

# String Theory with Mr C

ESC101: Fundamentals of Computing  
Purushottam Kar

# Make-up Lab

- Date: September 14<sup>th</sup>, 2018 (coming Friday)
- Time: 2PM – 5PM
- Room: CC-01 and CC-02
- Only Tuesday sections, B4, B5, B6, B13 need to attend
- No minor quiz (already done yesterday), only lab
- Prutor crashed yesterday and lab had to be cancelled
  - Detected hacking attempts from foreign servers ☺
  - Prutor defences strong – hacker could not even touch your data
  - Thanks to Prutor admins Umair, Prof. Karkare for making Prutor secure



# Mid-sem Theory Exam

- September 22<sup>nd</sup>, 2018 (Saturday)
- Time: 1PM – 3PM (afternoon)
- Rooms: assigned seating (like lab exam)
- Will be mailed to you – **sit at your own room/own seat**
  - If you do not then you will waste time moving to your proper seat
- Syllabus: till whatever is covered till Sep 14<sup>th</sup> tutorial
- No Make-up Exam – do not miss this exam
- Open handwritten notes – no printouts, mobiles, iPads.



# Doubt-clearing Session

- Date: September 15<sup>th</sup>, 2018 (coming Saturday)
- Time: 5PM – 7PM
- Room: CC-02
- Students not comfortable with English are welcome
- Other students also welcome to clear doubts
- Please revise and have list of doubts before coming
- Will not cover lectures again in detail – only doubts



# Advanced Track

- Advanced track students informed of their groups
  - Please contact your mentor and start discussing
  - Will float a few project ideas this evening
  - Free to choose your own creative idea too
- 
- Sorry about slides – last evening was very hectic
  - Prof. Karkare and Umair put in a lot of effort
  - Will put consolidated slides up tonight



# Character Arrays

6



ESC101: Fundamentals  
of Computing

# Character Arrays

6

All things we learnt about int/float arrays apply here too



# Character Arrays

6

All things we learnt about int/float arrays apply here too  
However, much more exciting things can be done here



# Character Arrays

6

All things we learnt about int/float arrays apply here too  
However, much more exciting things can be done here  
Char arrays also called *strings* (well ... almost all of them)



# Character Arrays

6

All things we learnt about int/float arrays apply here too

However, much more exciting things can be done here

Char arrays also called *strings* (well ... almost all of them)

English word *string* means a thread or a collection of items put together. The pearls were *strung* together.



# Character Arrays

6

All things we learnt about int/float arrays apply here too

However, much more exciting things can be done here

Char arrays also called *strings* (well ... almost all of them)

English word *string* means a thread or a collection of items put together. The pearls were strung together.

In C, string implies a character array (well ... almost)



# Character Arrays

6

All things we learnt about int/float arrays apply here too

However, much more exciting things can be done here

Char arrays also called *strings* (well ... almost all of them)

English word *string* means a thread or a collection of items put together. The pearls were strung together.

In C, string implies a character array (well ... almost)

Note: string is **not a datatype** in C



# Character Arrays

6

All things we learnt about int/float arrays apply here too

However, much more exciting things can be done here

Char arrays also called *strings* (well ... almost all of them)

English word *string* means a thread or a collection of items put together. The pearls were strung together.

In C, string implies a character array (well ... almost)

Note: string is **not a datatype** in C

Word string is **not a keyword** in C



# Character Arrays

All things we learnt about int/float arrays apply here too

However, much more exciting things can be done here

Char arrays also called *strings* (well ... almost all of them)

English word *string* means a thread or a collection of items put together. The pearls were strung together.

In C, string implies a character array (well ... almost)

Note: string is **not a datatype** in C

Word string is **not a keyword** in C

```
int string = 0;
```



# Character Arrays

All things we learnt about int/float arrays apply here too

However, much more exciting things can be done here

Char arrays also called *strings* (well ... almost all of them)

English word *string* means a thread or a collection of items put together. The pearls were strung together.

In C, string implies a character array (well ... almost)

Note: string is **not a datatype** in C

Word string is **not a keyword** in C

int string = 0;



# Declaring and Using Strings

16



ESC101: Fundamentals  
of Computing

# Declaring and Using Strings

16

Can be initialized at time of declaration



# Declaring and Using Strings

Can be initialized at time of declaration

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



# Declaring and Using Strings

Can be initialized at time of declaration

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

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

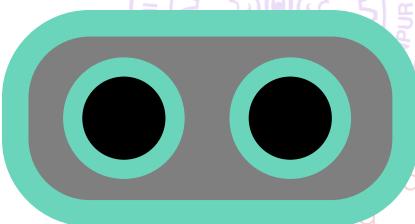


# Declaring and Using Strings

Can be initialized at time of declaration

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

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



# Declaring and Using Strings

Can be initialized at time of declaration

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

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

Partly initialized since only  
11 characters this phrase



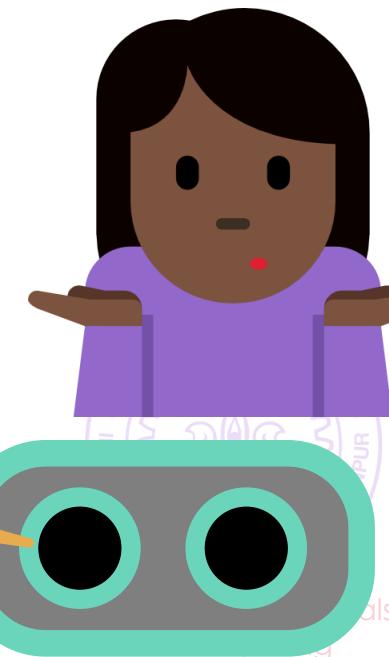
# Declaring and Using Strings

Can be initialized at time of declaration

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

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

Partly initialized since only  
11 characters this phrase



# Declaring and Using Strings

16

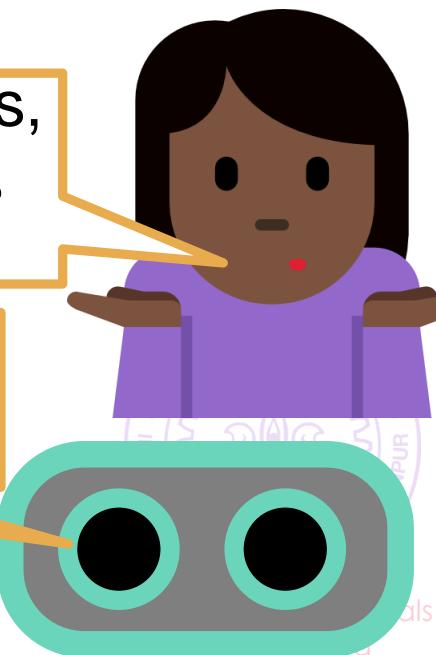
Can be initialized at time of declaration

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

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

Hello has 5 characters,  
World has 5, what is  
the 11<sup>th</sup> character?

Partly initialized since only  
11 characters this phrase

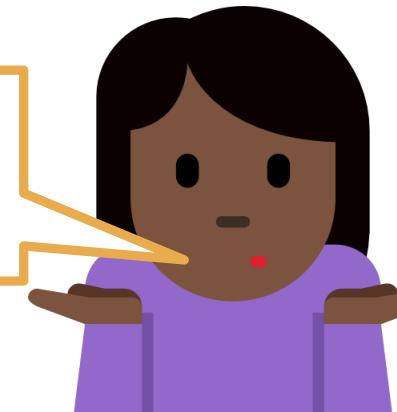


# Declaring and Using Strings

Can be initialized at time of declaration

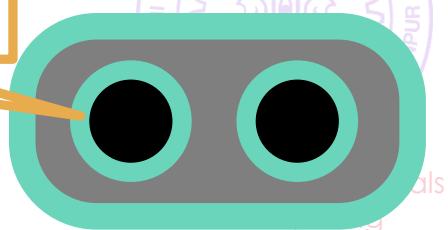
```
char str[50] = {'H','e','l','l','o',' ','W','o','r','l','d'};
```

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



Hello has 5 characters,  
World has 5, what is  
the 11<sup>th</sup> character?

Partly initialized since only  
11 characters this phrase



# Declaring and Using Strings

Can be initialized at time of declaration

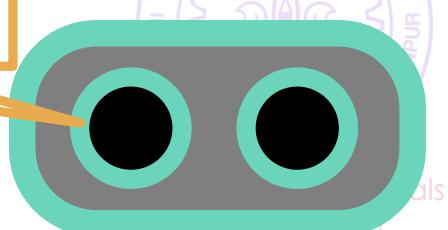
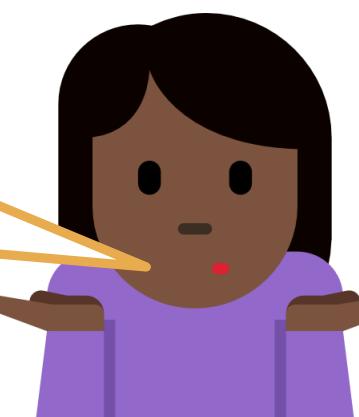
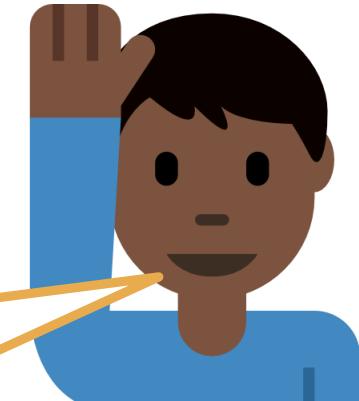
```
char str[50] = {'H','e','l','l','o',' ','W','o','r','l','d'};
```

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

The space between  
the two words. Space  
is a character too!

Hello has 5 characters,  
World has 5, what is  
the 11<sup>th</sup> character?

