# Wrapping up Strings

ESC101: Fundamentals of Computing

Purushottam Kar

## Announcements

- Please see schedule on website for holidays/exams
- Mid-sem lab exam, mid-sem theory exam marks will be declared within this week
- Pending lab/minor quiz grades also to be declared
- Will resume schedule of practice problems etc
- Advanced track has started – some nice projects
- Slight adjustments needed in grouping – mail today!

# Strings

## Strings

**String**: a character array delimited with a NULL character

## Strings

**String**: a character array delimited with a NULL character

# Strings

**String**: a character array delimited with a NULL character

# Strings

**String**: a character array delimited with a NULL character

# Strings

**String**: a character array delimited with a NULL character

## Strings

**String**: a character array delimited with a NULL character

# Strings

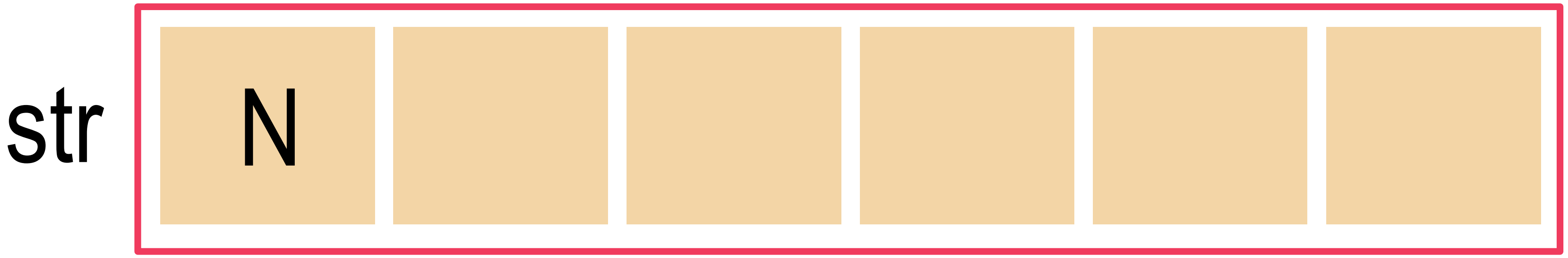
**String**: a character array delimited with a NULL character

