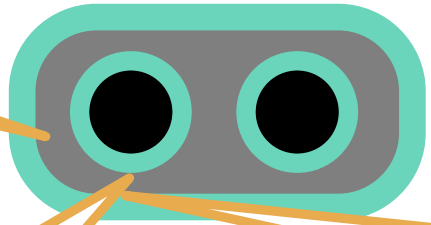
Strings

ESC101: Fundamentals of Computing

Nisheeth

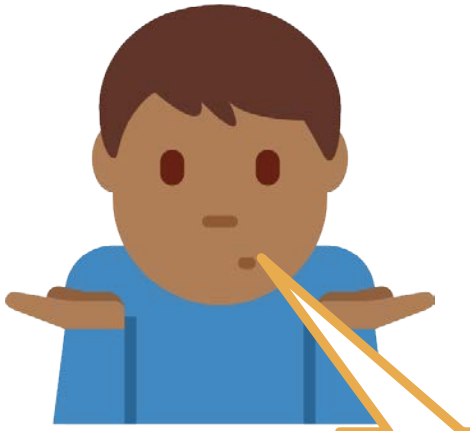
An important point about array initialization

- Is this a correct initialization for an integer array?



Yes ☺

```
int a[] = {2,-1,'A',2.15};
```



I told you that array stores values of the same data type

But it is okay to assign values of different data types. I will convert all of them (if convertible) into the same data type (that of array)

Why?

- The values being assigned to an array's elements are promoted/demoted based on the data type of the array elements
- The above will therefore be equivalent to

```
int a[] = {2,-1,65,2};
```

Character Arrays and Strings

- Character array: Each element is a character

```
char str[50] = {'H','e','l','l','o',' ','W','o','r','l','d'};
```

Note that not all 50 elements were initialized here (only first 11 were)

- String: A sequence of characters enclosed in double quotes " "
- A string can be declared and initialized as

```
char str[50] = "Hello World";
```

- Internally, a string is stored as a char array whose last element is '\0'

```
char str[50] = {'H','e','l','l','o',' ','W','o','r','l','d','\0'};
```

Equivalent to "Hello World"

The null character

The null character \0

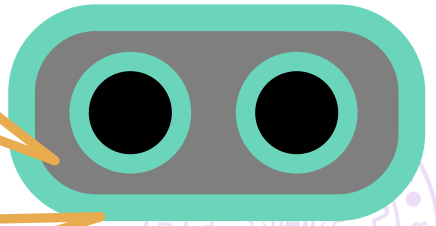
- Used to signal the end of a string (has ASCII code 0)
- Character arrays with a null character are treated as strings
- Mr. C will stop reading a character array after he sees \0

```
char str[50] = {'H','e','l','\0','l','o',' ','W','o','r','l','d'};
```

```
printf("%s",str);
```

Note: We use %s to print a string

Hmm … string is only till the \0. I will consider anything after that as garbage

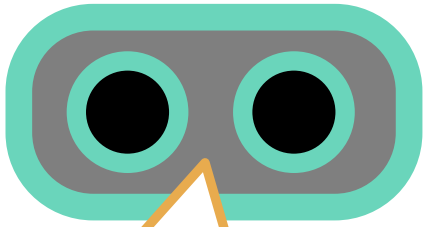


Hel

Different ways to declare/initialize a string

- Some valid ways to declare and initialize a string

```
char str[] = "Hello World";
char str[50] = "Hello World";
char str[12] = "Hello World";
char str[] = {'H','e','l','l','o',' ','W','o','r','l','d','\0'};
char str[50] = {'H','e','l','l','o',' ','W','o','r','l','d','\0'};
char str[12] = {'H','e','l','l','o',' ','W','o','r','l','d','\0'};
```

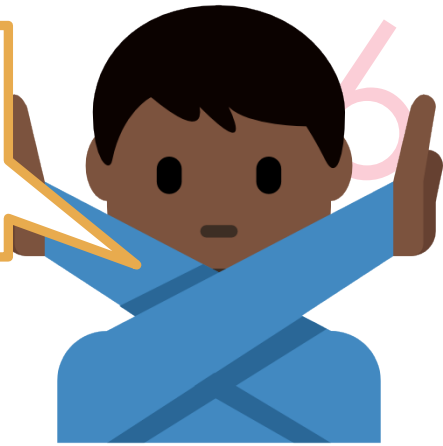


You need not specify the size of string. But if you specify the size, it should be at least one more than the length of the string

Note that Hello World has length 11, so size 12 is fine. Less than that may cause issues

Mr. C and the null character

Somewhat like saying
int num = {3,2,1};