Partly initialized since only  
11 characters this phrase



# Declaring and Using Strings

Can be initialized at time of declaration

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

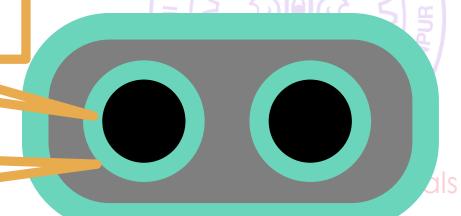
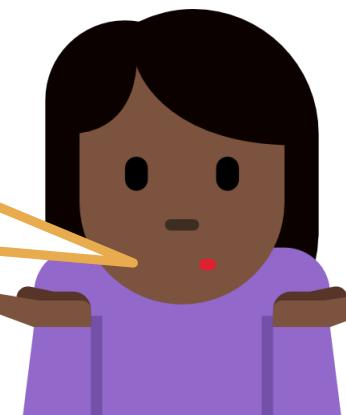
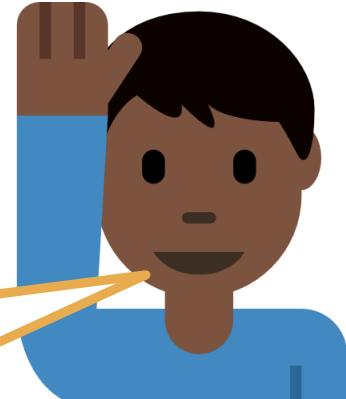
```
char str[50] = "Hello World";
```

The space between  
the two words. Space  
is a character too!

Hello has 5 characters,  
World has 5, what is  
the 11<sup>th</sup> character?

Partly initialized since only  
11 characters this phrase

Very good!



# Declaring and Using Strings

Can be initialized at time of declaration

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

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

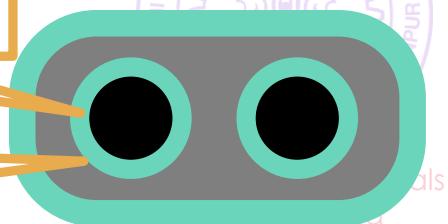
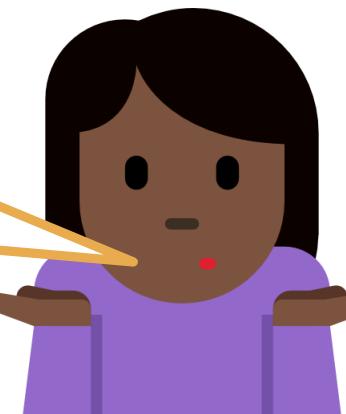
Just like int arrays, char arrays cannot be initialized this way after declaration is done

The space between the two words. Space is a character too!

Hello has 5 characters, World has 5, what is the 11<sup>th</sup> character?

Partly initialized since only 11 characters this phrase

Very good!



# Declaring and Using Strings

Can be initialized at time of declaration

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

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

Just like int arrays, char arrays cannot be initialized this way after declaration is done

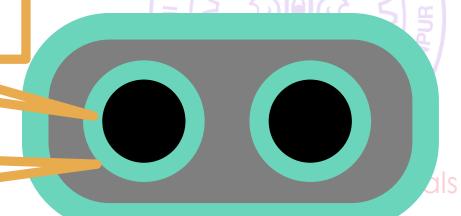
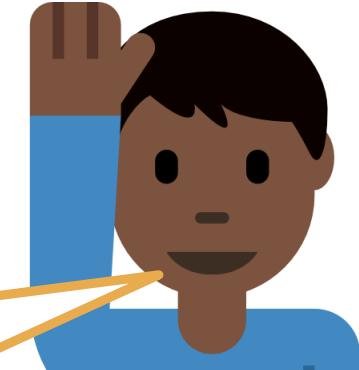
Other ways: scanf (with %s), gets

The space between the two words. Space is a character too!

Hello has 5 characters, World has 5, what is the 11<sup>th</sup> character?

Partly initialized since only 11 characters this phrase

Very good!



# Declaring and Using Strings

Can be initialized at time of declaration

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

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

Just like int arrays, char arrays cannot be initialized this way after declaration is done

Other ways: scanf (with %s), gets

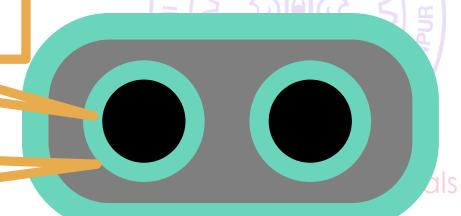
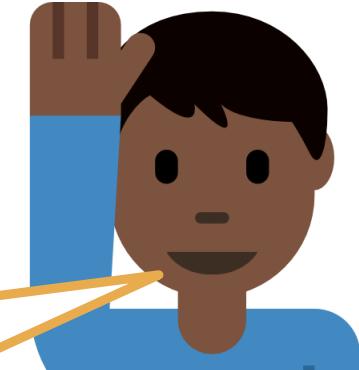
Both are very unsafe – crash!

The space between the two words. Space is a character too!

Hello has 5 characters, World has 5, what is the 11<sup>th</sup> character?

Partly initialized since only 11 characters this phrase

Very good!



# Declaring and Using Strings

Can be initialized at time of declaration

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

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

Just like int arrays, char arrays cannot be initialized this way after declaration is done

Other ways: scanf (with %s), gets

Both are very unsafe – crash!

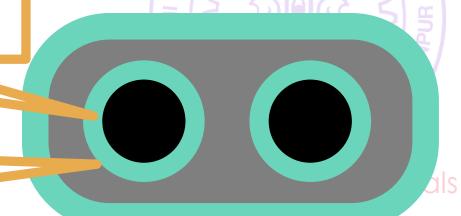
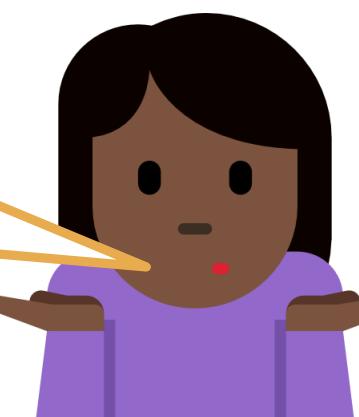
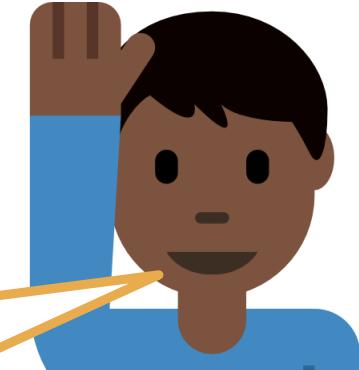
To print: puts, printf (with %s)

The space between the two words. Space is a character too!

Hello has 5 characters, World has 5, what is the 11<sup>th</sup> character?

Partly initialized since only 11 characters this phrase

Very good!



# Declaring and Using Strings

Can be initialized at time of declaration

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

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

Just like int arrays, char arrays cannot be initialized this way after declaration is done

Other ways: scanf (with %s), gets

Both are very unsafe – crash!

To print: puts, printf (with %s)

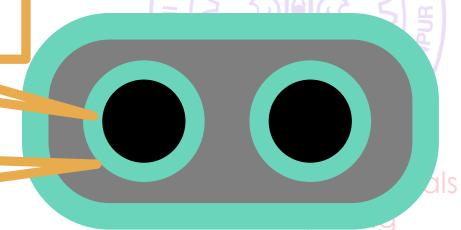
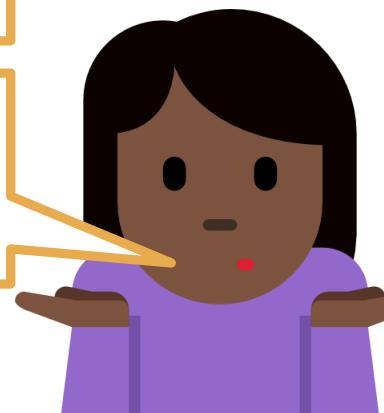
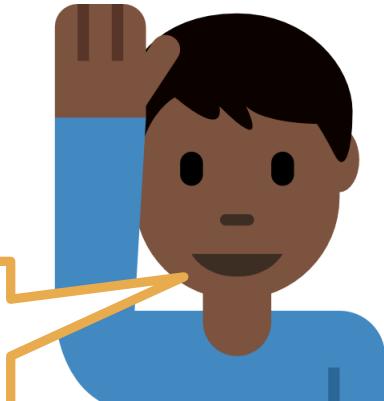
getline most powerful but have to wait for it a bit ☺

The space between the two words. Space is a character too!

Hello has 5 characters, World has 5, what is the 11<sup>th</sup> character?

Partly initialized since only 11 characters this phrase

Very good!