# Strings

**String**: a character array delimited with a NULL character

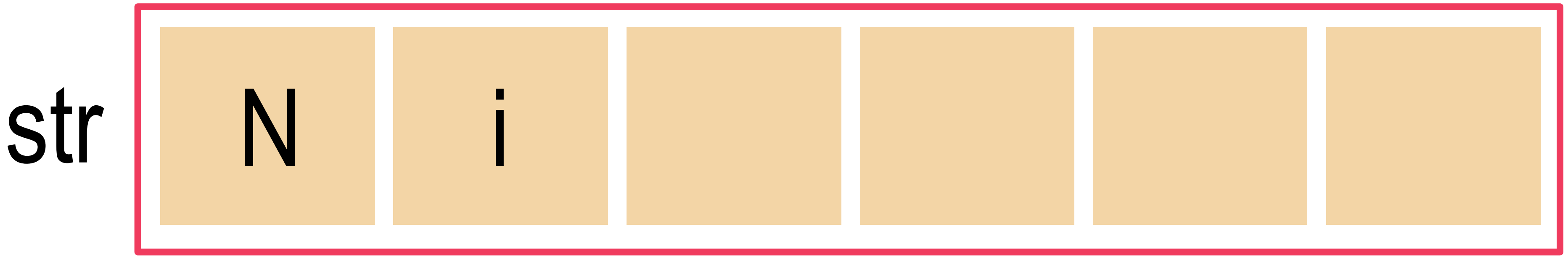
## Strings

**String**: a character array delimited with a NULL character



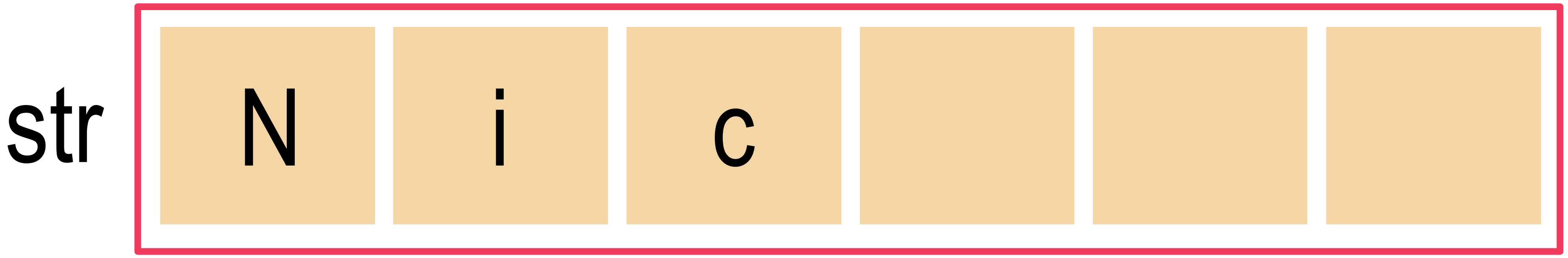
## Strings

**String**: a character array delimited with a NULL character



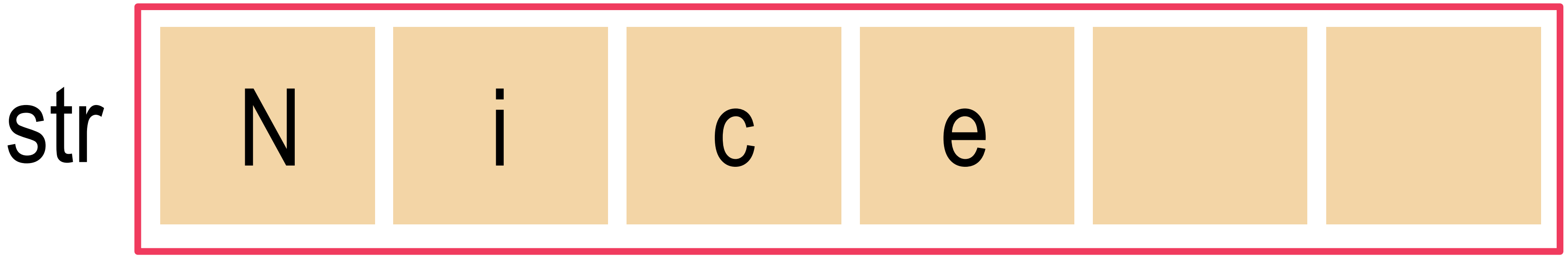
# Strings

**String**: a character array delimited with a NULL character



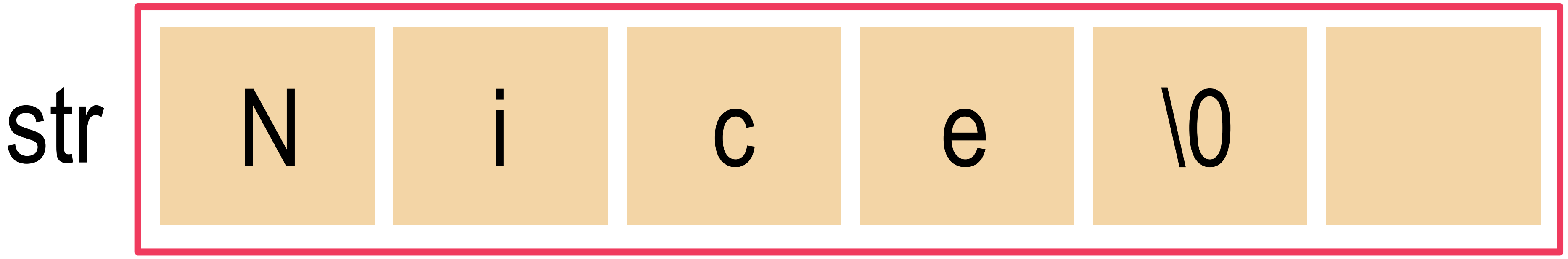
## Strings

**String**: a character array delimited with a NULL character



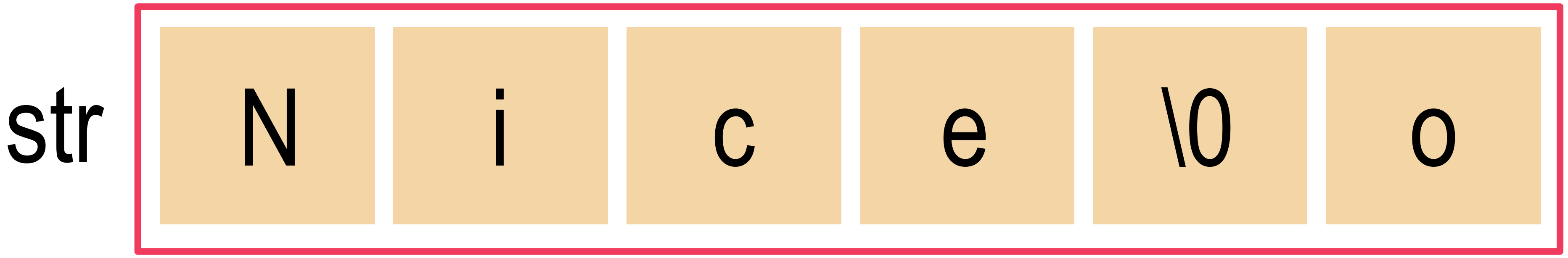
# Strings

**String**: a character array delimited with a NULL character



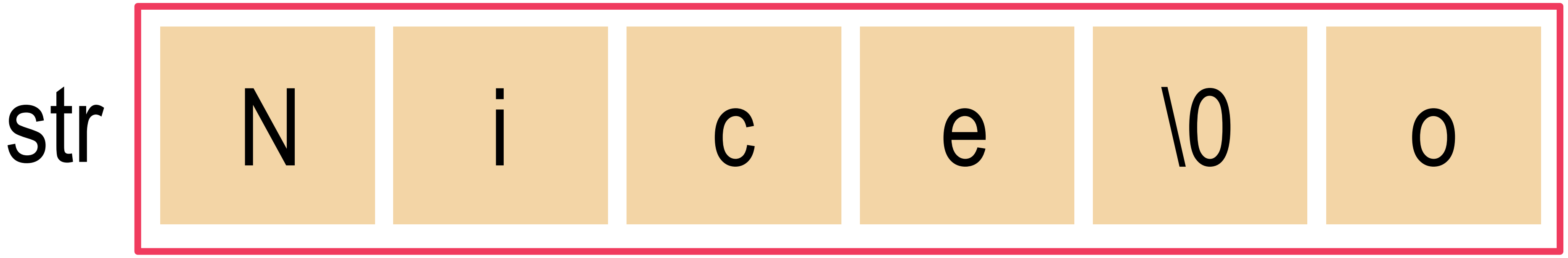
## Strings

**String**: a character array delimited with a NULL character



## Strings

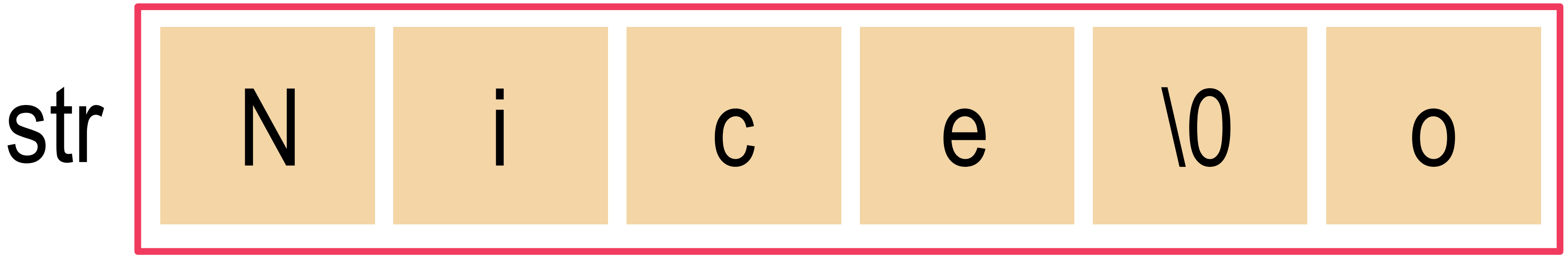
**String**: a character array delimited with a NULL character



In functions like printf, chars after NULL ignored

# Strings

**String**: a character array delimited with a NULL character

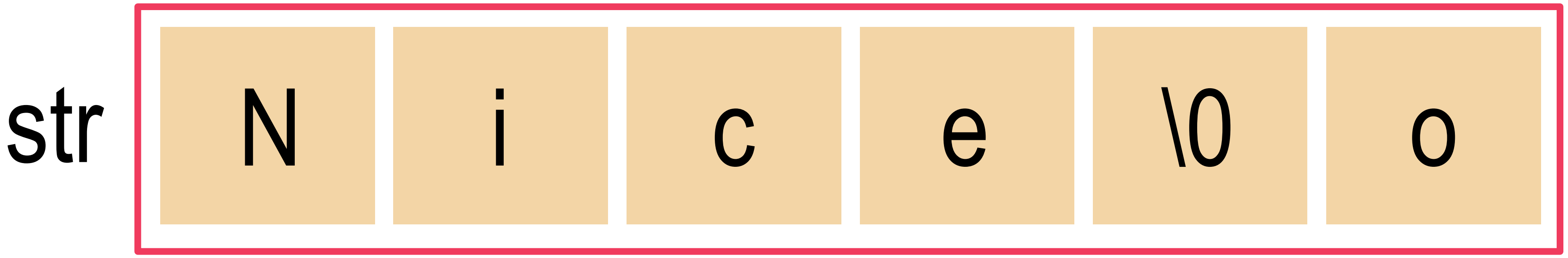


In functions like printf, chars after NULL ignored

**Substring**: a contiguous subsequence of a string

# Strings

**String**: a character array delimited with a NULL character



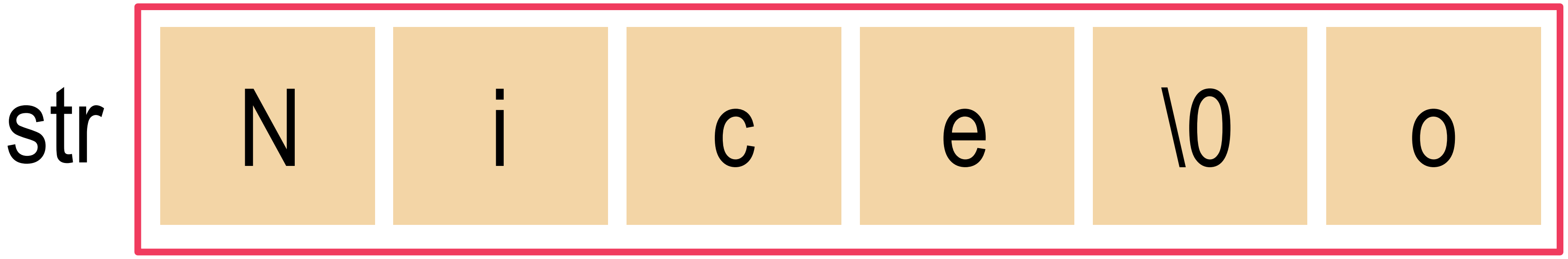
In functions like printf, chars after NULL ignored

**Substring**: a contiguous subsequence of a string

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

## Strings

**String**: a character array delimited with a NULL character



In functions like printf, chars after NULL ignored

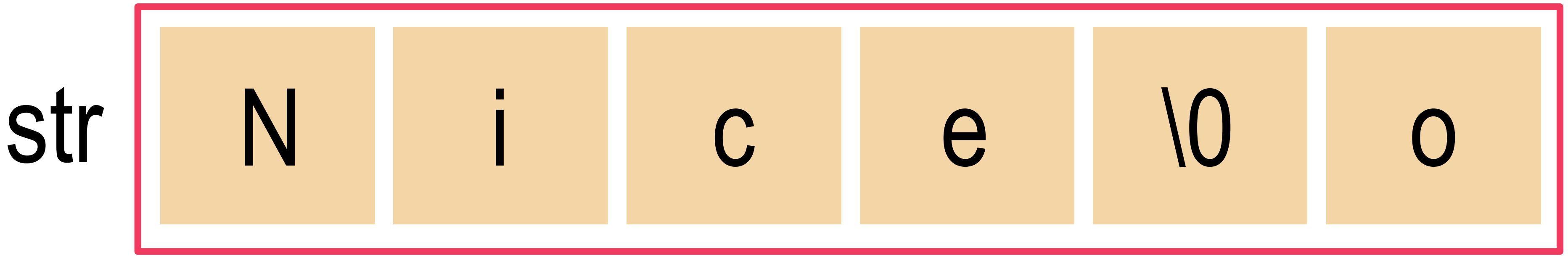
**Substring**: a contiguous subsequence of a string

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

## Strings

**String**: a character array delimited with a NULL character



In functions like printf, chars after NULL ignored

**Substring**: a contiguous subsequence of a string

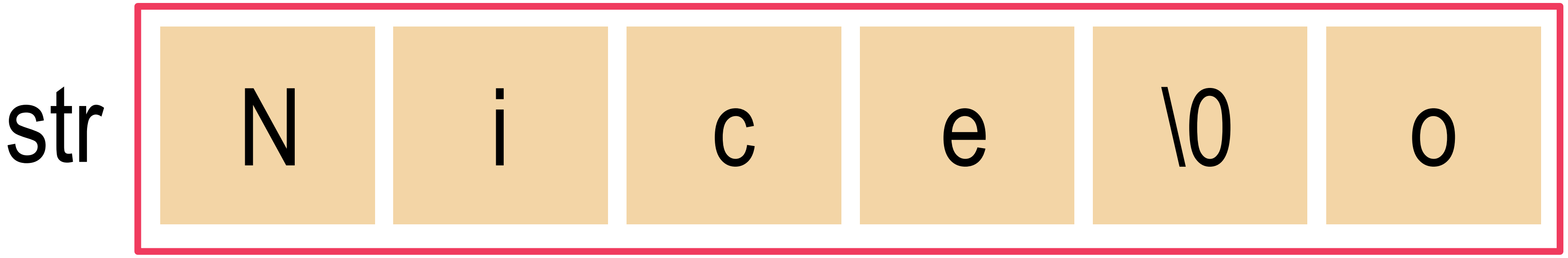
E.g. "Nice", "Nic", "ice", "ce", "c", "Ni" are substrings of the above string

