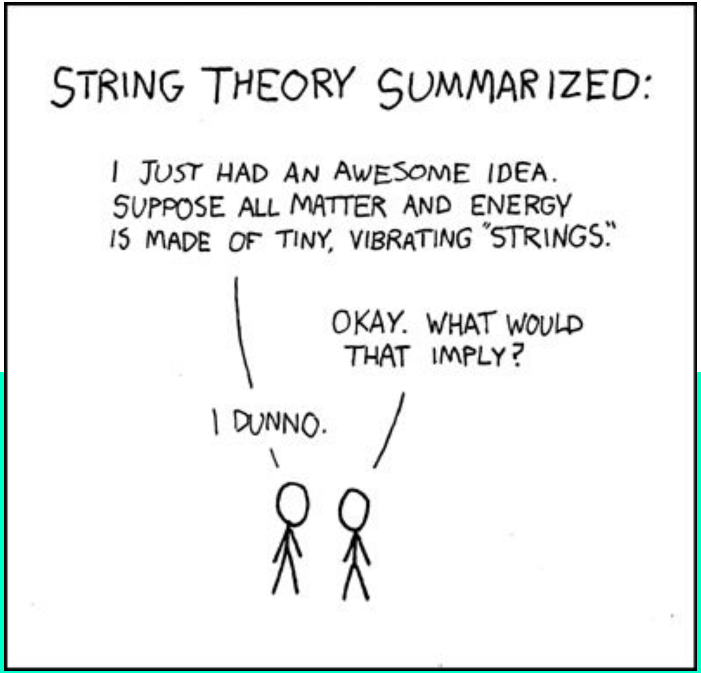
Lab 3 - Strings and I/O


STRING THEORY SUMMARIZED:
I JUST HAD AN AWESOME IDEA. SUPPOSE ALL MATTER AND ENERGY IS MADE OF TINY, VIBRATING "STRINGS."
OKAY. WHAT WOULD THAT IMPLY?
I DUNNO.

ReadMe

Reminders

- Lab 3 due Sunday, September 30th at 8pm on autograder
- Project 2 due Friday, October 5th at 8pm

Agenda

- Review of C-style strings, streams
- Compilation Sequence & Git

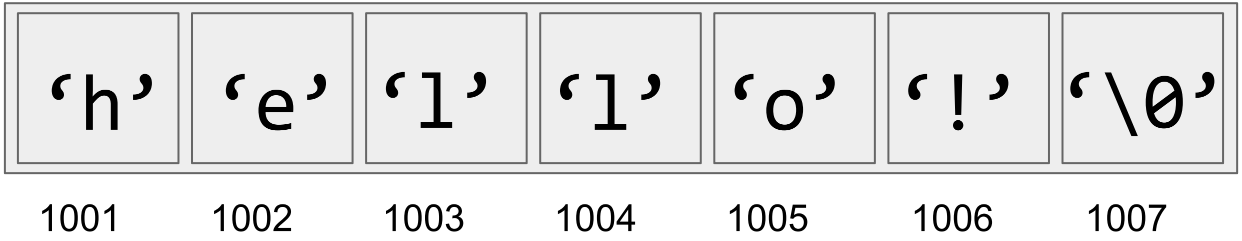
30

- Worksheet < 30
- Main part of the lab
- CHALLENGE: exam style question

remainder

C-Style Strings

- C-style strings are arrays of characters with sentinel value/ null character (‘\0’) at the end.
- The null character tells us when to stop traversing the array
- Different from C++ style strings (i.e. string str=“hi!”)
- We can declare them like this:
  - char greeting[7] = {‘h’, ‘e’, ‘l’, ‘l’, ‘o’, ‘!’, ‘\0’};
  - const char * farewell = “goodbye”;



‘h’	‘e’	‘l’	‘l’	‘o’	‘!’	‘\0’
1001	1002	1003	1004	1005	1006	1007

What will this code do?

```
char first[9] = {‘M’, ‘i’, ‘c’, ‘h’, ‘i’, ‘g’, ‘a’, ‘n’ ‘\0’};

char second[6] = {‘S’, ‘t’, ‘a’, ‘t’, ‘e’, ‘\0’};

first = second;



if(first == second) { cout << “they match!” << endl; }



char * third = second;
```

