(void)
{
    string name = get_string("Name: ");
    printf("hello, %s\n", name);
}
```

- Compiling – converts source code to assembly code. What are some things we remember about assembly code?

Compiling

- Preprocessing – headers! These are lines such as *#include <cs50.h>*, which tells **clang** to look for this header file first since it contains functions that we want to use in our program!

```c
#include <cs50.h>
#include <stdio.h>

int main(void)
{
    string name = get_string("Name: ");
    printf("hello, %s\n", name);
}
```



```c
string get_string(string prompt);
int printf(const char *format, ...);

int main(void)
{
    string name = get_string("Name: ");
    printf("hello, %s\n", name);
}
```

- Compiling – converts source code to assembly code. What are some things we remember about assembly code?
- Assembling – converts assembly code to machine code (binary)

Compiling

- Preprocessing – headers! These are lines such as *#include <cs50.h>*, which tells **clang** to look for this header file first since it contains functions that we want to use in our program!

```c
#include <cs50.h>
#include <stdio.h>

int main(void)
{
    string name = get_string("Name: ");
    printf("hello, %s\n", name);
}
```

```c
string get_string(string prompt);
int printf(const char *format, ...);

int main(void)
{
    string name = get_string("Name: ");
    printf("hello, %s\n", name);
}
```

- Compiling – converts source code to assembly code. What are some things we remember about assembly code?
- Assembling – converts assembly code to machine code (binary)
- Linking – contents of the previously compiled libraries that we want to link (ex. *cs50.c)* are combined with the binary in our program. Thus, in the program above, *hello.c, cs50.c, and printf.c* are all compiled into *hello.*

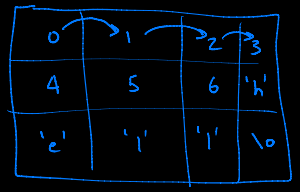
- Compiling (again!)
- Memory
- Arrays
- Strings
- Command-line Arguments

Strings array[3];

array[0] = "hi"

array[0][0] = 'h'

Memory and Arrays



When we store numbers in an array, each value is back to back, so we can use arithmetic to get an index. Then, we can instantly jump to that address. This gives us random access, which is constant time.

Suppose we want to store six values in memory. We then ask our operating system for just enough bytes for six numbers.

What if we wanted to add the number 50?

Memory and Arrays

When we store numbers in an array, each value is back to back, so we can use arithmetic to get an index. Then, we can instantly jump to that address. This gives us random access, which is constant time.

Suppose we want to store six values in memory. We then ask our operating system for just enough bytes for six numbers.

What if we wanted to add the number 50? Since we only asked our operating system for enough bytes for six numbers, the operating system might have already allocated the memory from 6 and beyond to some other aspect of our program.

For a temporary fix, we could've just asked the operating system for enough space for 7 or 8 or even 100 values. **In this case, we're asking for more memory than we actually need. Then, the computer has less space for other programs to store and run.**

int array[6];

Linked List

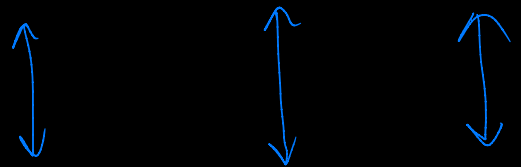
Single quote vs double quote?

```
'c'

char c = 'z'

String s = "hello"
```

Single quote vs double quote?

Chars to ints? Ints to chars?

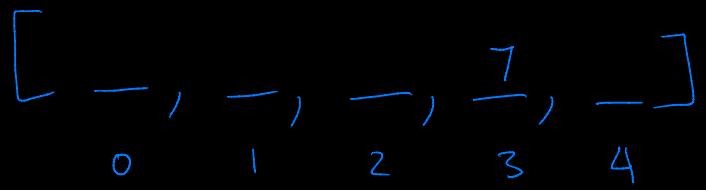
type name[size]



int scores[5]

NOT ZERO-INDEXED

scores[3] = 7;



```
int scores[3];

for (int i = 0; i < 3; i++)
{
    scores[i] = i;
}
```

scores[0] = 0;

scores[1] = 1;

scores[2] = 2;

```
int scores[3];

for (int i = 0; i < 3; i++)
{
    scores[i] = get_int("Score: ");
}
```

An array example

Create an array of size 5, where each element is two times the previous, and the first element is 1