# The null character

32



ESC101: Fundamentals  
of Computing

# The null character

32

ASCII value 0: used to signal the end of a string



# The null character

ASCII value 0: used to signal the end of a string

Can actually print and read a null character – escape sequence \0



# The null character

ASCII value 0: used to signal the end of a string

Can actually print and read a null character – escape sequence \0

Character arrays with a null character called strings



# The null character

ASCII value 0: used to signal the end of a string

Can actually print and read a null character – escape sequence \0

Character arrays with a null character called strings

**Delimiter:** a character or symbol used to signal the end of a list or end of a stream



# The null character

ASCII value 0: used to signal the end of a string

Can actually print and read a null character – escape sequence \0

Character arrays with a null character called strings

**Delimiter:** a character or symbol used to signal the end of a list or end of a stream

In many of our lab questions where input is a list of numbers, -1 is delimiter



# The null character

ASCII value 0: used to signal the end of a string

Can actually print and read a null character – escape sequence \0

Character arrays with a null character called strings

**Delimiter:** a character or symbol used to signal the end of a list or end of a stream

In many of our lab questions where input is a list of numbers, -1 is delimiter

Stop reading numbers after -1 is encountered



# The null character

ASCII value 0: used to signal the end of a string

Can actually print and read a null character – escape sequence \0

Character arrays with a null character called strings

**Delimiter:** a character or symbol used to signal the end of a list or end of a stream

In many of our lab questions where input is a list of numbers, -1 is delimiter

Stop reading numbers after -1 is encountered

For strings null character is delimiter – Mr C stops reading after \0



# The null character

ASCII value 0: used to signal the end of a string

Can actually print and read a null character – escape sequence \0

Character arrays with a null character called strings

**Delimiter:** a character or symbol used to signal the end of a list or end of a stream

In many of our lab questions where input is a list of numbers, -1 is delimiter

Stop reading numbers after -1 is encountered

For strings null character is delimiter – Mr C stops reading after \0

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



# The null character

ASCII value 0: used to signal the end of a string

Can actually print and read a null character – escape sequence \0

Character arrays with a null character called strings

**Delimiter:** a character or symbol used to signal the end of a list or end of a stream

In many of our lab questions where input is a list of numbers, -1 is delimiter

Stop reading numbers after -1 is encountered

For strings null character is delimiter – Mr C stops reading after \0

```
char str[50] = {'H','e','l','\0','l','o',' ','W','o','r','l','d'};  
printf("%s",str);
```



# The null character

ASCII value 0: used to signal the end of a string

Can actually print and read a null character – escape sequence \0

Character arrays with a null character called strings

**Delimiter:** a character or symbol used to signal the end of a list or end of a stream

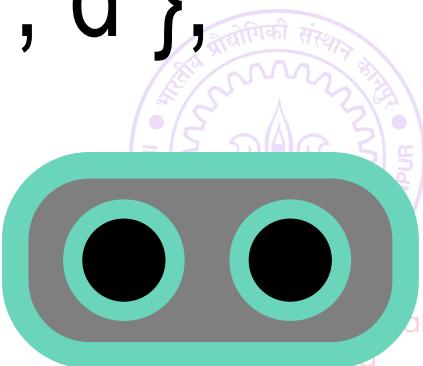
In many of our lab questions where input is a list of numbers, -1 is delimiter

Stop reading numbers after -1 is encountered

For strings null character is delimiter – Mr C stops reading after \0

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

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



# The null character

ASCII value 0: used to signal the end of a string

Can actually print and read a null character – escape sequence \0

Character arrays with a null character called strings

**Delimiter:** a character or symbol used to signal the end of a list or end of a stream

In many of our lab questions where input is a list of numbers, -1 is delimiter

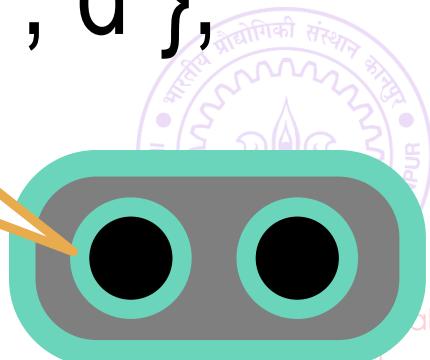
Stop reading numbers after -1 is encountered

For strings null character is delimiter – Mr C stops reading after \0

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

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

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



# The null character

ASCII value 0: used to signal the end of a string

Can actually print and read a null character – escape sequence \0

Character arrays with a null character called strings

**Delimiter:** a character or symbol used to signal the end of a list or end of a stream

In many of our lab questions where input is a list of numbers, -1 is delimiter

Stop reading numbers after -1 is encountered

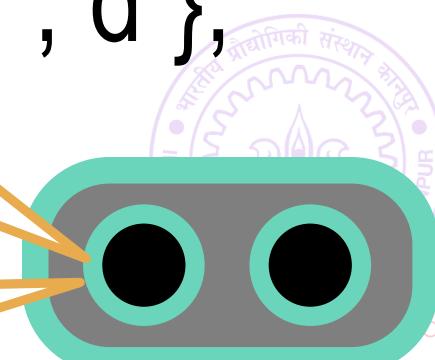
For strings null character is delimiter – Mr C stops reading after \0