What will this code do?

```
char first[9] = {‘M’, ‘i’, ‘c’, ‘h’, ‘i’, ‘g’, ‘a’, ‘n’ ‘\0’};

char second[6] = {‘S’, ‘t’, ‘a’, ‘t’, ‘e’, ‘\0’};

first = second;
```

Doesn’t compile

```
if(first == second) { cout << “they match!” << endl; }
```

Compares pointers

```
char * third = second;
```

Doesn’t copy, just sets third to point at the same string as second

Two ways to declare:

- As an array:
  - char color[7] = “purple”;
  - We can change this array after we’ve declared it!

- As a pointer to a string literal:
  - const char * shape = “square”;
  - We cannot change this c-string after we’ve declared it!

Declaring arrays -- what works and what doesn’t?

What works

- char arr[] = “hello”;
- char arr[] =

{‘h’, ‘e’, ‘l’, ‘l’, ‘o’, ‘\0’}

- char arr[6] = “hello”;
- char arr[6] =

{‘h’, ‘e’, ‘l’, ‘l’, ‘o’, ‘\0’}

- const char * arr = “hello”;

What doesn’t

- char arr[5] = “hello”
- char arr[] =

{‘h’, ‘e’, ‘l’, ‘l’, ‘o’}

- char arr[5] =

{‘h’, ‘e’, ‘l’, ‘l’, ‘o’}

- char * arr = “hello”
- const char * arr =

{‘h’, ‘e’, ‘l’, ‘l’, ‘o’, ‘\0’}

Declaring arrays -- what works and what doesn't?

What works

- char arr[] = “hello”;
- char arr[] =

{‘h’, ‘e’, ‘l’, ‘l’, ‘o’, ‘\0’}

- char arr[6] = “hello”;
- char arr[6] =

{‘h’, ‘e’, ‘l’, ‘l’, ‘o’, ‘\0’}

- const char * arr = “hello”;

What doesn’t

- char arr[5] = “hello”

Needs size 6 for ‘\0’

- char arr[] =

Needs ‘\0’

{‘h’, ‘e’, ‘l’, ‘l’, ‘o’}

- char arr[5] =

Needs ‘\0’

{‘h’, ‘e’, ‘l’, ‘l’, ‘o’}

should be const

- char * arr = “hello”
- const char * arr =

can only declare with a * to a literal

{‘h’, ‘e’, ‘l’, ‘l’, ‘o’, ‘\0’}

Printing C-Strings

When we try to print arrays, it typically doesn’t work like we’d want. For example,

```
int array[2] = {42, 8675308};

cout << array << endl; // outputs a pointer to 42!
```

However, C treats char * / char [] in a special way…

```
char wow[6] = {‘p’, ‘u’, ‘p’, ‘p’, ‘y’, ‘\0’}

cout << wow << endl;
```

// outputs “puppy” because cout will keep printing until a null character is reached

Useful Tips:

Convert a c-string to a C++ string:

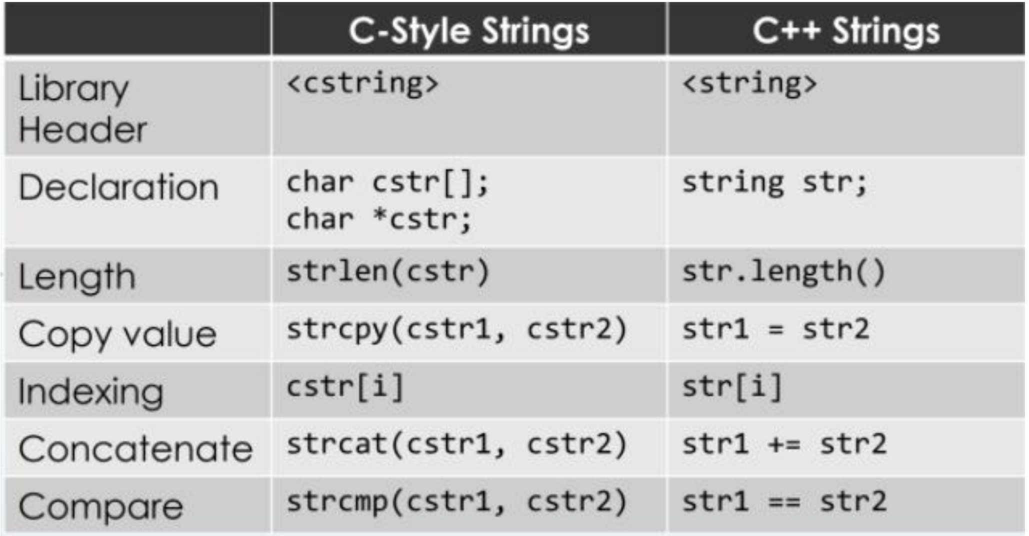
```
const char * cstr = “hello”;

string cpp_str = string(cstr);
```

Convert a C++ string into a c-string:

```
string cpp_str = “hello”;

const char * cstr =
cpp_str.c_str();
```



	C-Style Strings	C++ Strings
Library Header	`<cstring>`	`<string>`

Declaration	`char cstr[];`
	`char *cstr;`
		`string str;`

Length	`strlen(cstr)`	`str.length()`
Copy value	`strcpy(cstr1, cstr2)`	`str1 = str2`
Indexing	`cstr[i]`	`str[i]`
Concatenate	`strcat(cstr1, cstr2)`	`str1 += str2`
Compare	`strcmp(cstr1, cstr2)`	`str1 == str2`

Command Line Arguments

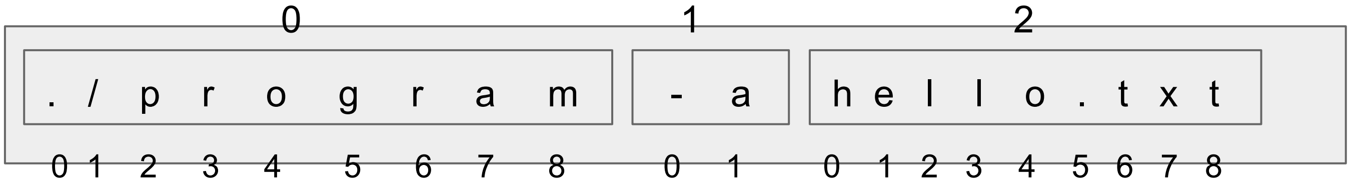
Just like we can pass arguments into any function, we can pass arguments into the main function of our programs:

```
int main(int argc, char * argv[])
```

OR

```
int main(int argc, char ** argv)
```

An array of arrays of characters -- for example, if we run: “   ./program -a hello.txt   “, we will have something like this:



0	1	2
. / p r o g r a m	- a	h e l l o . t x t
0 1 2 3 4 5 6 7 8	0 1	0 1 2 3 4 5 6 7 8

argc == 3

Recall : Using cin & cout

```
int main() {
    string password;
    cout << “Please enter your password: \n”;
    cin >> password;
    cout << “Processing your request…” << endl;
    int waitTime = 0;
    while(true) { ++waitTime; }
}
```

File I/O With Streams

- #include the library <fstream>
- This will allow you to read / write to files using these: “<<” and “>>” just like you would reading from cin / writing from cout.
- We can use “ifstream” (input file stream) and “ofstream” (output file stream) objects in order to read from files and write to files.

File Input With Streams

```
ifstream inputStream;

inputStream.open(“input.txt”);

if(!inputStream.is_open()) {
    cout << “error opening file!\n”;
    exit(1);
}

string nextWord;

while(inputStream >> nextWord) {
    cout << “The next word is “ << nextWord << endl;
}

inputStream.close();
```

Declare input file stream

Open file

Check for errors

Reads one word at a time

Close the file when you’re done!