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

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

# Strings

**String**: a character array delimited with a NULL character



In functions like printf, chars after NULL ignored

**Substring**: a contiguous subsequence of a string

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

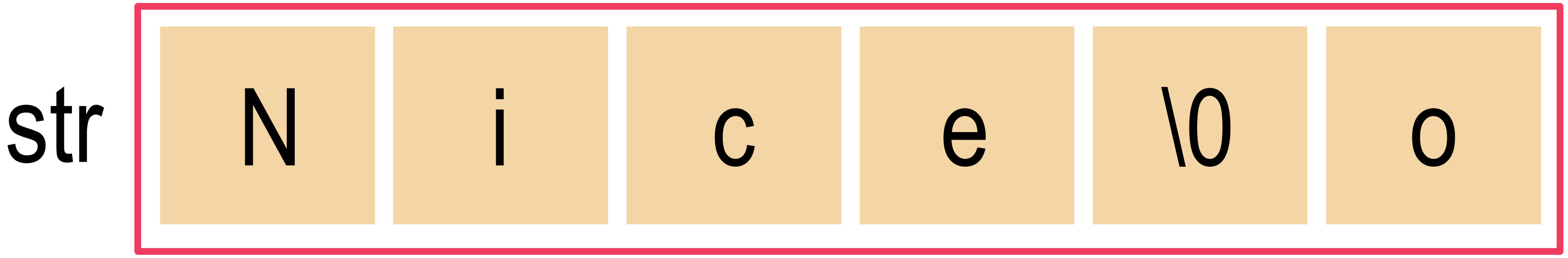
"Nce", "Nie", "ie", "Ne" **NOT substrings** (not contiguous) of above string

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

Substrings need not contain the NULL character – WARNING!

# Strings

**String**: a character array delimited with a NULL character



In functions like printf, chars after NULL ignored

**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

# EOF (end of file)

# EOF (end of file)

A very special non-character – cannot be printed

## EOF (end of file)

A very special non-character – cannot be printed

Signals end of input (no more characters in input)

## EOF (end of file)

A very special non-character – cannot be printed

Signals end of input (no more characters in input)

stdio.h gives you a named constant **EOF** for convenience

# EOF (end of file)

A very special non-character – cannot be printed

Signals end of input (no more characters in input)

stdio.h gives you a named constant **EOF** for convenience

Has no ASCII value – but internally stored as -1

# EOF (end of file)

A very special non-character – cannot be printed

Signals end of input (no more characters in input)

stdio.h gives you a named constant **EOF** for convenience

Has no ASCII value – but internally stored as -1

Recall characters have ASCII values from 0 to 127 only

## EOF (end of file)

A very special non-character – cannot be printed

Signals end of input (no more characters in input)

stdio.h gives you a named constant **EOF** for convenience

Has no ASCII value – but internally stored as -1

Recall characters have ASCII values from 0 to 127 only

getchar() will read EOF as a character but not scanf/gets

# EOF (end of file)

A very special non-character – cannot be printed

Signals end of input (no more characters in input)

stdio.h gives you a named constant **EOF** for convenience

Has no ASCII value – but internally stored as -1

Recall characters have ASCII values from 0 to 127 only

getchar() will read EOF as a character but not scanf/gets

# EOF (end of file)

A very special non-character – cannot be printed

Signals end of input (no more characters in input)

stdio.h gives you a named constant **EOF** for convenience

Has no ASCII value – but internally stored as -1

Recall characters have ASCII values from 0 to 127 only

getchar() will read EOF as a character but not scanf/gets

Be careful, do not confuse EOF with NULL and \n.

# EOF (end of file)

A very special non-character – cannot be printed

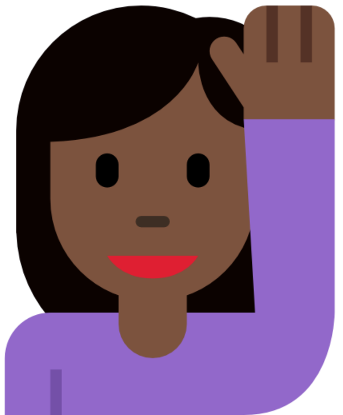
Signals end of input (no more characters in input)

stdio.h gives you a named constant **EOF** for convenience

Has no ASCII value – but internally stored as -1

Recall characters have ASCII values from 0 to 127 only

getchar() will read EOF as a character but not scanf/gets



Be careful, do not confuse EOF with NULL and \n.

# EOF (end of file)

A very special non-character – cannot be printed

Signals end of input (no more characters in input)

stdio.h gives you a named constant **EOF** for convenience

Has no ASCII value – but internally stored as -1

Recall characters have ASCII values from 0 to 127 only

getchar() will read EOF as a character but not scanf/gets

NULL and \n are valid characters with proper ASCII values

Be careful, do not confuse EOF with NULL and \n.

## Keep in mind when using strings

## Keep in mind when using strings

Use sufficiently big character arrays when reading input

# Keep in mind when using strings

- Use sufficiently big character arrays when reading input
  - Short arrays can cause segfaults or weird behaviour

## Keep in mind when using strings

- Use sufficiently big character arrays when reading input
  - Short arrays can cause segfaults or weird behaviour
- EOF is not a substitute for NULL

# Keep in mind when using strings

- Use sufficiently big character arrays when reading input
  - Short arrays can cause segfaults or weird behaviour
- EOF is not a substitute for NULL
  - Do not store EOF in a char array and expect Mr C to treat it like NULL

## Keep in mind when using strings

- Use sufficiently big character arrays when reading input
  - Short arrays can cause segfaults or weird behaviour
- EOF is not a substitute for NULL
  - Do not store EOF in a char array and expect Mr C to treat it like NULL
  - Strings must be delimited using only NULL, never EOF or \n or anything else

## Keep in mind when using strings

Use sufficiently big character arrays when reading input

- Short arrays can cause segfaults or weird behaviour

EOF is not a substitute for NULL

- Do not store EOF in a char array and expect Mr C to treat it like NULL
- Strings must be delimited using only NULL, never EOF or \n or anything else

When using scanf/printf be careful not to mix %c and %s

# Keep in mind when using strings

- Use sufficiently big character arrays when reading input
  - Short arrays can cause segfaults or weird behaviour
- EOF is not a substitute for NULL
  - Do not store EOF in a char array and expect Mr C to treat it like NULL
  - Strings must be delimited using only NULL, never EOF or \n or anything else
- When using scanf/printf be careful not to mix %c and %s
  - %c is used to read into, or print, a single character variable

## Keep in mind when using strings

- Use sufficiently big character arrays when reading input
  - Short arrays can cause segfaults or weird behaviour
- EOF is not a substitute for NULL
  - Do not store EOF in a char array and expect Mr C to treat it like NULL
  - Strings must be delimited using only NULL, never EOF or \n or anything else
- When using scanf/printf be careful not to mix %c and %s
  - %c is used to read into, or print, a single character variable
  - Can surely use with an element of a char array e.g. printf("%c", &str[i]);

## Keep in mind when using strings

- Use sufficiently big character arrays when reading input
  - Short arrays can cause segfaults or weird behaviour
- EOF is not a substitute for NULL
  - Do not store EOF in a char array and expect Mr C to treat it like NULL
  - Strings must be delimited using only NULL, never EOF or \n or anything else
- When using scanf/printf be careful not to mix %c and %s
  - %c is used to read into, or print, a single character variable
  - Can surely use with an element of a char array e.g. printf("%c", &str[i]);
  - %s is used to read into, or print, a character array

# Keep in mind when using strings

- Use sufficiently big character arrays when reading input
  - Short arrays can cause segfaults or weird behaviour
- EOF is not a substitute for NULL
  - Do not store EOF in a char array and expect Mr C to treat it like NULL
  - Strings must be delimited using only NULL, never EOF or \n or anything else
- When using scanf/printf be careful not to mix %c and %s
  - %c is used to read into, or print, a single character variable
  - Can surely use with an element of a char array e.g. printf("%c", &str[i]);
  - %s is used to read into, or print, a character array
  - Recall, no & symbol needed when using %s in scanf

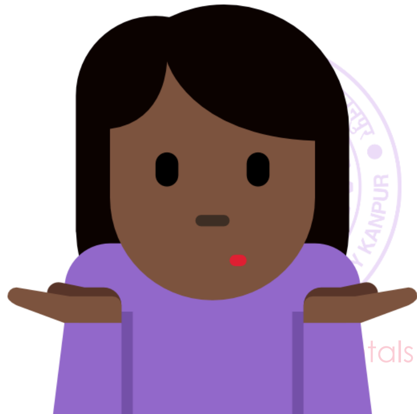
# Practice question

## Practice question

*Read at most 200 characters from input and print those characters in reverse. Input may include whitespaces*