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

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

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

Hel



# Mr C and the null character

45



# Mr C and the null character

45

Mr C is actually very careful delimiting strings using \0



# Mr C and the null character

45

Mr C is actually very careful delimiting strings using \0

When we say



# Mr C and the null character

45

Mr C is actually very careful delimiting strings using \0

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



# Mr C and the null character

45

Mr C is actually very careful delimiting strings using \0

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

Mr C actually stores a \0 after last character 'e' himself



# Mr C and the null character

45

Mr C is actually very careful delimiting strings using \0

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

Mr C actually stores a \0 after last character 'e' himself



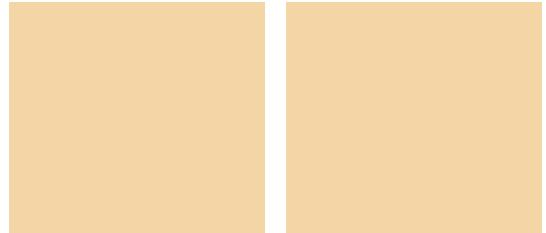
# Mr C and the null character

45

Mr C is actually very careful delimiting strings using \0

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

Mr C actually stores a \0 after last character 'e' himself



# Mr C and the null character

45

Mr C is actually very careful delimiting strings using \0

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

Mr C actually stores a \0 after last character 'e' himself



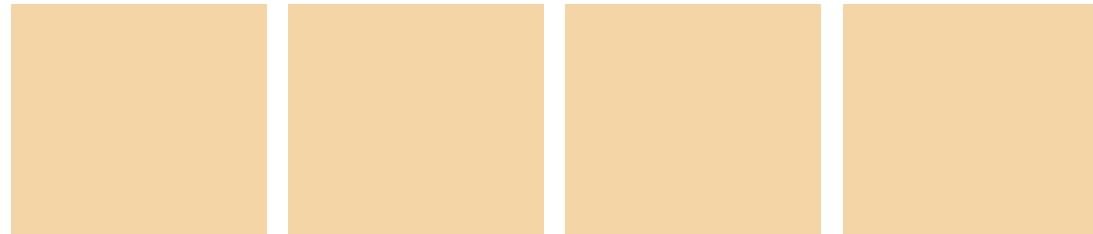
# Mr C and the null character

45

Mr C is actually very careful delimiting strings using \0

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

Mr C actually stores a \0 after last character 'e' himself



# Mr C and the null character

45

Mr C is actually very careful delimiting strings using \0

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

Mr C actually stores a \0 after last character 'e' himself



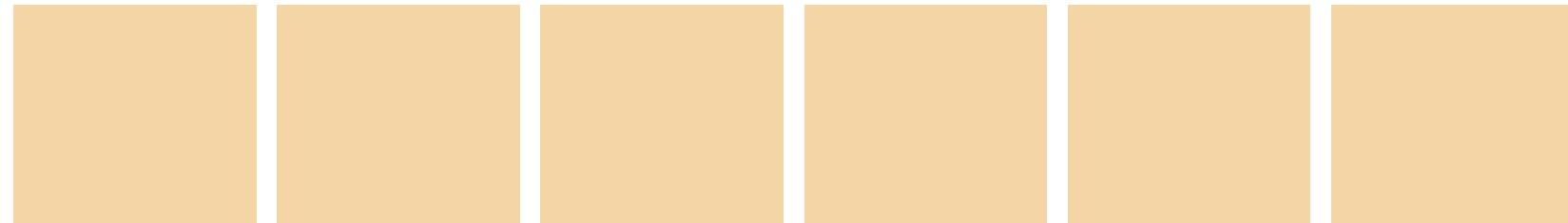
# Mr C and the null character

45

Mr C is actually very careful delimiting strings using \0

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

Mr C actually stores a \0 after last character 'e' himself



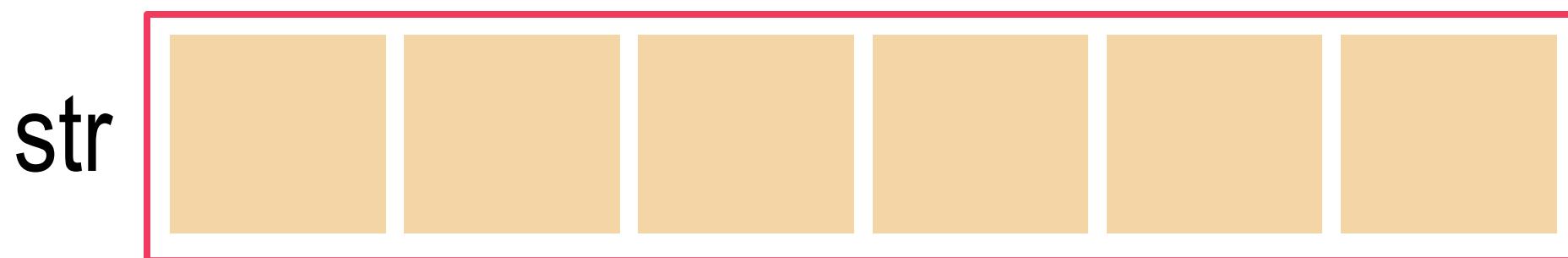
# Mr C and the null character

45

Mr C is actually very careful delimiting strings using \0

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

Mr C actually stores a \0 after last character 'e' himself



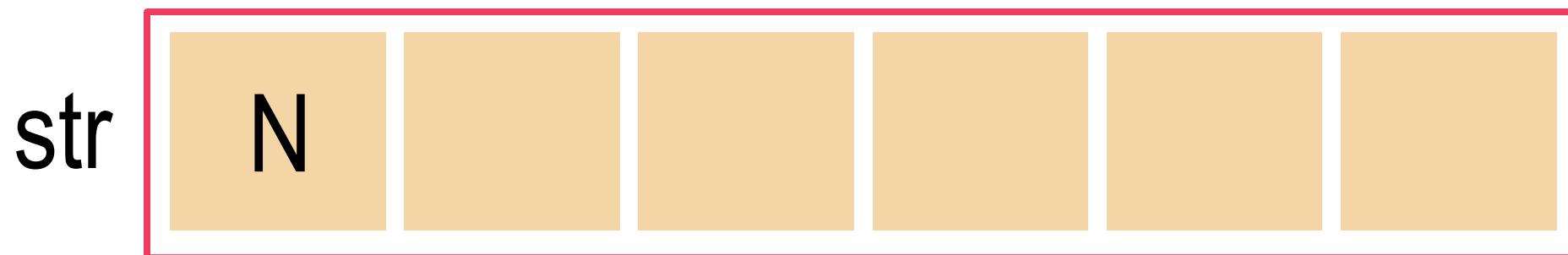
# Mr C and the null character

45

Mr C is actually very careful delimiting strings using \0

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

Mr C actually stores a \0 after last character 'e' himself



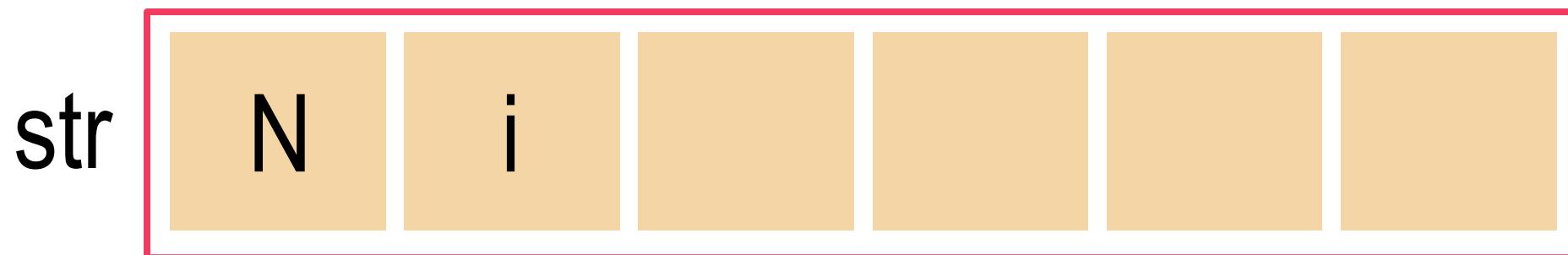
# Mr C and the null character

45

Mr C is actually very careful delimiting strings using \0

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

Mr C actually stores a \0 after last character 'e' himself



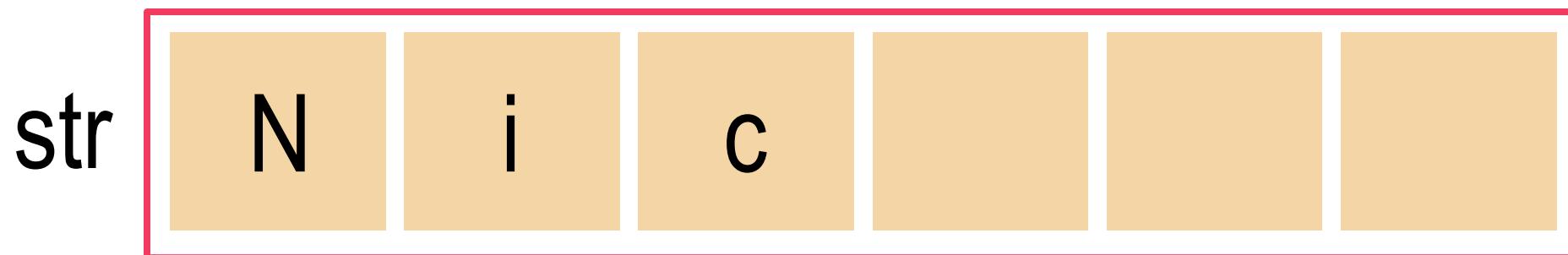
# Mr C and the null character

45

Mr C is actually very careful delimiting strings using \0

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

Mr C actually stores a \0 after last character 'e' himself



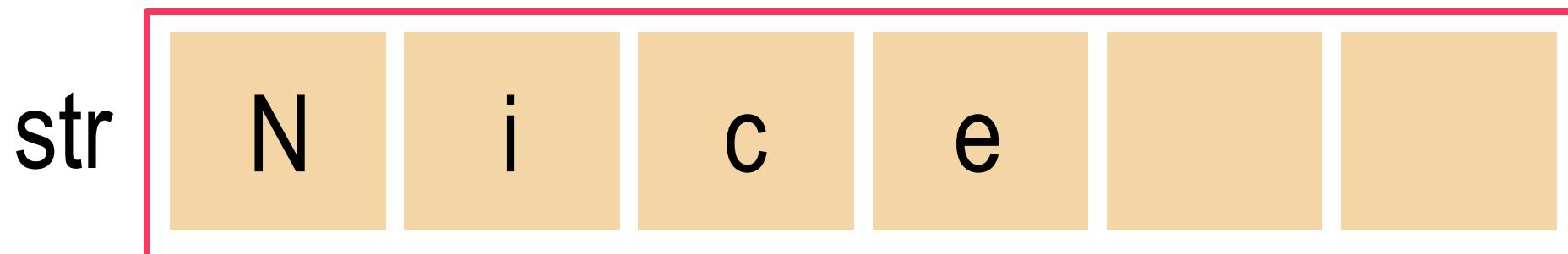