```
int scores[5];
scores[0] = 1;
for (int i=1; i<5; i++) {
    scores[i] = 2 * scores[i-1];
3   scores[i] = 2 * scores[0];
```

1, 2, 4, 8, 16

'a' : 97 'c' : 99
count [0] count [2]

Recall…

```
int count[26];
string s = get_string("Give string: ");
length = strlen(s)
for (int i = 0; i < length; i++) {
    char = s[i];
    count[s[i] - 'a']++;
}
```

count[0] = # of As
count[1] = # of Bs

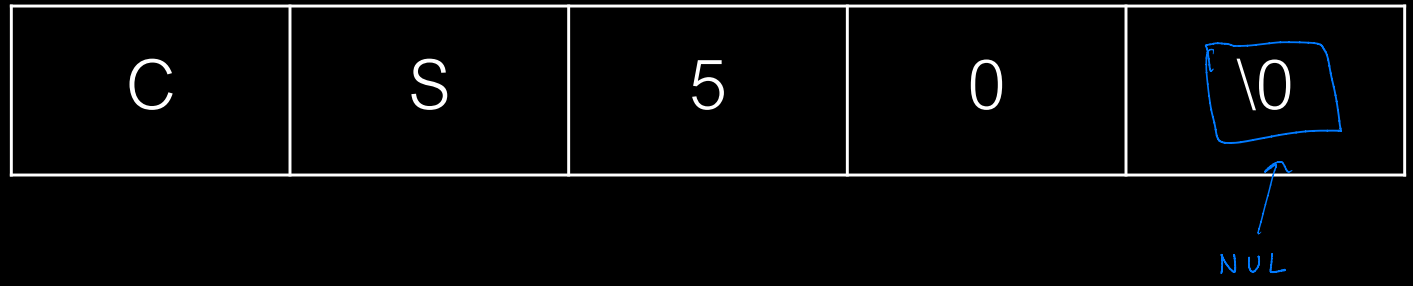
up to length - 1

“abcdefbcdea”

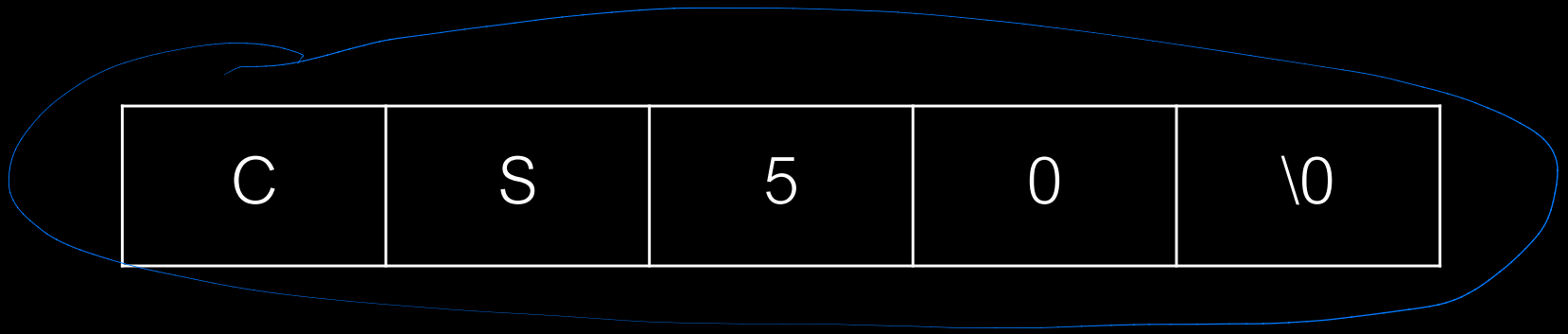
How can I store the quantity of each character in an array of length 26?

- Compiling (again!)
- Memory
- Arrays
- Strings
- Command-line Arguments

string s = "CS50";



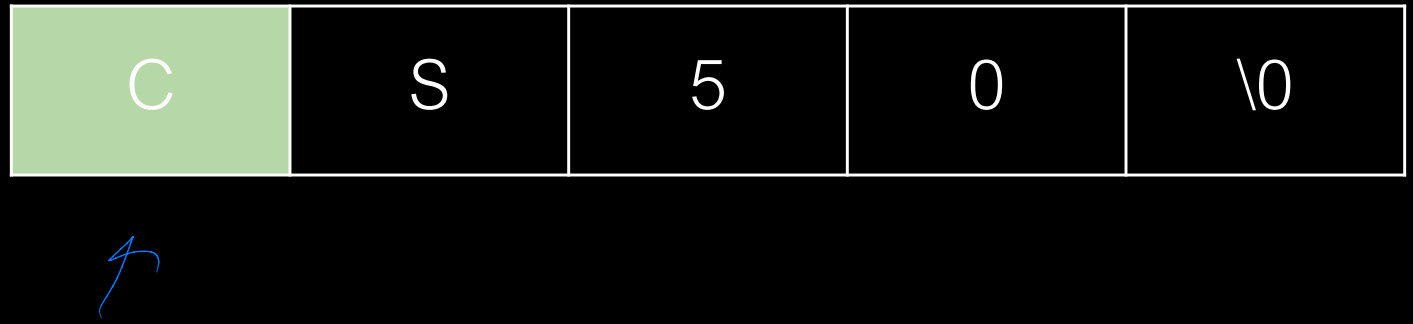
```
string s = "CS50";
```



What data structure is this?

ARRAY

s[0]



s[1]

A string example

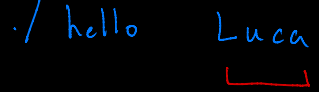
Print a string character by character, each on a new line