File Output With Streams

Declare output file stream

```
ofstream outputStream;
```

Open file

```
outputStream.open(“output.txt”);
```

Check for errors

```
if(!outputStream.is_open()) {
    cout << “error opening file!\n”;
    exit(1);
}
```

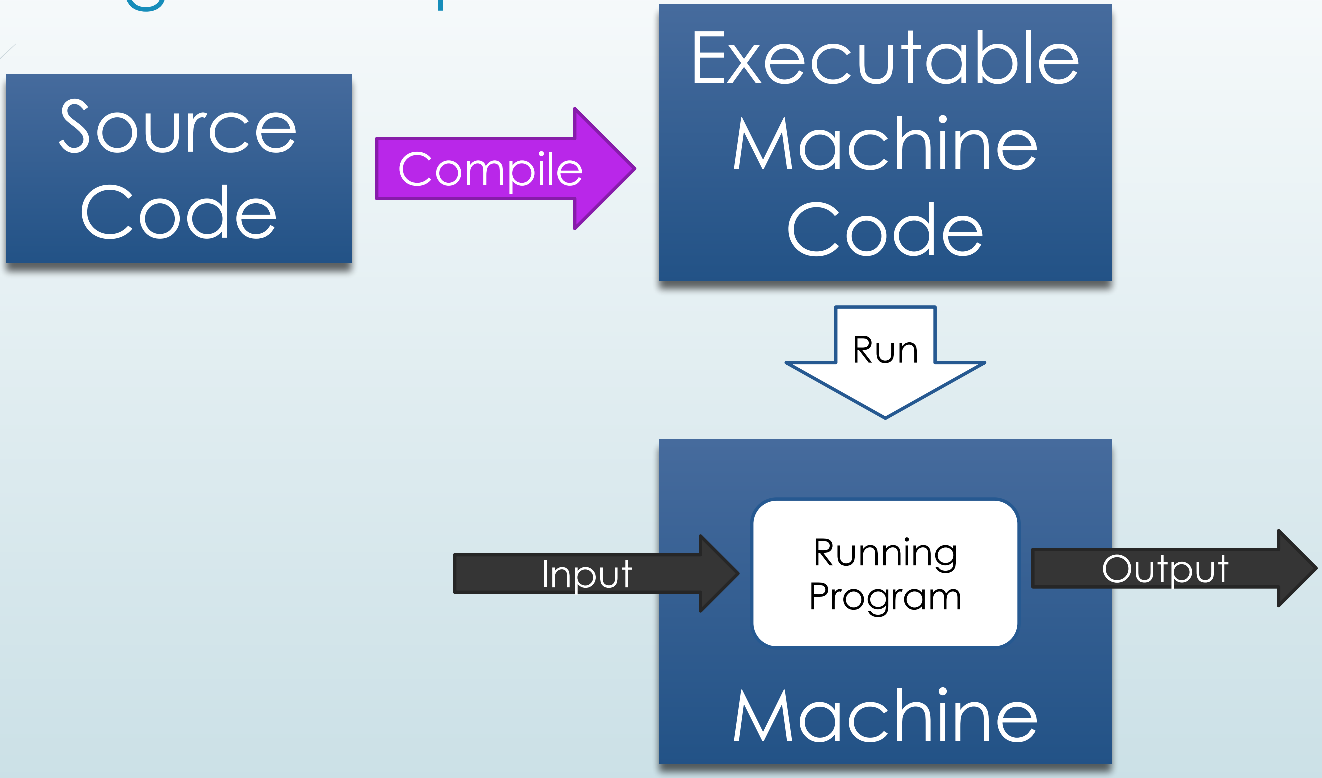
Counts to 200 & writes to output.txt

```
for(int i = 0; i < 200; ++i) {
    outputStream << i << endl;
}
```