# Mr C and the null character

45

Mr C is actually very careful delimiting strings using \0

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

Mr C actually stores a \0 after last character 'e' himself



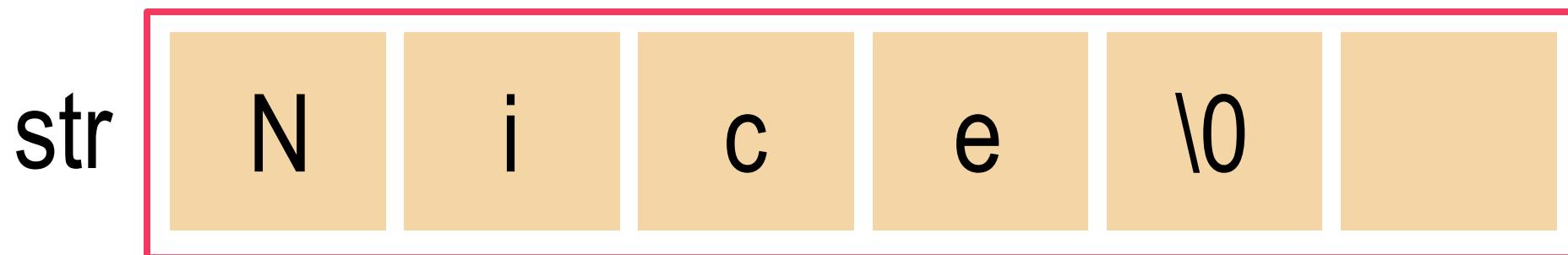
# Mr C and the null character

45

Mr C is actually very careful delimiting strings using \0

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

Mr C actually stores a \0 after last character 'e' himself



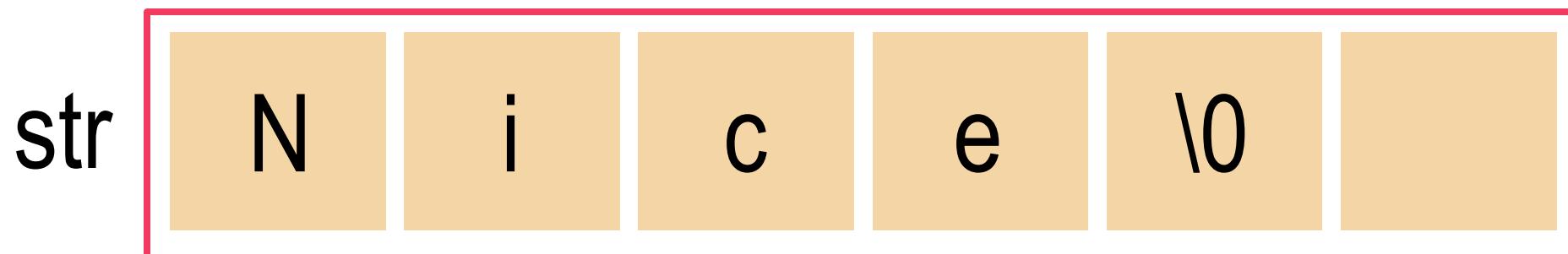
# Mr C and the null character

45

Mr C is actually very careful delimiting strings using \0

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

Mr C actually stores a \0 after last character 'e' himself



**Warning:** uninitialized character arrays contain junk



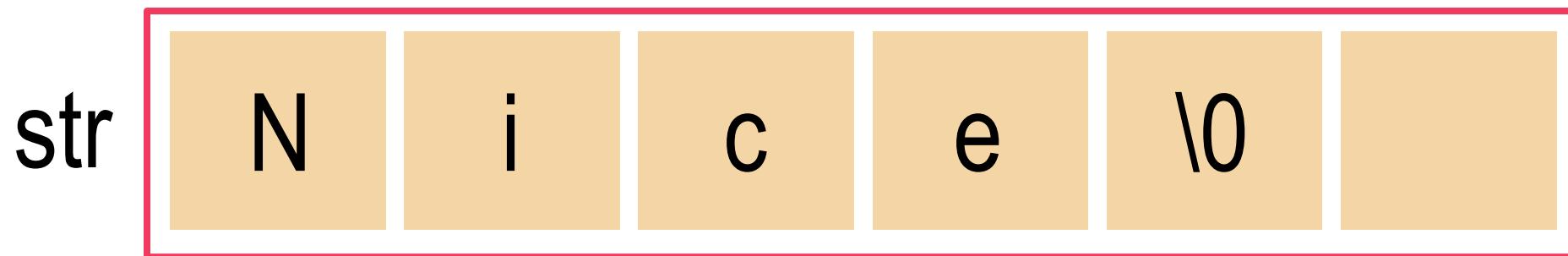
# Mr C and the null character

45

Mr C is actually very careful delimiting strings using \0

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

Mr C actually stores a \0 after last character 'e' himself



**Warning:** uninitialized character arrays contain junk

```
char str = "A";
```

```
putchar(str);
```



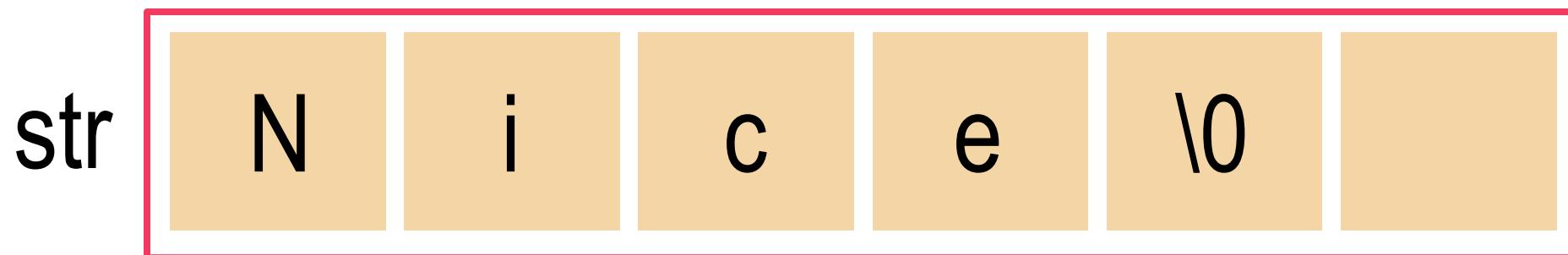
# Mr C and the null character

45

Mr C is actually very careful delimiting strings using \0

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

Mr C actually stores a \0 after last character 'e' himself



**Warning:** uninitialized character arrays contain junk

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



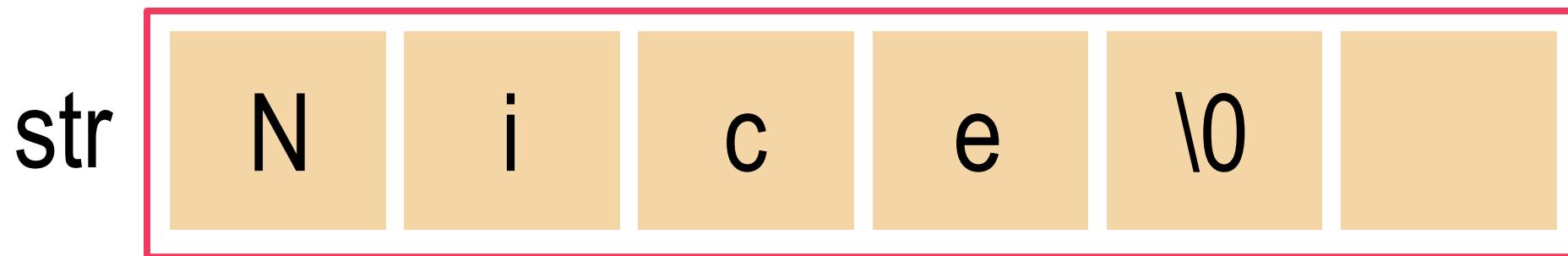
# Mr C and the null character

45

Mr C is actually very careful delimiting strings using \0

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

Mr C actually stores a \0 after last character 'e' himself



**Warning:** uninitialized character arrays contain junk

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



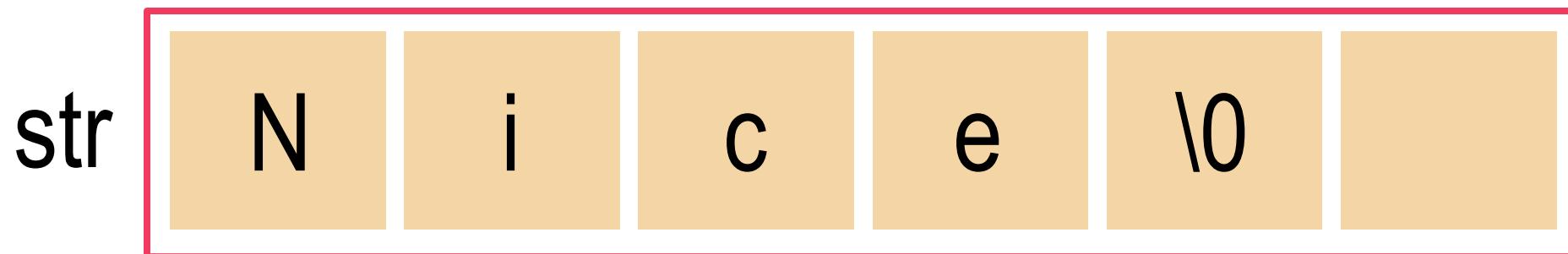
# Mr C and the null character

45

Mr C is actually very careful delimiting strings using \0

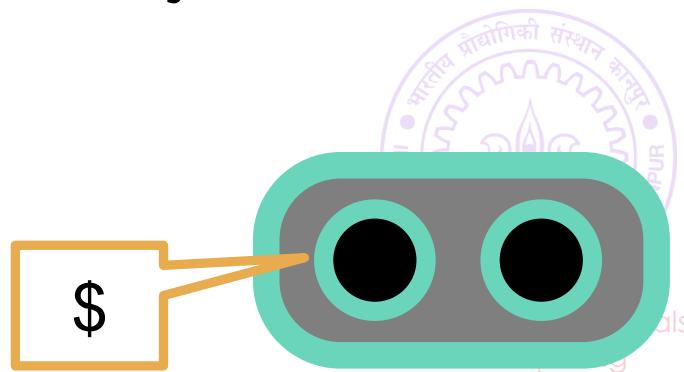
When we say **char str[6] = "Nice";**

Mr C actually stores a \0 after last character 'e' himself



**Warning:** uninitialized character arrays contain junk

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



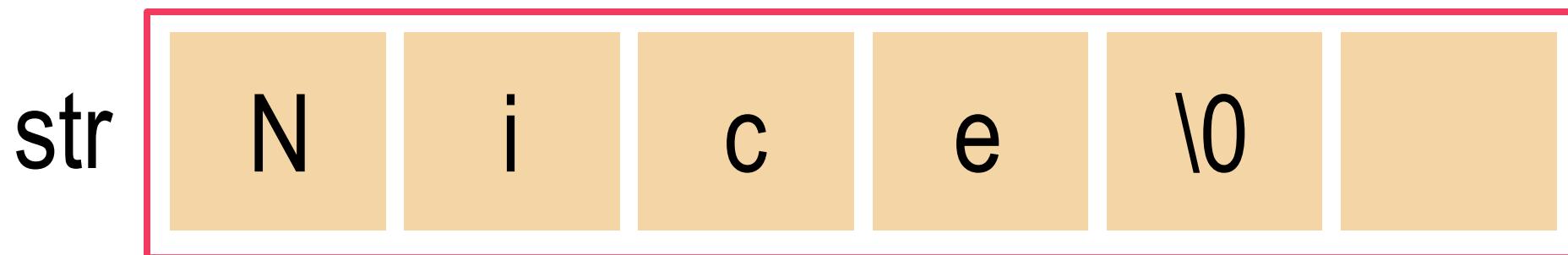
# Mr C and the null character

45

Mr C is actually very careful delimiting strings using \0

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

Mr C actually stores a \0 after last character 'e' himself



**Warning:** uninitialized character arrays are junk

char str = "A";        
putchar(str);

Strings are character arrays. "A" is a string.  
'A' is a character

\$



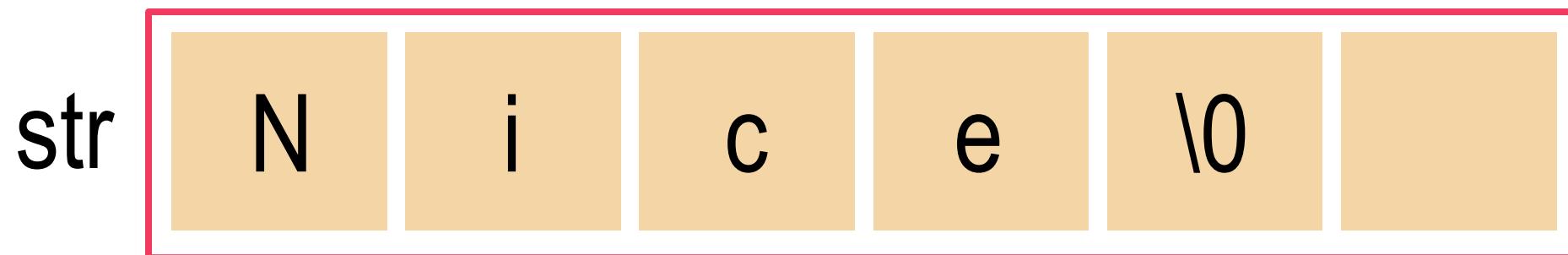
# Mr C and the null character



Mr C is actually very careful delimiting strings using \

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

Mr C actually stores a \0 after last character 'e' himself



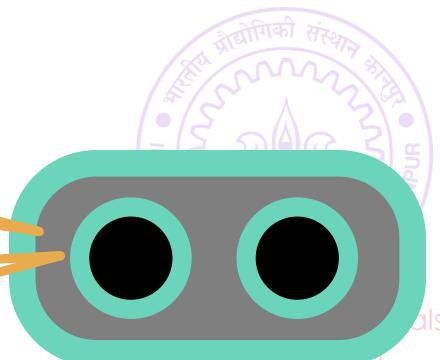
**Warning:** uninitialized character

**char str = "A";**

**putchar(str);**



Strings are character arrays. "A" is a string.  
'A' is a character



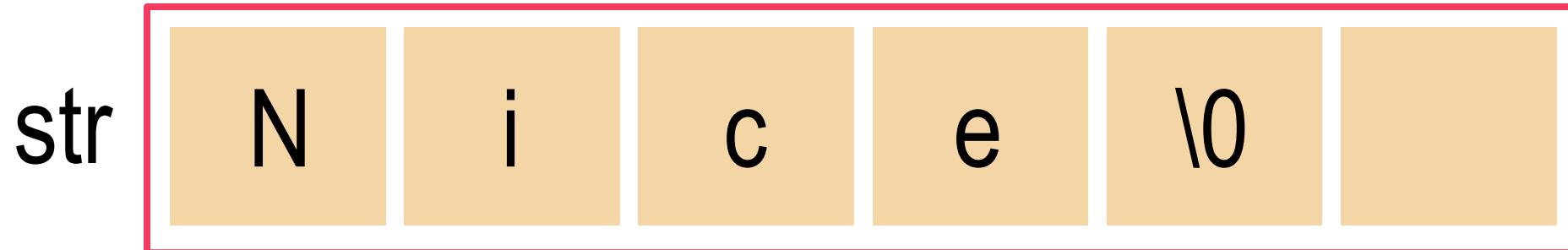
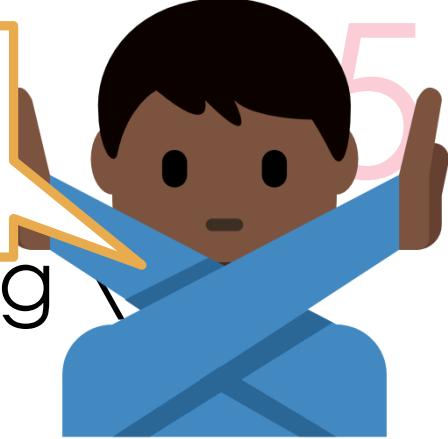
# Mr C and the null character