When we say char str[6] = "Nice";

Mr C will store a \0 after last character ‘e’

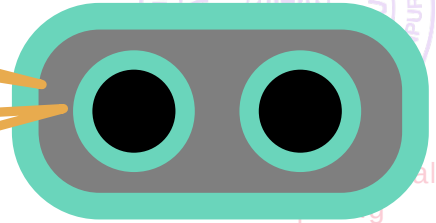
str	N	i	c	e	\0	

Warning: uninitialized character arrays contain junk

```
char str = "A";
putchar(str);
```

Strings are character arrays. “A” is a string. ‘A’ is a character

$



Mr. C and the nu ter

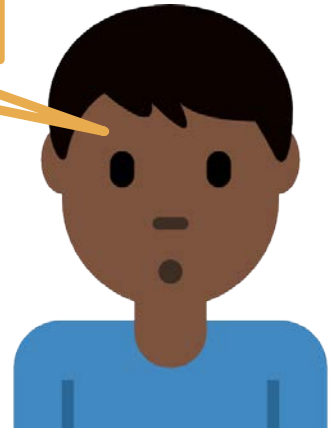
Will see it shortly

We did not write &str in scanf?

In fact when we read a string using gets or scanf, Mr. C yet again automatically puts a \0 at the end

```
char str[6] = "Nice";
```

The rest of the char array is still there



str	S	o	\0	e	\0	

```
scanf("%s",str);    printf("%s",str);
```

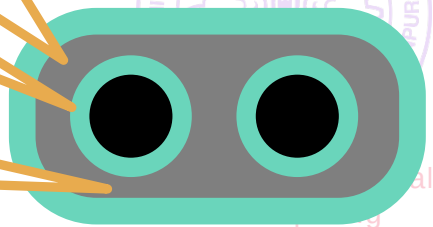
Will learn about this in a few weeks

Console | Activity Log | Input

So

Yes, I did not erase ‘e’ and ‘\0’ that were already there. I just overwrote the first two characters and then put a \0

No, since str is the whole array



So

Strings/char arrays are very useful

Will see some functions today

- Can use them to perform usual operations on text such as manipulation of words and sentences
- Very useful: Can also use strings to work with very big numbers

```
char bigNum[] =
"132339991122231395842506385218414025205258259436843253926503698250925809808250286028529520";
```

- In the big number above, what is the i-th digit (int) from left?

a char — bigNum[i-1] – '0'

- What is the i-th digit (int) from right?

a char — bigNum[len – i] – '0'

Len is the size of the string bigNum (can get it using strlen function

- Can use strings to write programs to do adding, multiplication, etc for very big numbers

Example: Adding two VERY BIG numbers

- Suppose we have two very big numbers
- Can represe

char bigNum 093899875874787574868";

char bigN: 4859865969363490980 7";

Suppose the sizes of the strings are len1 and len2, respectively (can get it using strlen function

Can store the result in another string/char array. Example:8+7 will give 15, ignoring carry 1, we have 5. To store 5 as a char, we can do '0' + 5 which will give the character '5'

Now ignore carry digit (if any) and add '0' to get the char version of result

Add these rightmost digits first

3432534665457360938998758747875748 68

43353 8646348598659693 09807

char '8'

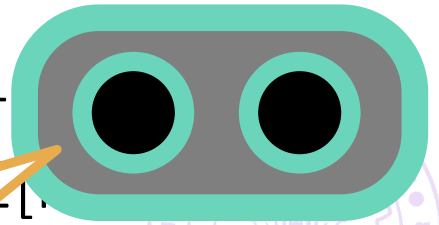
char '7'

sum_rightmost_digit = bigNum1[len1-1] – '0' + bigNum2[len2-

sum_second_digit_from_right = bigNum1

+ carry digit (if any) fr

Try writing the full program as a practice



Keep going right to left by repeating this procedure (and store result as a string)....

scanf with Strings

scanf("%s",str);

Will discuss the reason in detail when we study Pointers

Use %s to read string from input

No & needed since the whole char array is being read

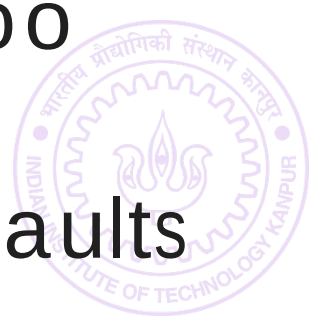
Mr C will automatically append a \0 at the end

Drawback: stops reading the moment any whitespace character is seen (\n, \t or space)

Very Risky: if user enters more characters than size of char array – segmentation fault!