```
string s = "hello";
length = strlen(s);
    for (int i = 0; i < length; i++)
        {
          printf("%c\n", s[i]);
        }
```

- Compiling (again!)
- Memory
- Arrays
- Strings
- Command-line Arguments

Command-Line Arguments

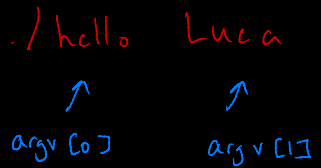
```
int main(int argc, string argv[])
{


}
```



Command-Line Arguments

argc: number of command line arguments

argv: array of command line arguments



./caesar 2

argv[0] argv[1]

```c
#include <cs50.h>
#include <stdio.h>

int main(int argc, string argv[])
{
    if (argc != 2)
    {
        printf("missing command-line argument\n");
        return 1;
    }
    printf("hello, %s\n", argv[1]);
    return 0;
}
```

NONZERO = BROKEN

correctness → Right

style → Commenting, Formatting, etc.

design → clever designs, runtime, etc.

tinyurl.com/cs50-style-design

Exercises

Create two files in your IDE: reverse.c and addition.c.

1. Write a program `reverse.c` that takes a string as input, and reverses it.

2. Write a program `addition.c` that adds two numbers provided as command-line arguments.

Exercises

Create two files in your IDE: reverse.c and addition.c.

1. Write a program `reverse.c` that takes a string as input, and reverses it.

- Sample Usage

```bash
$ ./reverse
Text: This is CS50.
Reverse: .05SC si sihT
```

Exercises

Create two files in your IDE: reverse.c and addition.c.

2. Write a program `addition.c` that adds two numbers provided as command-line arguments.

Details

- The program should accept two integers as command-line arguments.
- The program should output both original numbers, and their sum. If the program is run as `./additional 2 8`, for example, the output should be `2 + 8 = 10`.
- If the incorrect number of command-line arguments is provided, the program should display an error and return with exit code 1.

Lab

2 Words: Word 1 Word 2

Points Array: How much is each letter worth

isupper()

islower()

arr[n]

String word1 = "hello";

String word2 = "hi!";

CaT

String s

Compute_score() {

Check if letter is upper:
Find points somehow → 'C' - 'A' = 2

Check if letter is lower:
Find points somehow → 'c' - 'a' = 2

Add all points together

}

points[2] = points using c

Checks if uppercase

if (isupper(letter)) {
index = letter - 'A'
total = total + POINTS[index]
}

letter - 'A'

points[0] = points for a/A

```
int compute_score(string word)
{
    int length = strlen(word);
    int total = 0;
    for (int i = 0; i < length; i++)
    {
        if (isupper(word[i]))
        {
            total += POINTS[word[i] - 'A'];
        }
        if (islower(word[i]))
        {
            total += POINTS[word[i] - 'a'];
        }
    }
    return total;
}
```

sum += 1;
sum = sum + 1;] SAME

String word = word1;

```
if (score1 > score2)
{
    printf("Player 1 wins!\n");
}
if (score2 > score1)
{
    printf("Player 2 wins!\n");
}
if (score1 == score2)
{
    printf("Tie!\n");
}
```

Word = toupper(word)

tinyurl.com/luca-section

CS50 Section 1.5