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

Mr C is actually very careful delimiting strings using \0

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

Mr C actually stores a \0 after last character 'e' himself



**Warning:** uninitialized character arrays are junk

char str = "A";



putchar(str);

Strings are character arrays. "A" is a string.  
'A' is a character

\$



# Mr C and the null character

70



# Mr C and the null character

70

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



# Mr C and the null character

70

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";
```



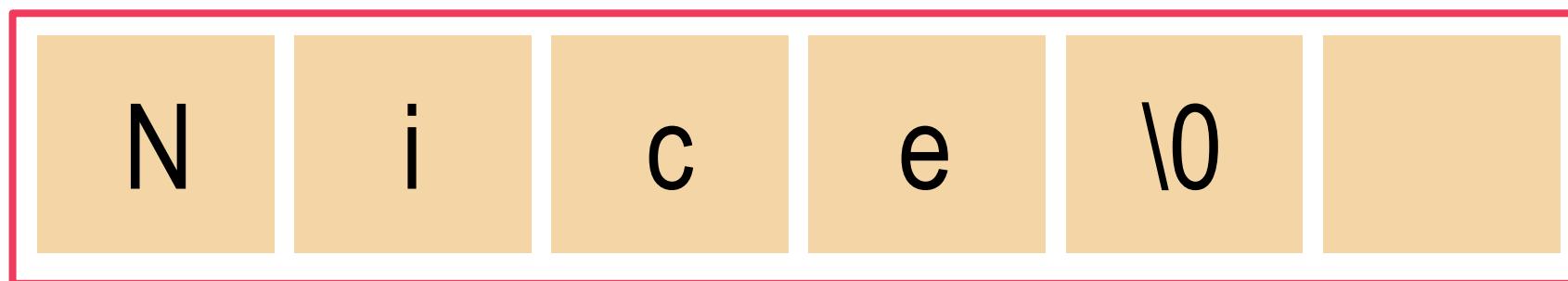
# Mr C and the null character

70

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";
```

str



# Mr C and the null character

70

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";
```

str [ N | i | c | e | \0 | ]

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



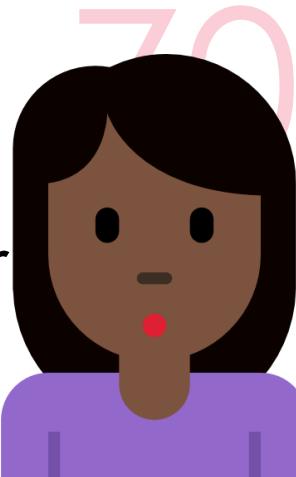
# Mr C and the null character

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";
```

str [ N | i | c | e | \0 | ]

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



# Mr C and the null character

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";
```

str [ N | i | c | e | \0 | ]

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

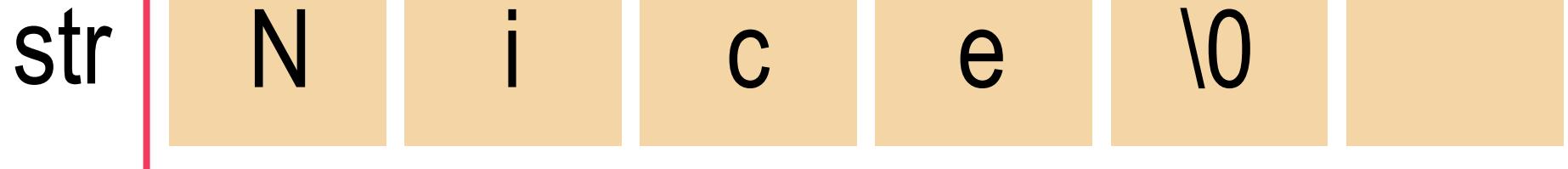


# Mr C and the null character

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";
```



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

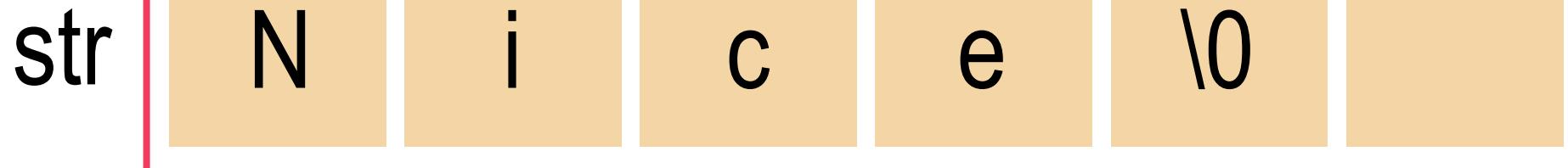


# Mr C and the null character

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";
```



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

No, since str  
is an array



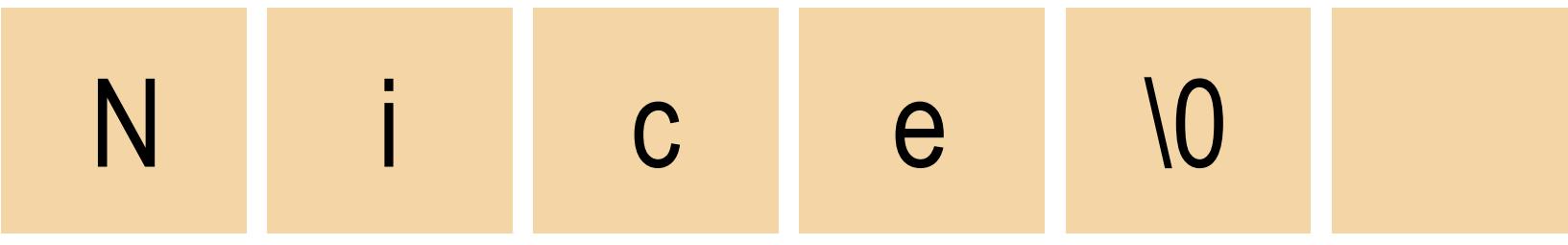
# Mr C and the null character

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";
```

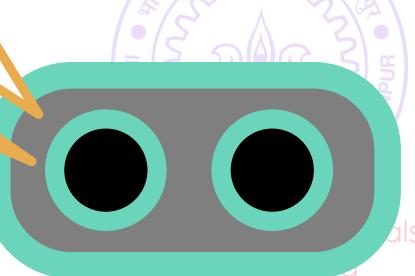
str



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

Will learn about  
this in a few weeks

No, since str  
is an array



70



# Mr C and the null character

We did not write  
&str in scanf?

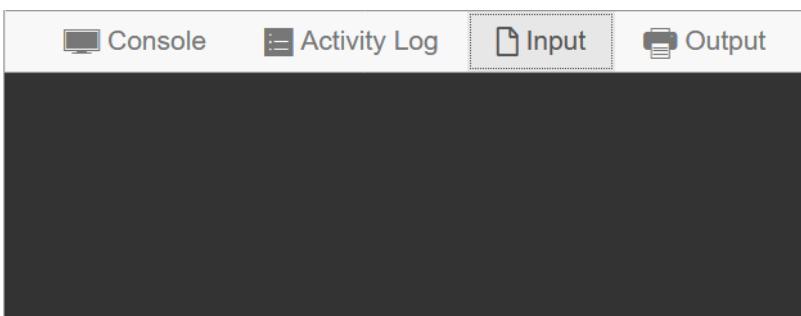
In fact when we read a string using gets or scanf, Mr C yet again automatically puts a \0 at the end