# Practice question

*Read at most 200 characters from input and print those characters in reverse. Input may include whitespaces*



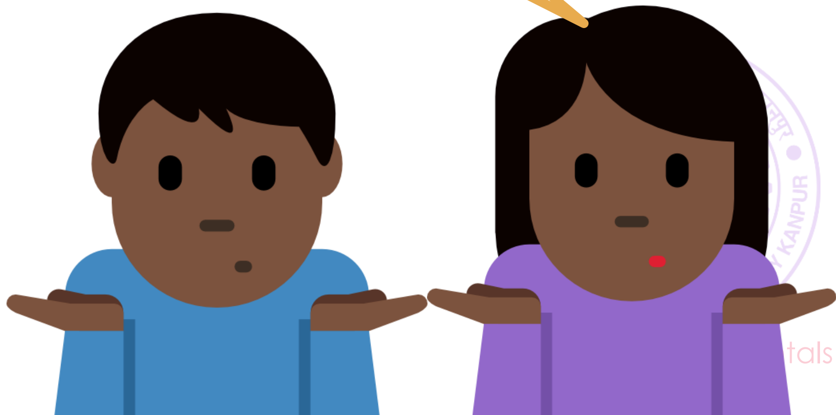
# Practice question

*Read at most 200 characters from input and print those characters in reverse. Input may include whitespaces*

Can't use scanf – will stop reading at spaces

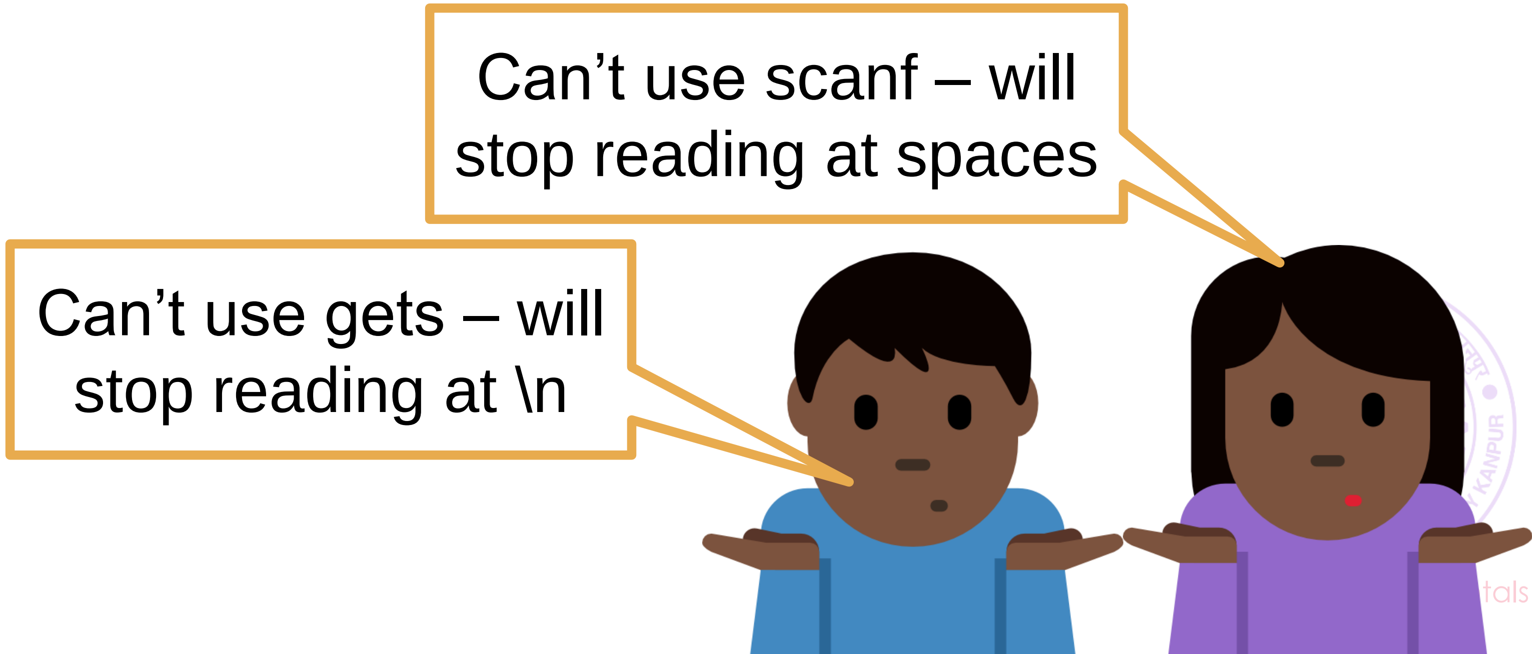
# Practice question

*Read at most 200 characters from input and print those characters in reverse. Input may include whitespaces*



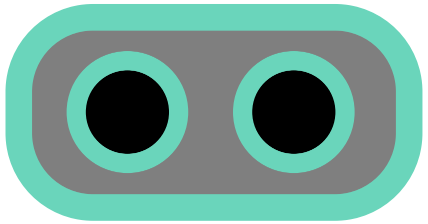
## Practice question

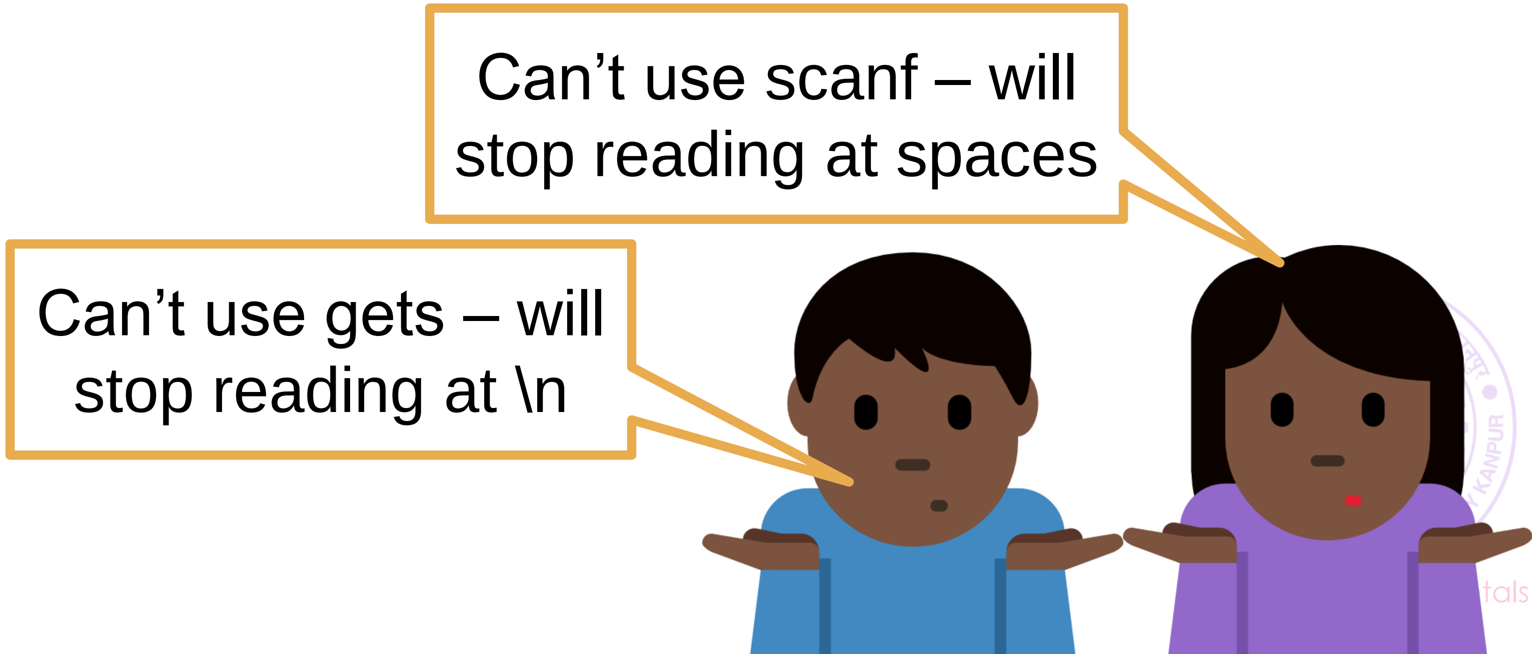
*Read at most 200 characters from input and print those characters in reverse. Input may include whitespaces*


Can't use scanf – will stop reading at spaces
Can't use gets – will stop reading at \n

# Practice question

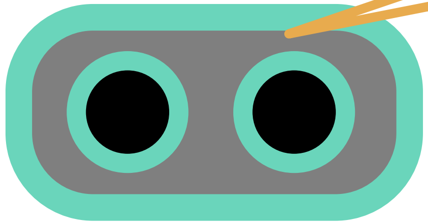
*Read at most 200 characters from input and print those characters in reverse. Input may include whitespaces*