Close the file when you’re done!

```
outputStream.close();
```

Using a Compiler



Source Code

Compile

Executable Machine Code

Run

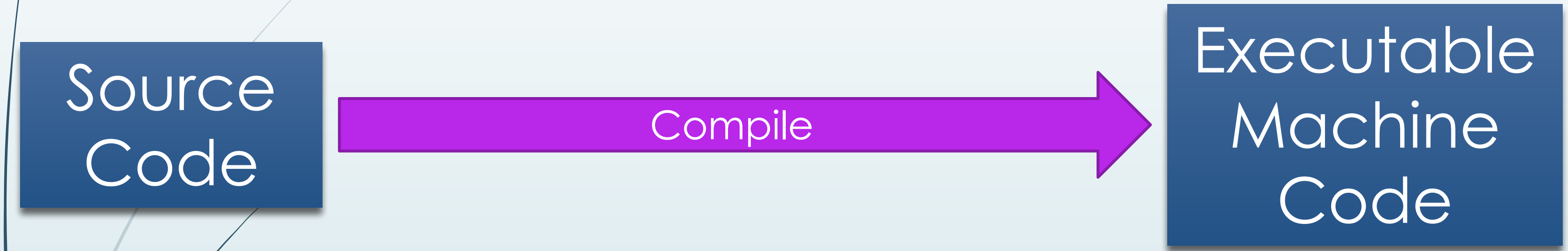
Input

Running Program

Output

Machine

The Compilation Sequence



Source Code

Compile

Executable Machine Code

1. Preprocessing
2. Compilation Proper
3. Assembly
4. Linking

Preprocessing

- Takes care of any preprocessor directives
  - e.g. #include, #define

```
g++ -E stats.cpp -o hello.ii
```

Compilation Proper

- Convert source into *assembly instructions*
  - This is the big one. It’s quite complicated.

- Many languages (including C++) have separate compilation
  - Each source file is compiled independently

```
g++ -S hello.ii -o hello.s
g++ -S lib.ii -o lib.s
```

Assembly

- Convert assembly instructions into a binary *object file*
- The code is not human-readable anymore!

```
g++ -c hello.s -o hello.o
g++ -c lib.s -o lib.o
```

Linking

- Source files have been compiled separately until this point.
- Linking essentially connects the *definition* or *implementation* of a function with places where it is used.

```
g++ hello.o lib.o -o hello.exe
```

Using g++

- If we use g++ without any special flags, by default the entire compilation process is performed.

```
g++ hello.cpp lib.cpp -o hello.exe
```