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

str

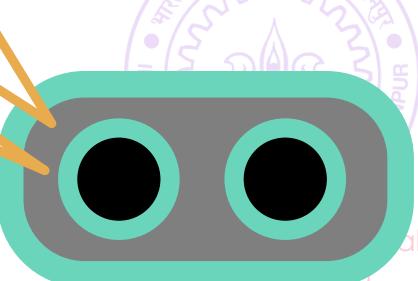
N	i	c	e	\0	
---	---	---	---	----	--

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



Will learn about  
this in a few weeks

No, since str  
is an array



# Mr C and the null character

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";
```

str

N

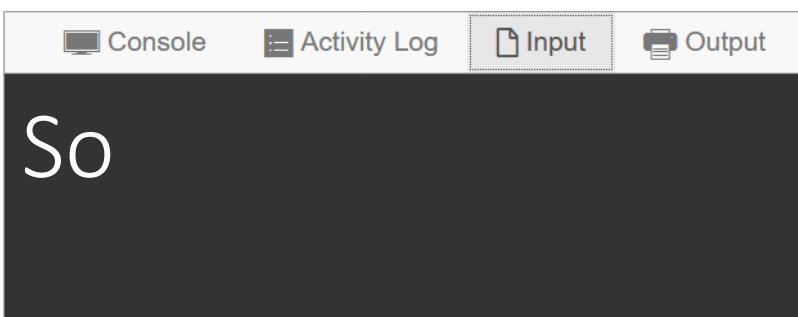
i

c

e

\0

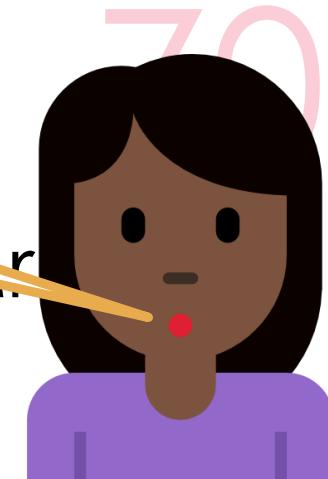
```
scanf("%s",str);
```



So

Will learn about  
this in a few weeks

No, since str  
is an array



# Mr C and the null character

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";
```

str

S

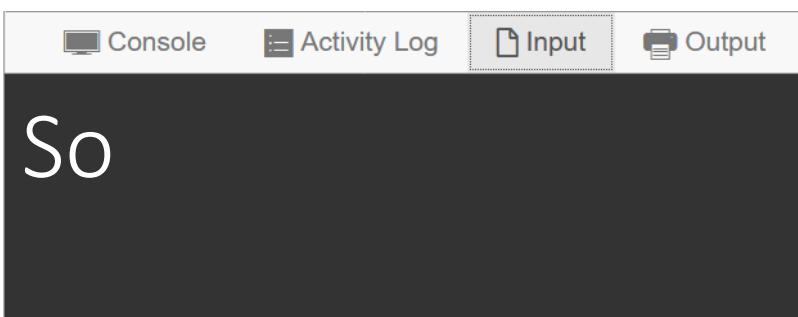
o

\0

e

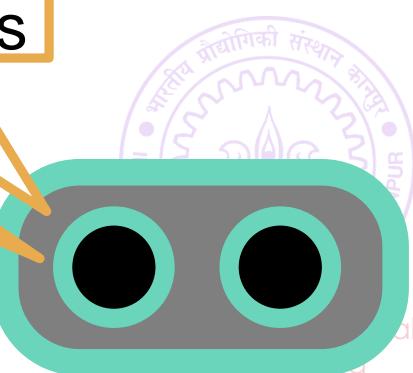
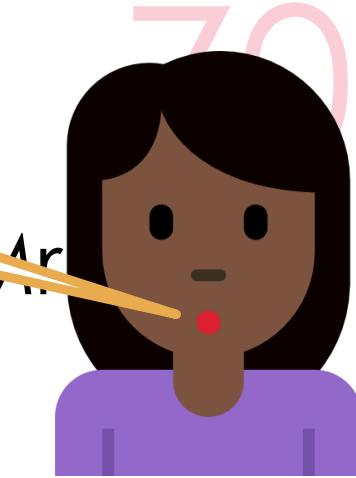
\0

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



Will learn about  
this in a few weeks

No, since str  
is an array



# Mr C and the null character

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";
```

str

S

o

\0

e

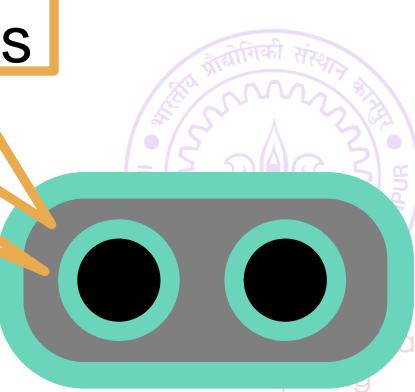
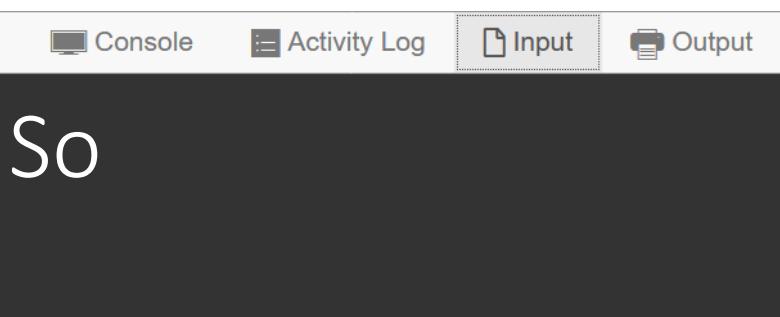
\0

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

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

Will learn about  
this in a few weeks

No, since str  
is an array



# Mr C and the null character

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";
```

str

S

o

\0

e

\0

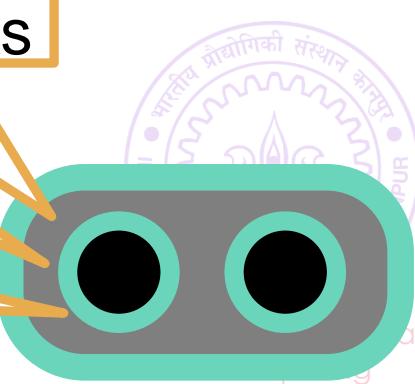
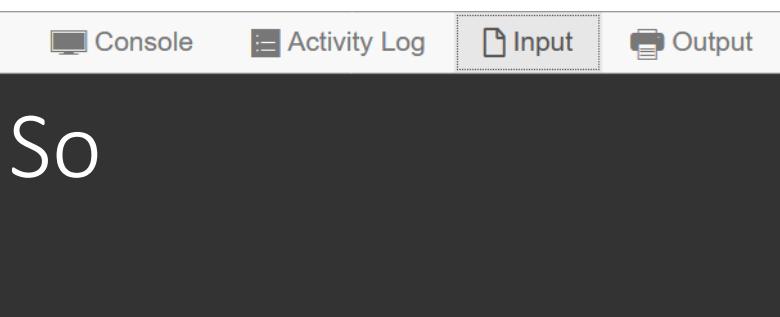
```
scanf("%s",str);
```

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

Will learn about  
this in a few weeks

No, since str  
is an array

So



# Mr C and the null character

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";
```

str

S

o

\0

e

\0

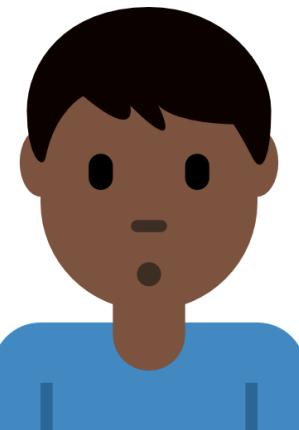
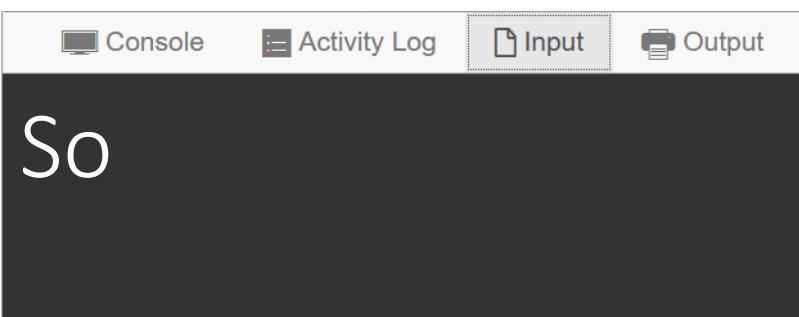
```
scanf("%s",str);
```

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

Will learn about  
this in a few weeks

No, since str  
is an array

So

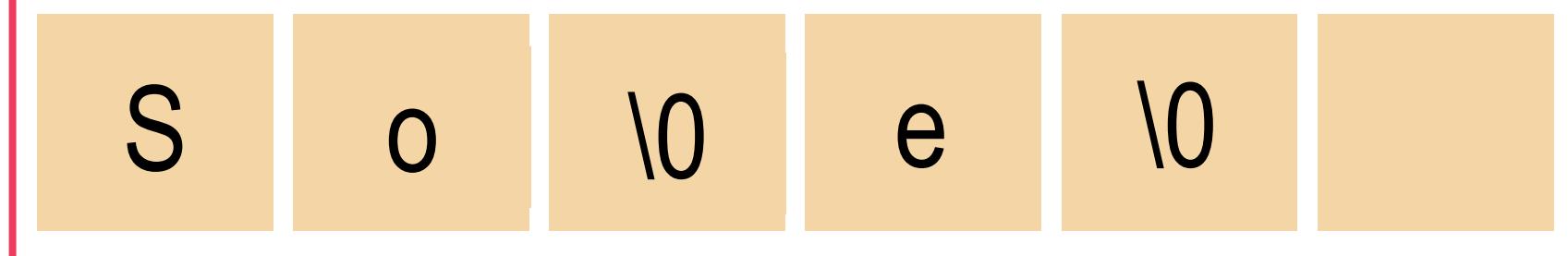


# Mr C and the null character

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";
```

str



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

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

We did not write  
&str in scanf?