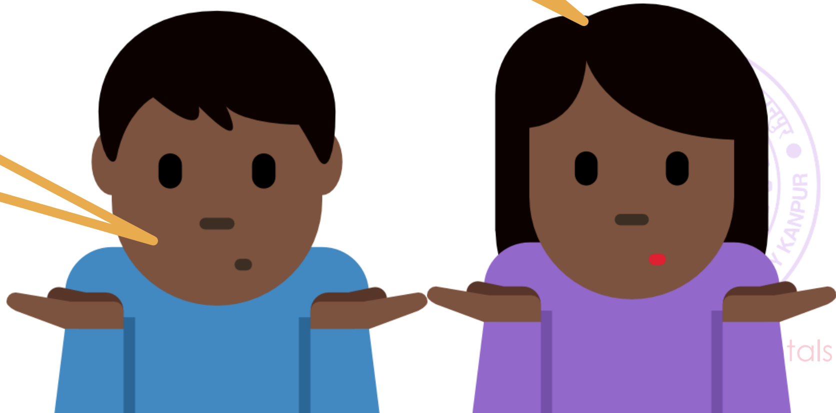



Can't use scanf – will stop reading at spaces
Can't use gets – will stop reading at \n

## Practice question

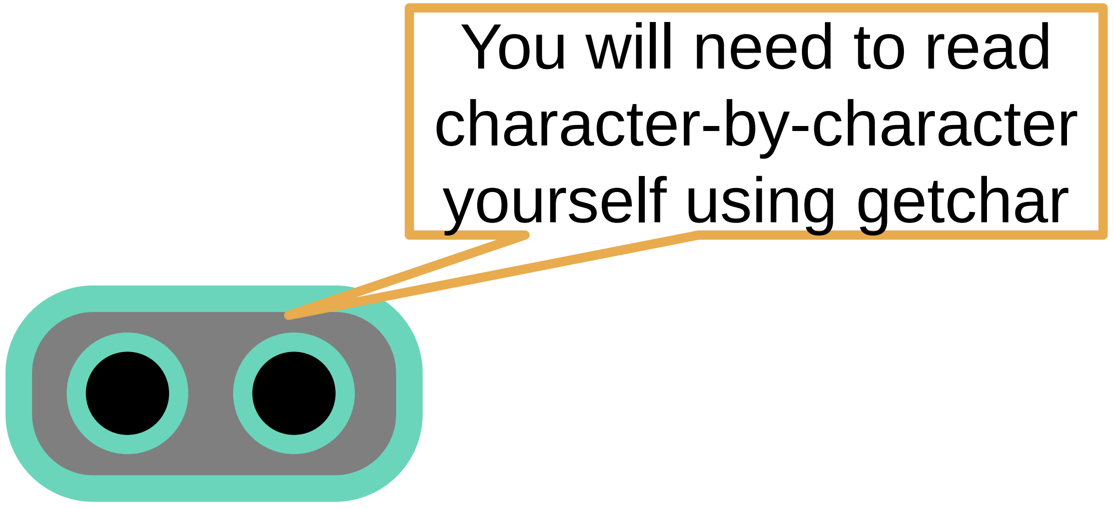
*Read at most 200 characters from input and print those characters in reverse. Input may include whitespaces*

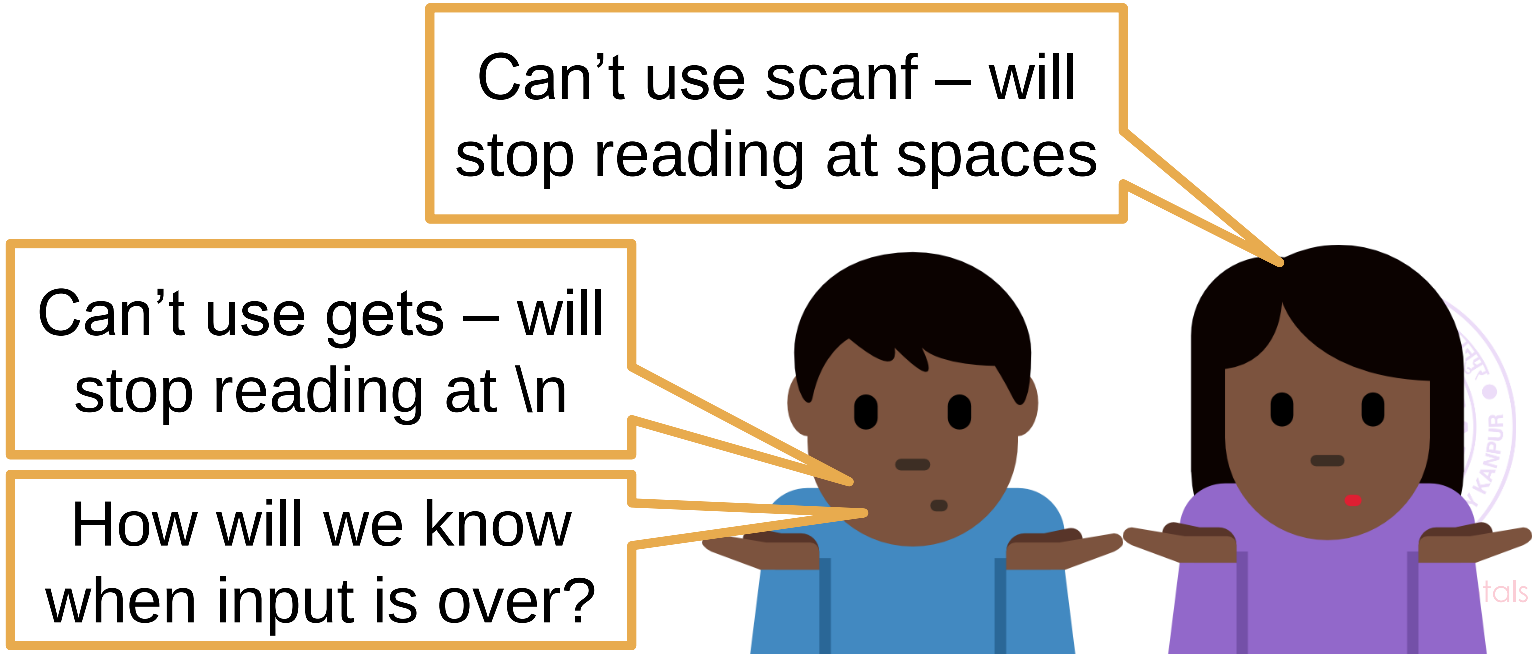

You will need to read character-by-character yourself using getchar


Can't use scanf – will stop reading at spaces
Can't use gets – will stop reading at \n

# Practice question

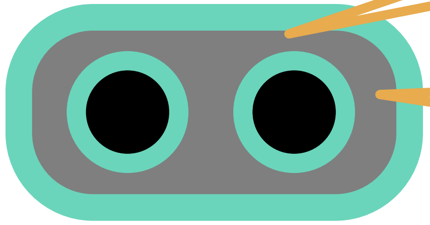
*Read at most 200 characters from input and print those characters in reverse. Input may include whitespaces*

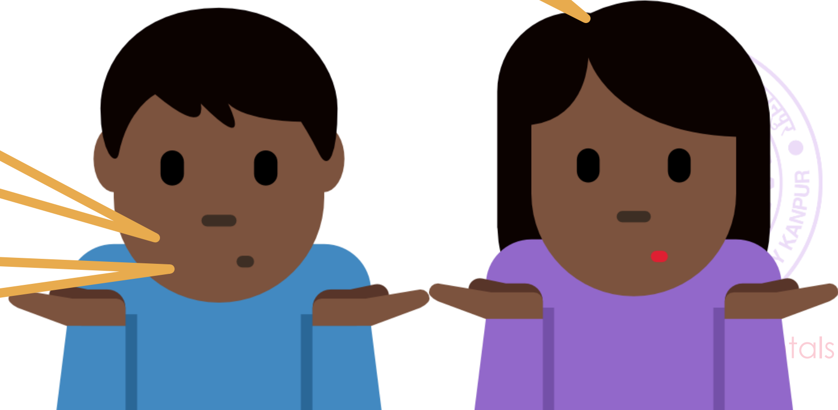

You will need to read character-by-character yourself using getchar


Can't use scanf – will stop reading at spaces
Can't use gets – will stop reading at \n
How will we know when input is over?

# Practice question

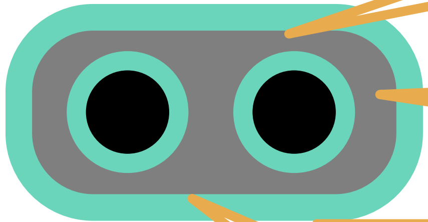
*Read at most 200 characters from input and print those characters in reverse. Input may include whitespaces*


You will need to read character-by-character yourself using getchar
I will alert you by sending you an EOF


Can't use scanf – will stop reading at spaces
Can't use gets – will stop reading at \n
How will we know when input is over?