Caution: Prutor **will give runtime error** if user enters too many more characters than space is available.

gcc and other industrial compilers will also give segfaults



scanf with Strings: An Example

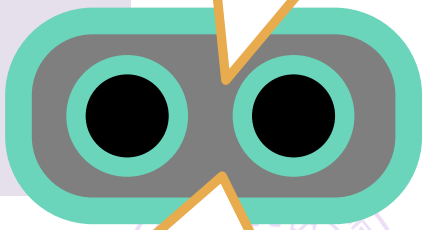
```
#include <stdio.h>

int main() {
 char str1[20], str2[20];

 scanf("%s",str1);
 scanf("%s",str2);

 printf("%s + %s\n", str1, str2);

 return 0;
}
```

INPUT
IIT Kanpur

OUTPUT
IIT + Kanpur

Read “I” as first string, stopped when saw white space and read “am” as second string, stopped again when saw the next space (“DON” ignored)

INPUT
I am DON



OUTPUT
I + am

Not scared of you DON. I won’t read you ☺

gets with Strings

No need for %s

```
gets(str);
```

Shortcut to read a single line of input
read all characters till \n – but **doesn't store \n, throws it away**

No & needed since the whole char array is being read

Mr C will au nd

Advantage: space or \t

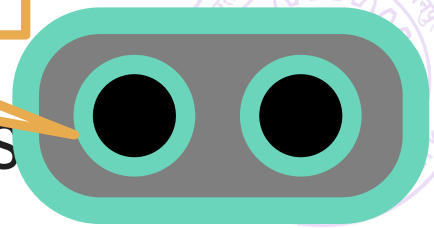
Very Risky: if user enters many more characters than space in char array

Caution: Prutor **will** rs too many more characters than space is

gcc and other industrial compilers will also give s

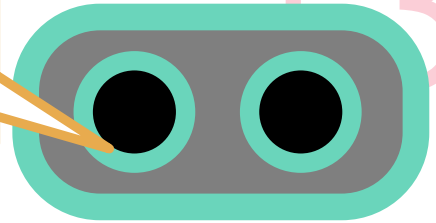
gets is *deprecated* in Clang
Do not use it regularly!

When some code becomes buggy or old or obsolete, it is declared as deprecated by the experts who developed that code



getline with Strings

Syntax? We will see it when discussing Pointers



A much **safer** version of gets

Reads a single line of input into the character array i.e. read all characters till \n – but **doesn't store \n, throws it away**

Mr C will automatically append a \0 at the end

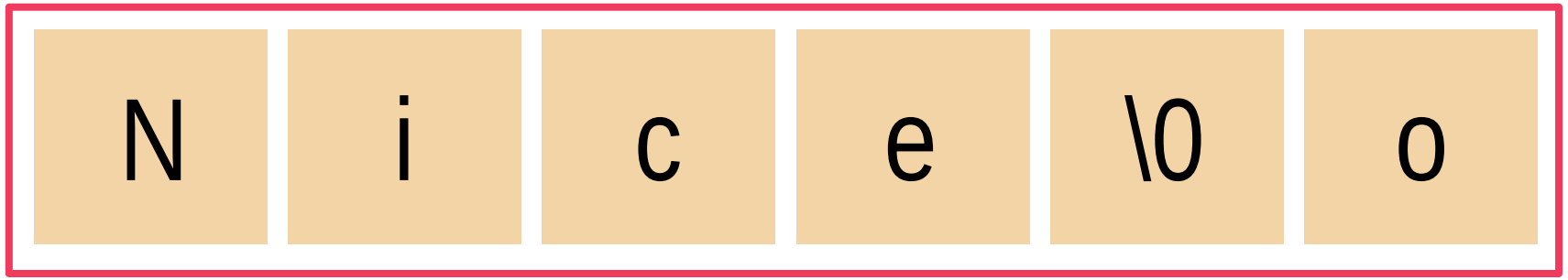
Advantage: If user enters more characters than length of char array, automatically enlarges the char array to be large enough to fit whatever user is entering

All compilers Clang, gcc etc do the above for getline

gets, scanf unsafe on gcc, but getline safe **everywhere**

String and Substring

String: Already saw that it is a character array ended with a NULL character



Substring: a contiguous subsequence of a string

E.g. "Nice", "Nic", "ice", "ce", "c", "Ni" are substrings of the above string

"Nce", "Nie", "ie", "Ne" **NOT substrings** (not contiguous) of above string

"No", "\0o", "\0", “abs", **NOT substrings** (contain chars not present in string)

Substrings need not contain the NULL character – WARNING!

Be careful when printing substrings – segmentation fault or weird behavior