Discussion 1B Notes (Week 6)

TA: Zhou Ren

Acknowledgement: Brian Choi's material

C Strings

C++ has an older brother, whose name is C. In C, we did not have the data type `string`, and C had its own way of representing strings. C++ inherited this older way of representing strings from C, and these old strings are still widely being used. For this reason, we are teaching you this old scheme, and we will call these strings "C strings."

Recall that a <u>string</u> is a <u>sequence of zero or more characters</u>. Also recall that an <u>array is a sequence of variables</u> of the type you define. Combing the two ideas, we see that we can use an <u>array of characters</u> to represent a string. Let us define a string as follows:

```
char s[10];
```

This creates space for a string of length 9. Wait, why 9, not 10?

Every C string must end with a marker that says "this is the end of the string." Otherwise, it is nothing but a character array! And this end marker occupies a slot, thus `char s[10]` can store up to 9 characters and an end marker, totalling 10. This end marker is what we call the **zero byte** (sometimes called the **null character**), which we use an escape sequence `\0` to represent (its ASCII number is 0), and we say C strings are **null-terminated**. The length of the string is determined by the location of the zero byte.

One can initialize a string as he/she declares it:

```
char s[10] = "HOWAREYOU";
```

H	O	W	A	R	E	Y	O	U	\0

This will automatically fill in the characters for you and fill the last slot with the zero byte.

You can also choose not to specify the size:

```
char s[] = "HOWAREYOU";
```


in which case the size is computed and set automatically.

You can also do:

```
char s[10] = "hello";
```


But you probably don't want to do the following for an obvious reason.

```
char s[10] = "Hello, world!";
```

Problem: What do you think will happen if you do the following?
(Hint: What matters is the location of the first zero byte.)

```
char s[10] = “HOWAREYOU”;
s[3] = ‘\0’;
cout << s << endl;
```


If the character array is a valid C string, you can print the string by doing:

```
cout << s;
```

Note that if this were a regular character array (one without the zero byte to mark the end), then `cout`-ing like this won’t work.

Functions for C strings

`.substr()`, `.size()` do not work with C strings, just as they don’t work with other arrays. But there are special functions that are defined for C strings. To use those functions, you need the following directive:

```
#include <cstring>
```

Let `s` and `t` be C strings, and let `n` be some integer. Here are some functions you must be interested in:

function	description
`strlen(s)`	Returns the length of `s`, not counting `‘\0’`.
`strcpy(t,s)`	Copies the string `s` to `t`. (**Notes**: `t=s` won’t do the job. Also, `strcpy` doesn’t do the size check for you. You must make sure there’s enough space in `t` yourself.)
`strncpy(t,s,n)`	Copies the first `n` characters of `s` to `t`. (**Note:** No size check.)
`strcat(t,s)`	Appends `s` to the end of `t`. (**Notes:** No size check. `t += s` won’t do the job.)
`strcmp(t,s)`	Compares `t` and `s`. Returns 0 if they are equal, something greater than 0 if `t > s`, and something less than 0 if `t < s`. (**Note**: `t == s` or `t < s` won’t do the job.)

Important Note

Once the char array is declared, values cannot be assigned using the shortcut, i.e.

```
char s[10];
s = “abcdefg”;
```

does not work. Direct assignment like that works only in an initialization expression. You should use `strcpy()` to assign a new string:

```
strcpy(s, “abcdefg”);
```

Problem: Write the function `isUppercase` that returns true if all characters in a C string `s` are uppercase letters, false otherwise.

```
bool isUppercase(char s[])
{









}
```

Did you use `strlen(s)` above? If so, can you do it without using it?

Creating an Array of C Strings

A C string is essentially an array of characters. This means an array of C strings is an array of arrays of characters. An array of arrays is simply a 2D array, meaning we can use:

```
char s[10][20];
```

to create an array of C strings. In `s`, we can store up to ___ C strings, and each C string can be at most ___ characters long. To make it an array of C strings, one must appropriately fill in zero bytes.

Then

```
s[3]
```

can be used to refer to the string in position 3, and

```
s[3][5]
```

can be used to refer to the letter in position 5 of the string in position 3.

Here's more concrete example:

```
char s[3][6];  // Can store three 5-letter words.
strcpy(s[0], “hello”);
strcpy(s[1], “apple”);
strcpy(s[2], “world”);

cout << s[0] << endl;   // prints “hello”
cout << s[2][2] << endl; // prints “r”
```

Problem: Write a function `printUppercaseStrings` that takes in an array of C strings called `arr` and prints all strings in the array that consist only of uppercase letters. The number of strings is given as an integer argument called `n`. When the function returns, it returns the total number of strings that it has printed. Assume the size of each C string is never greater than 20, and consider using `isUppercase` you defined in the previous page.

```
______ printUppercaseStrings(____________________________________________)
{









}
```

Converting a C string into a C++ string, and Vice Versa

A C string can be easily converted into a C++ string as follows:

```
char cs[10] = "hello";
string cpps;
cpps = cs;
```

If you are declaring a C++ string and want to initialize its value with a C string, you can do one of the following:

```
string cpps = cs;
string cpps(cs);
```

One can also convert a C++ string to a C string as follows:

```
char cs[10];
string cpps = "hello";
strcpy(cs, cpps.c_str());
```

However, you <u>cannot</u> do:

```
char cs[10] = cpps.c_str();
```

or

```
char cs[10];
cs = cpps.c_str();
```