# Practice question

*Read at most 200 characters from input and print those characters in reverse. Input may include whitespaces*



You will need to read character-by-character yourself using getchar

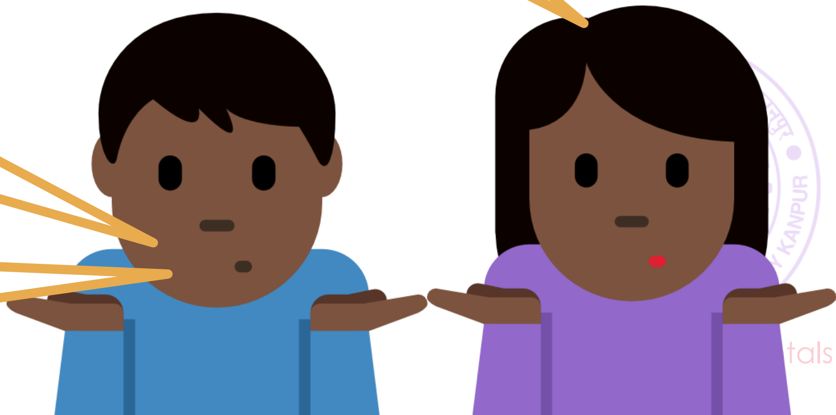
I will alert you by sending you an EOF

Be careful to store a NULL at end of string

Can't use scanf – will stop reading at spaces

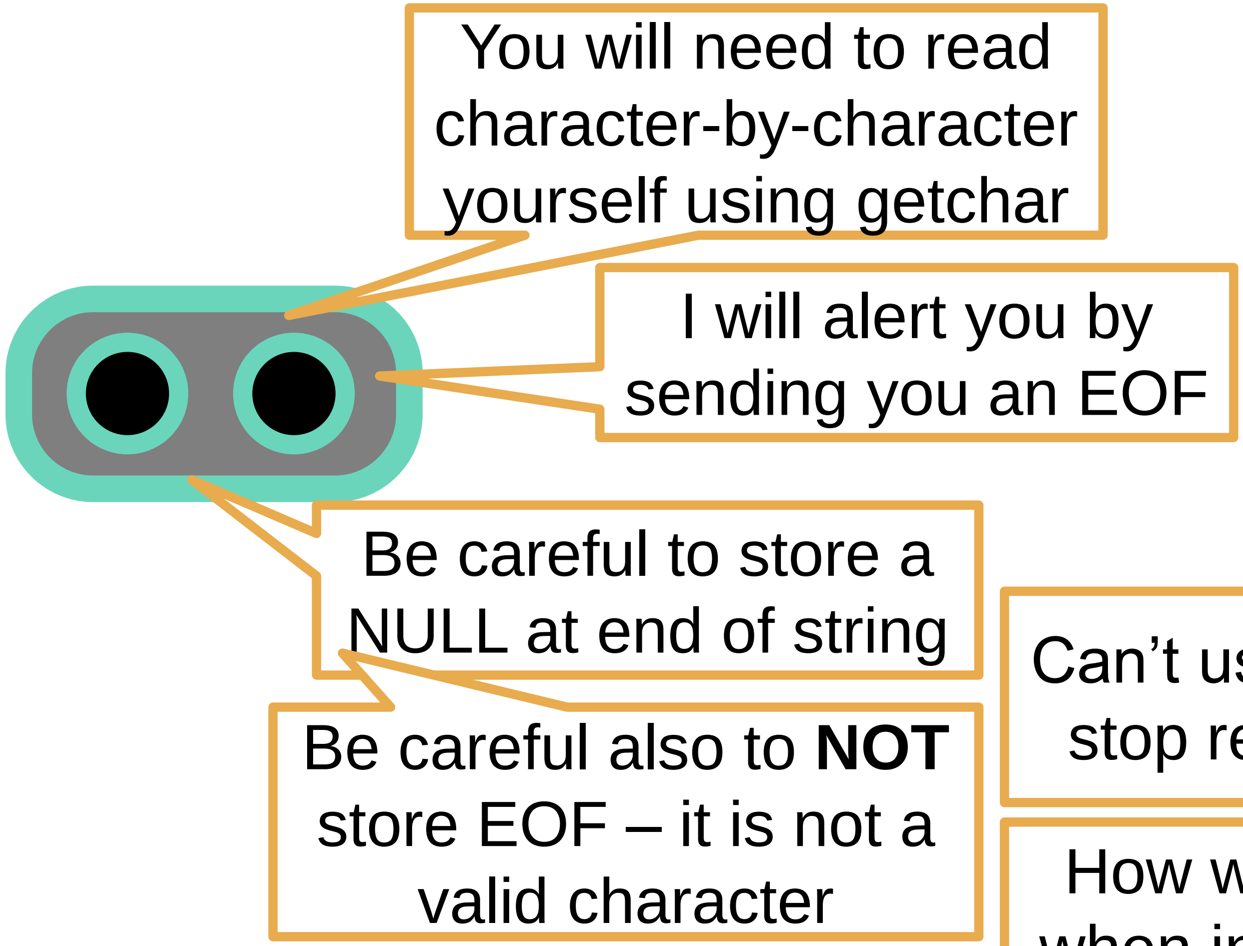
Can't use gets – will stop reading at \n

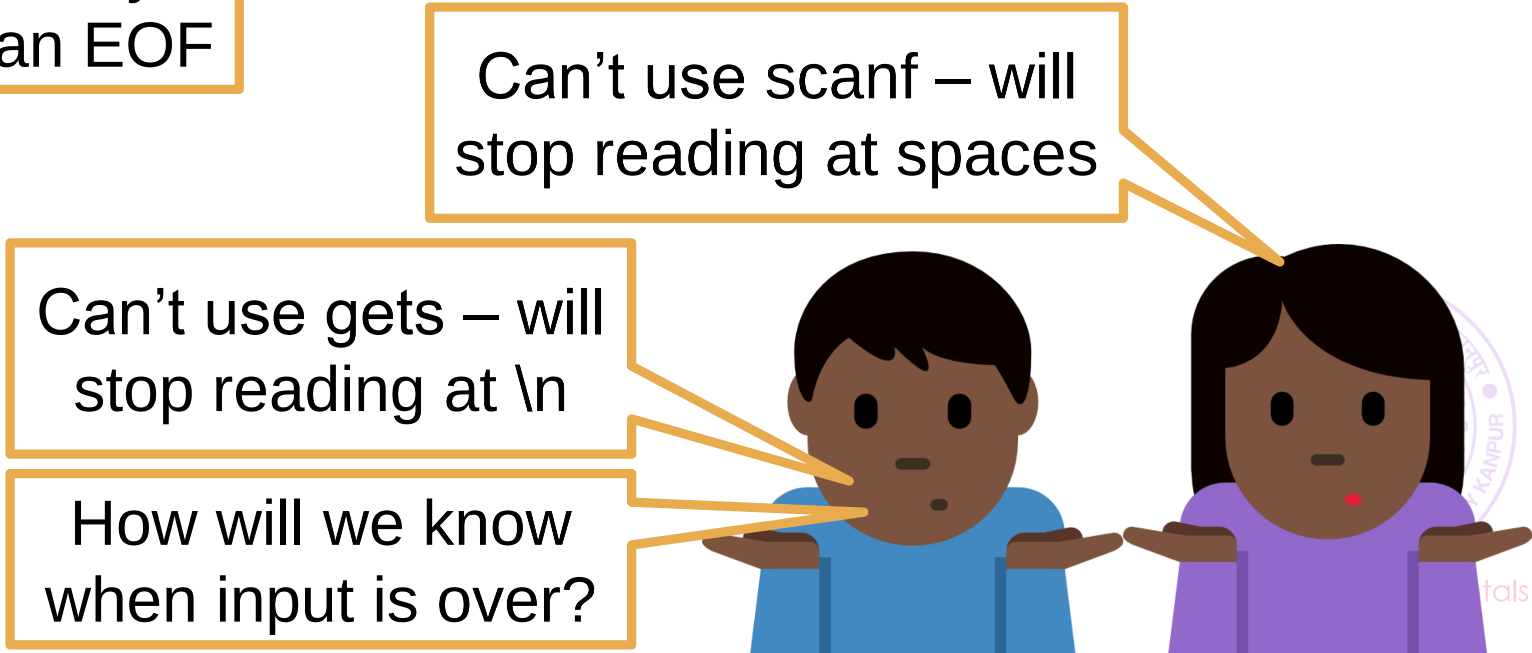
How will we know when input is over?



# Practice question

*Read at most 200 characters from input and print those characters in reverse. Input may include whitespaces*


You will need to read character-by-character yourself using getchar
I will alert you by sending you an EOF
Be careful to store a NULL at end of string
Be careful also to **NOT** store EOF – it is not a valid character


Can’t use scanf – will stop reading at spaces
Can’t use gets – will stop reading at \n
How will we know when input is over?