The rest of the char  
array is still there

So

Will learn about  
this in a few weeks

No, since str  
is an array

So

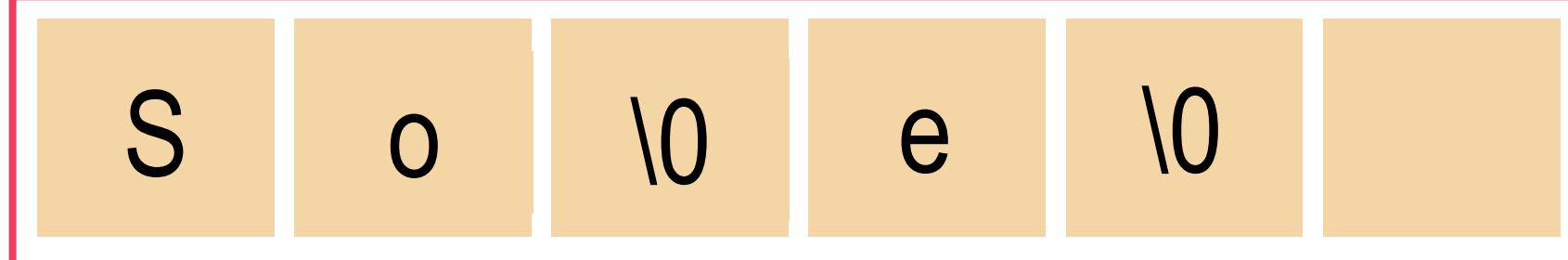


# Mr C and the null character

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";
```

str



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

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

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

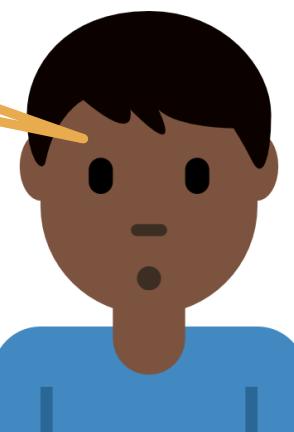
We did not write  
&str in scanf?

The rest of the char  
array is still there

Will learn about  
this in a few weeks

No, since str  
is an array

So



# Care with Char Arrays

88



# Care with Char Arrays

88

Remember this extra \0 that always gets appended at the end of every string when using gets, scanf, getline etc



# Care with Char Arrays

88

Remember this extra \0 that always gets appended at the end of every string when using gets, scanf, getline etc

If you are expecting the user to enter a string of 1000 characters, your char array should have size at least 1001



# Care with Char Arrays

88

Remember this extra \0 that always gets appended at the end of every string when using gets, scanf, getline etc

If you are expecting the user to enter a string of 1000 characters, your char array should have size at least 1001

The last character is required to store the delimiter \0



# Care with Char Arrays

88

Remember this extra \0 that always gets appended at the end of every string when using gets, scanf, getline etc

If you are expecting the user to enter a string of 1000 characters, your char array should have size at least 1001

The last character is required to store the delimiter \0

Functions that handle strings like printf (and many others we will see in next lecture) may crash if there is no \0



# Care with Char Arrays

88

Remember this extra \0 that always gets appended at the end of every string when using gets, scanf, getline etc

If you are expecting the user to enter a string of 1000 characters, your char array should have size at least 1001

The last character is required to store the delimiter \0

Functions that handle strings like printf (and many others we will see in next lecture) may crash if there is no \0

These functions will keep on accessing array elements till they find a \0 and if there is no \0, segmentation fault!



# Care with Char Arrays

88

Remember this extra \0 that always gets appended at the end of every string when using gets, scanf, getline etc

If you are expecting the user to enter a string of 1000 characters, your char array should have size at least 1001

The last character is required to store the delimiter \0

Functions that handle strings like printf (and many others we will see in next lecture) may crash if there is no \0

These functions will keep on accessing array elements till they find a \0 and if there is no \0, segmentation fault!

Since the functions will start accessing elements outside the array



# Care with Char Arrays

88

Remember this extra \0 that always gets appended at the end of every string when using gets, scanf, getline etc

If you are expecting the user to enter a string of 1000 characters, your char array should have size at least 1001

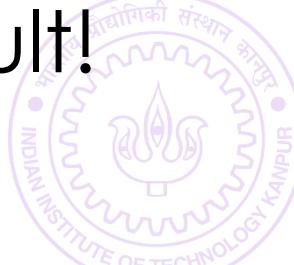
The last character is required to store the delimiter \0

Functions that handle strings like printf (and many others we will see in next lecture) may crash if there is no \0

These functions will keep on accessing array elements till they find a \0 and if there is no \0, segmentation fault!

Since the functions will start accessing elements outside the array

Prutor will show runtime error if there is a segfault



# scanf with strings

96



ESC101: Fundamentals  
of Computing

# scanf with strings

Use %s to read string from input



# scanf with strings

Use %s to read string from input

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

96



# scanf with strings

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

96

Use %s to read string from input

No & needed since char array getting passed



# scanf with strings

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

96

Use %s to read string from input

No & needed since char array getting passed

Mr C will automatically append a \0 at the end



# scanf with strings

scanf("%s",str);

96

Use %s to read string from input

No & needed since char array getting passed

Mr C will automatically append a \0 at the end

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



# scanf with strings

96

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

Use %s to read string from input

No & needed since char array getting passed

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 space in char array – segmentation fault!



# scanf with strings

96

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

Use %s to read string from input

No & needed since char array getting passed

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 space in char array – segmentation fault!

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



# scanf with strings

96

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

Use %s to read string from input

No & needed since char array getting passed

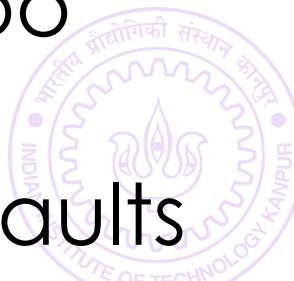
Mr C will automatically append a \0 at the end

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

**Very Risky:** if user enters more characters than space in 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



# gets with strings

105



ESC101: Fundamentals  
of Computing

# gets with strings

Shortcut to read a single line of input  
read all characters till \n – **don't store the \n throw it away**



# gets with strings

105

gets(str);

Shortcut to read a single line of input

read all characters till \n – **don't store the \n throw it away**



# gets with strings

105

gets(str);

Shortcut to read a single line of input  
read all characters till \n – **don't store the \n throw it away**

No & needed since char array getting passed



# gets with strings

105  
gets(str);

Shortcut to read a single line of input  
read all characters till \n – **don't store the \n throw it away**

No & needed since char array getting passed

Mr C will automatically append a \0 at the end



# gets with strings

105  
gets(str);

Shortcut to read a single line of input  
read all characters till \n – **don't store the \n throw it away**

No & needed since char array getting passed

Mr C will automatically append a \0 at the end

**Advantage:** does not stop reading on seeing space or \t



# gets with strings

105  
gets(str);

Shortcut to read a single line of input  
read all characters till \n – **don't store the \n throw it away**

No & needed since char array getting passed

Mr C will automatically append a \0 at the end

**Advantage:** does not stop reading on seeing space or \t

**Very Risky:** if user enters many more characters than  
space in char array – segmentation fault!



# gets with strings

105  
gets(str);

Shortcut to read a single line of input  
read all characters till \n – **don't store the \n throw it away**

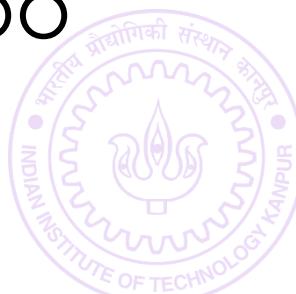
No & needed since char array getting passed

Mr C will automatically append a \0 at the end

**Advantage:** does not stop reading on seeing space or \t

**Very Risky:** if user enters many more characters than  
space in char array – segmentation fault!

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



# gets with strings

gets(str); 105

Shortcut to read a single line of input  
read all characters till \n – **don't store the \n throw it away**

No & needed since char array getting passed

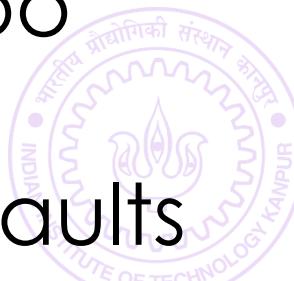
Mr C will automatically append a \0 at the end

**Advantage:** does not stop reading on seeing space or \t

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



# gets with strings

105  
gets(str);

Shortcut to read a single line of input  
read all characters till \n – **don't store the \n throw it away**

No & neede

Mr C will aut

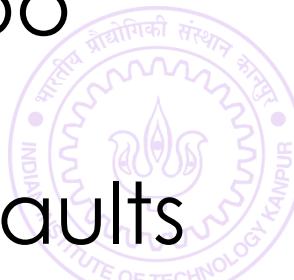
**Advantage:**

gets is deprecated in Clang  
Do not use it regularly!

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



# gets with strings

105  
gets(str);

Shortcut to read a single line of input  
read all characters till \n – **don't store the \n throw it away**

No & neede

Mr C will aut

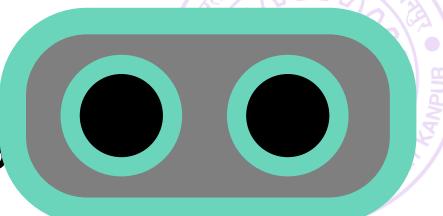
**Advantage:**

gets is deprecated in Clang  
Do not use it regularly!

**Very Risky:** if user enters many more characters than space in 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 s



# gets with strings

105  
gets(str);

Shortcut to read a single line of input  
read all characters till \n – **don't store the \n throw it away**

No & neede

Mr C will aut

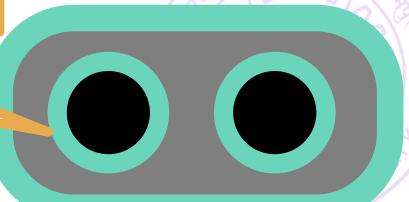
**Advantage:**

gets is deprecated in Clang  
Do not use it regularly!

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

**Caution:** Prutor **will** many more characters than spaces too

gcc and other industrial compilers will also give s



# getline with strings

117



ESC101: Fundamentals  
of Computing

# getline with strings

A much **safer** version of gets



# getline with strings

A much **safer** version of gets

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



# getline with strings

A much **safer** version of gets

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

Mr C will automatically append a \0 at the end



# getline with strings

A much **safer** version of gets

Reads a single line of input into the character array i.e.  
read all characters till \n – **don't store the \n throw 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



# getline with strings

A much **safer** version of gets

Reads a single line of input into the character array i.e.  
read all characters till \n – **don't store the \n throw 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



# getline with strings

A much **safer** version of gets

Reads a single line of input into the character array i.e.  
read all characters till \n – **don't store the \n throw 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**