GIT

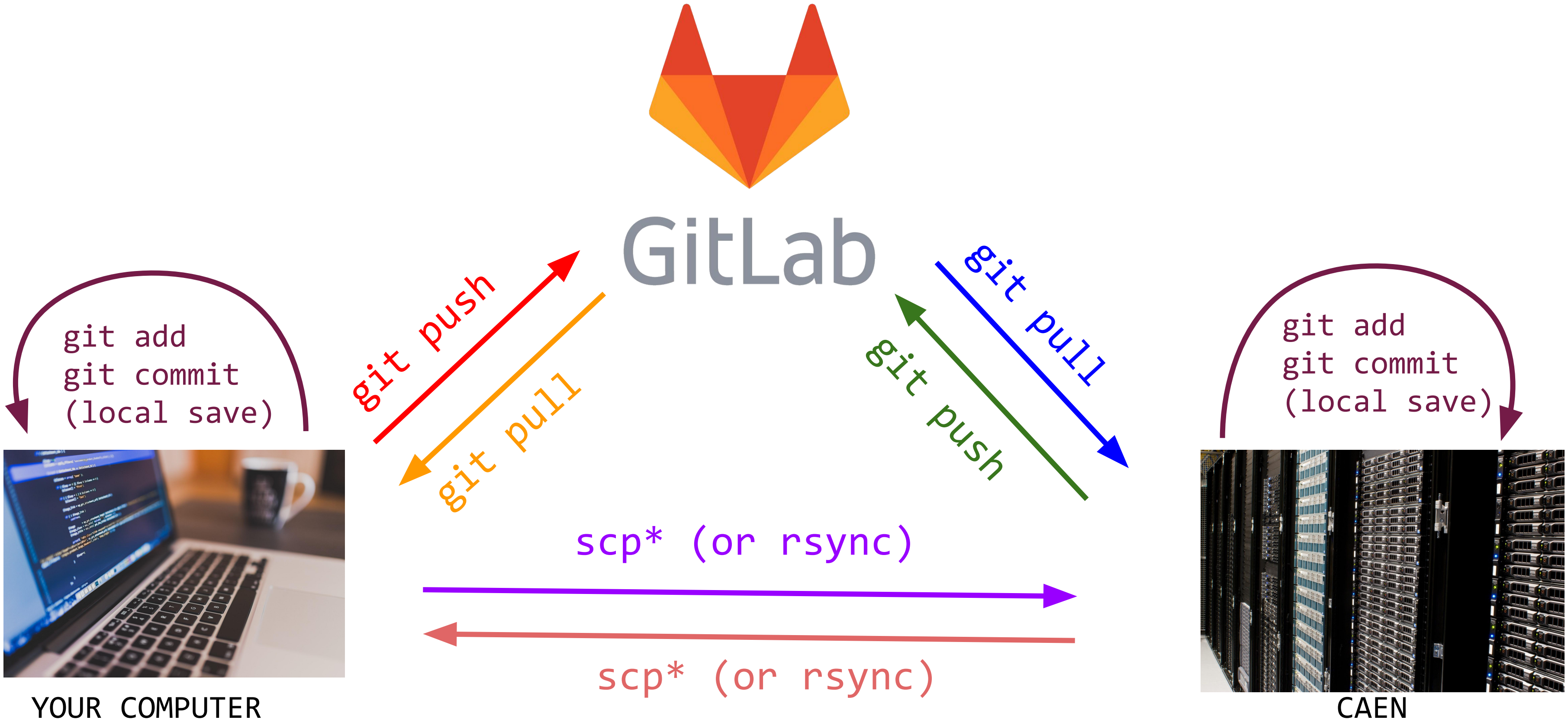
- Git is basically a version control program
- Helps us keep all of our files safe, keep every version of our project
    - We can return to an old version of our code if we break something
    - We can “push” to GitLab so that if something bad happens to our computer, our files are still safe
- Git ≠ GitLab, but GitLab ≅ GitHub
- When you use Git:
    - You save copies of previous versions in your directory so that you can return to them at any time
- When you use GitLab:
    - You also save copies of previous versions remotely- on GitLab’s servers- so if something goes wrong, you can always get your code back

Committing and pushing your code

- “Saves” in git are called “commits.”
- Remember these 3 basic steps:
  1. git add -A == “get ready to save all of the files in this directory”
  2. git commit -m “this is my message” == “save all of the added files, and associate this message with that save”
  3. git push -u origin master == “push my new saved code to GitLab so that the remote GitLab knows about the new changes too, not just my computer”

CAEN, GitLab, and You

How to get your files from one place to another



GitLab

git add
git commit
(local save)

git push

git pull

git pull

git push

git add
git commit
(local save)

scp* (or rsync)

scp* (or rsync)

YOUR COMPUTER

CAEN

Do the worksheet together in class

Main Part of Lab + Exam Style Questions

Challenge -- Exam Style Question

```
// Construct a function that, given a c-string, will find the
// instances of the character ‘elt’ within the c-string and duplicate
// them. The function should return the total number of times that
// ‘elt’ was duplicated.
// The size of the cstring should remain the same.

// Ex.
// char * searchMe = {‘p’,’o’,’t’,’a’,’t’,’o’,’\0’};
// duplicateMe(searchMe,’o’) -> returns 1, searchMe == pooato
// char * searchMe = {‘w’,’o’,’o’,’t’,’e’,’d’,’\0’};
// duplicateMe(searchMe,’o’) -> returns 1, searchMe == wooted
// char * searchMe = {‘a’,’l’,’a’,’b’,’a’,’m’,’a’,’\0’};
// duplicateMe(searchMe,’a’) -> returns 3, searchMe == aaaaaaa

int duplicateMe(char* str, char elt){

      // Your code here

}
```