# String Operations

## String Operations

At their core, strings are just character arrays

## String Operations

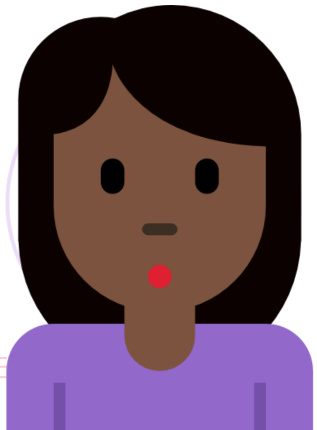
At their core, strings are just character arrays

Limitations of arrays apply to them as well

## String Operations

At their core, strings are just character arrays

Limitations of arrays apply to them as well



# String Operations

At their core, strings are just character arrays

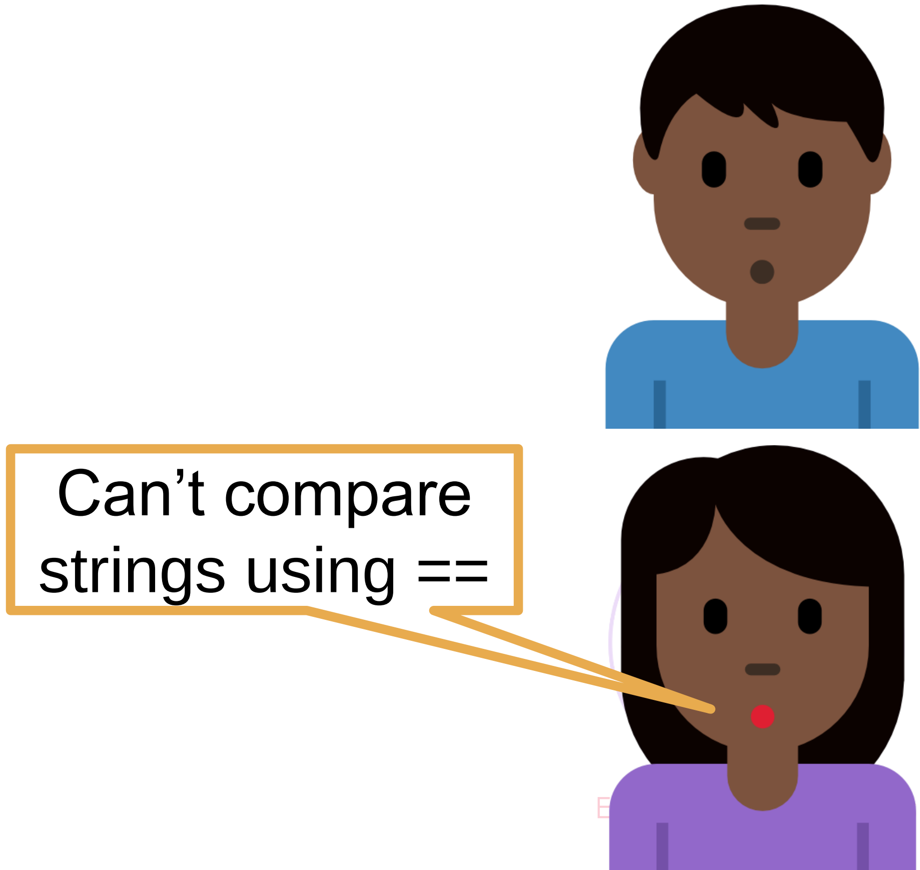
Limitations of arrays apply to them as well

Can't compare strings using ==

# String Operations

At their core, strings are just character arrays

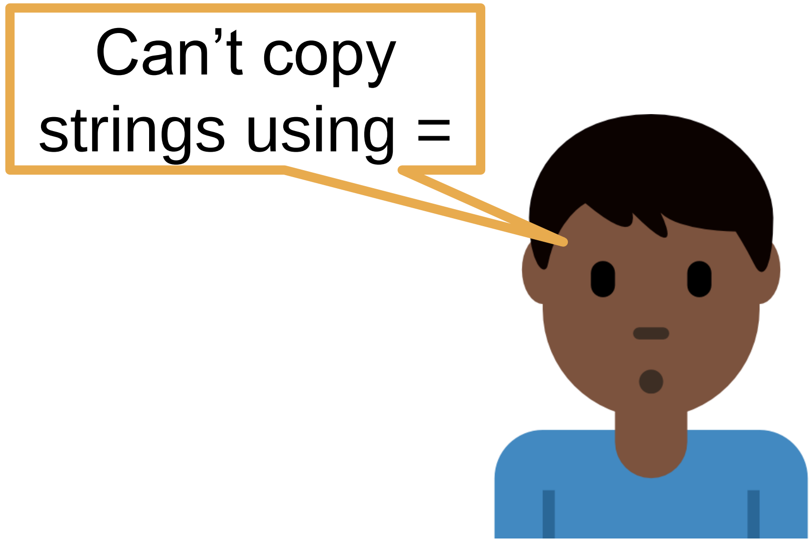
Limitations of arrays apply to them as well

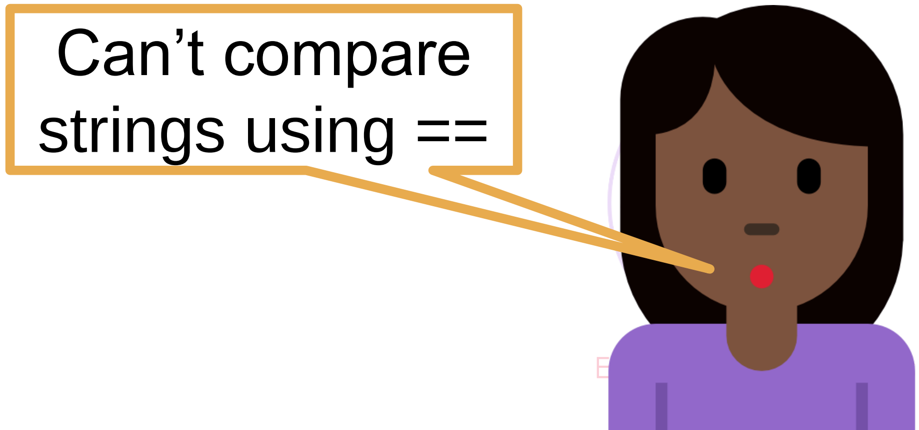


## String Operations

At their core, strings are just character arrays

Limitations of arrays apply to them as well

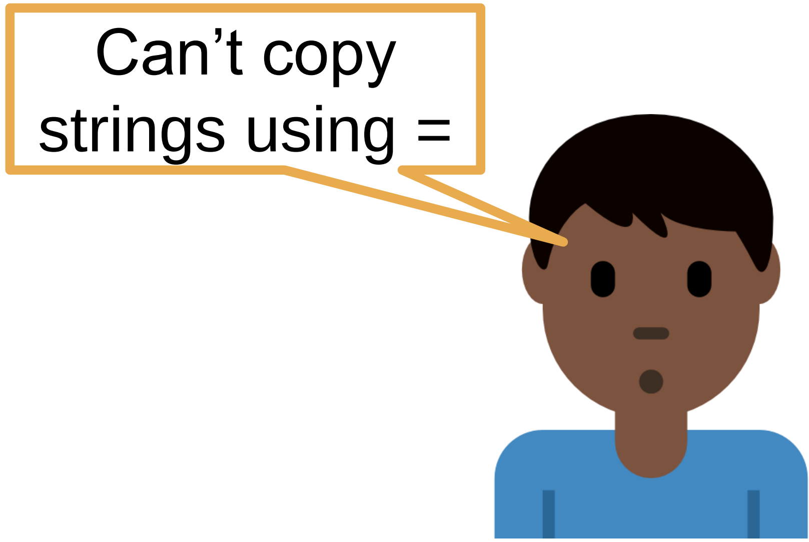


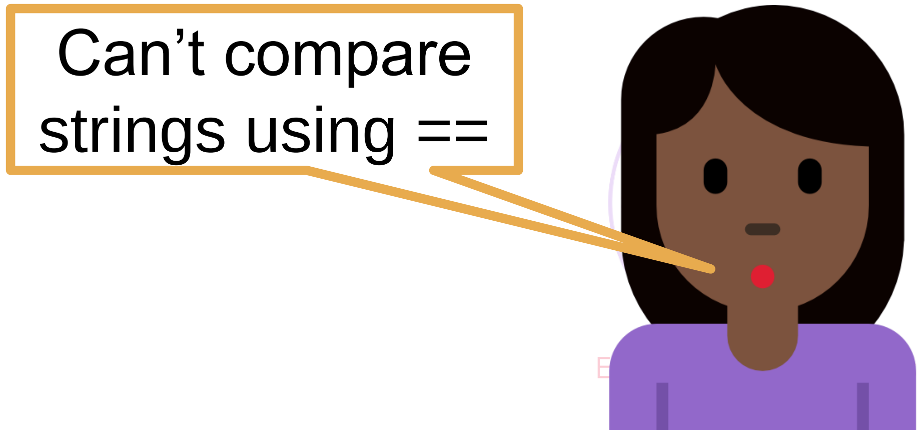


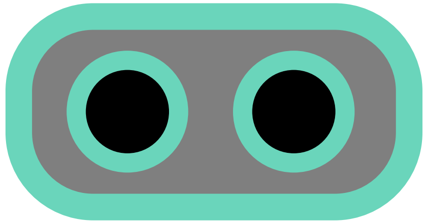
## String Operations

At their core, strings are just character arrays

Limitations of arrays apply to them as well



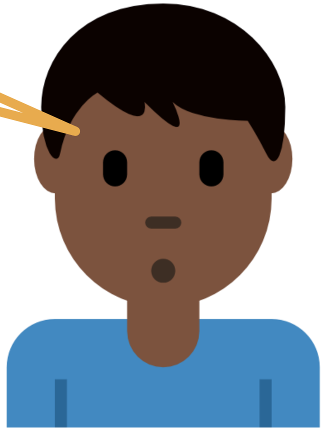


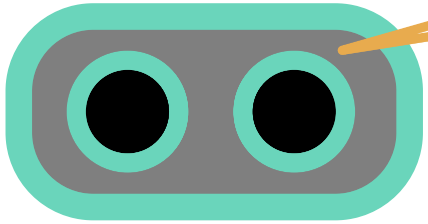


## String Operations

At their core, strings are just character arrays

Limitations of arrays apply to them as well

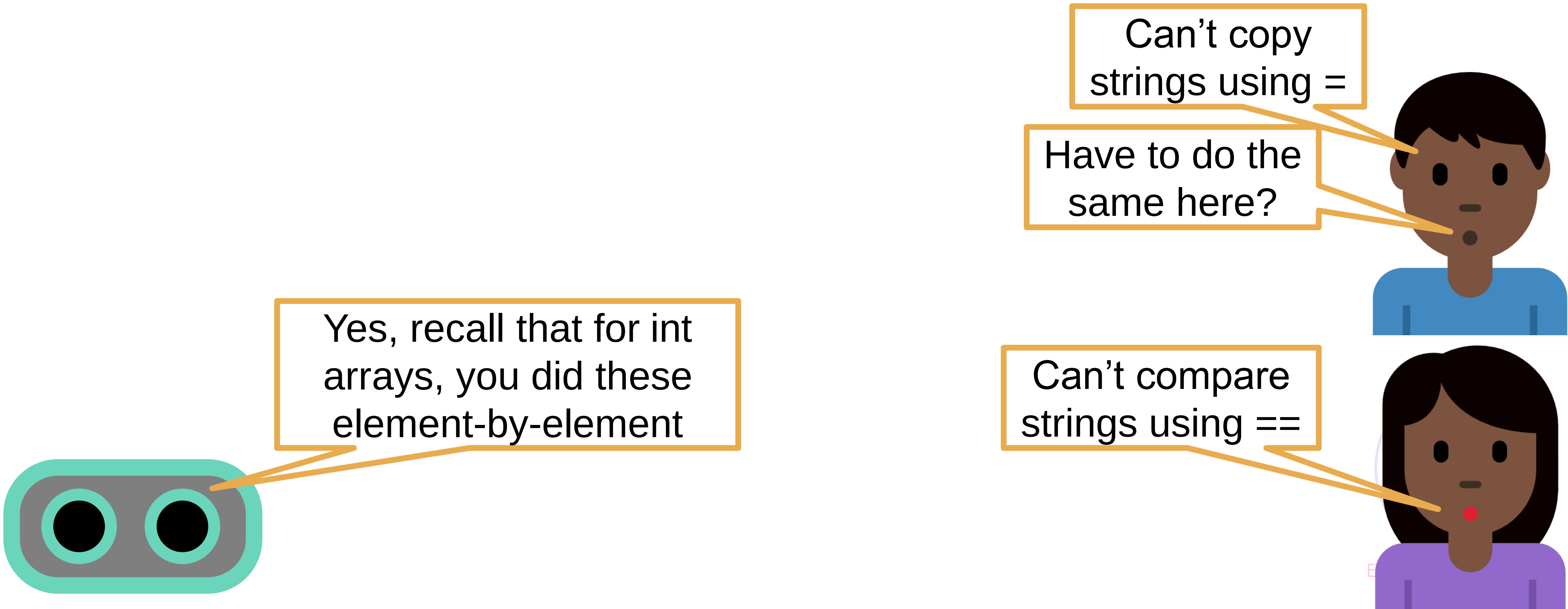



Yes, recall that for int arrays, you did these element-by-element

## String Operations

At their core, strings are just character arrays

Limitations of arrays apply to them as well


Can’t copy strings using =
Have to do the same here?
Can’t compare strings using ==
Yes, recall that for int arrays, you did these element-by-element

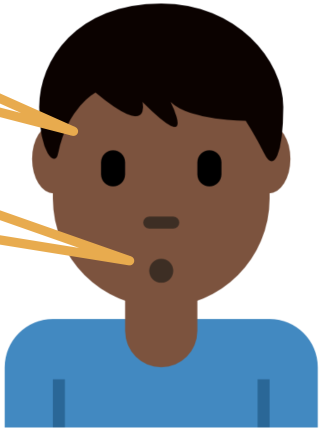
# String Operations

At their core, strings are just character arrays

Limitations of arrays apply to them as well

Can't copy strings using =

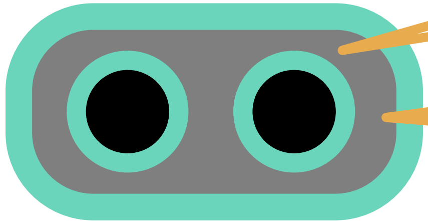
Have to do the same here?



Can't compare strings using ==

Yes, recall that for int arrays, you did these element-by-element

Fortunately, convenient library functions available for strings



# String Operations

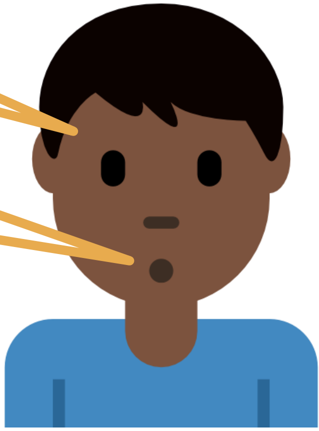
At their core, strings are just character arrays

Limitations of arrays apply to them as well

Available through string.h

Can't copy strings using =

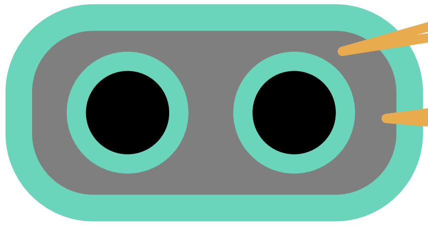
Have to do the same here?



Can't compare strings using ==

Yes, recall that for int arrays, you did these element-by-element

Fortunately, convenient library functions available for strings



## Useful string functions – string.h

## Useful string functions – string.h

strlen(str): returns length of the string str (upto NULL)

## Useful string functions – string.h

strlen(str): returns length of the string str (upto NULL)

- NULL character not counted in length

# Useful string functions – string.h

strlen(str): returns length of the string str (upto NULL)

- NULL character not counted in length

strcpy(dest, src): copies string src to string dest

# Useful string functions – string.h

- strlen(str): returns length of the string str (upto NULL)
  - NULL character not counted in length
- strcpy(dest, src): copies string src to string dest
  - Careful: characters after NULL character are not copied

## Useful string functions – string.h

strlen(str): returns length of the string str (upto NULL)

- NULL character not counted in length

strcpy(dest, src): copies string src to string dest

- Careful: characters after NULL character are not copied
- If you want them copied too, do so yourself! Mr C stops at NULL character

# Useful string functions – string.h

strlen(str): returns length of the string str (upto NULL)

- NULL character not counted in length

strcpy(dest, src): copies string src to string dest

- Careful: characters after NULL character are not copied
- If you want them copied too, do so yourself! Mr C stops at NULL character
- Make sure dest array has enough length (at least strlen(src) + 1 for NULL)

# Useful string functions – string.h

strlen(str): returns length of the string str (upto NULL)

- NULL character not counted in length

strcpy(dest, src): copies string src to string dest

- Careful: characters after NULL character are not copied
- If you want them copied too, do so yourself! Mr C stops at NULL character
- Make sure dest array has enough length (at least strlen(src) + 1 for NULL)

strcat(dest, src): appends src at the end of dest and moves NULL to the end of combined string

# Useful string functions – string.h

strlen(str): returns length of the string str (upto NULL)

- NULL character not counted in length

strcpy(dest, src): copies string src to string dest

- Careful: characters after NULL character are not copied
- If you want them copied too, do so yourself! Mr C stops at NULL character
- Make sure dest array has enough length (at least strlen(src) + 1 for NULL)

strcat(dest, src): appends src at the end of dest and moves NULL to the end of combined string

- Anything after NULL in dest will be lost – WARNING!

## Useful string functions – string.h

strlen(str): returns length of the string str (upto NULL)

- NULL character not counted in length

strcpy(dest, src): copies string src to string dest

- Careful: characters after NULL character are not copied
- If you want them copied too, do so yourself! Mr C stops at NULL character
- Make sure dest array has enough length (at least strlen(src) + 1 for NULL)

strcat(dest, src): appends src at the end of dest and moves NULL to the end of combined string

- Anything after NULL in dest will be lost – WARNING!
- Make sure dest array has enough length to absorb new characters

# Useful string functions – string.h

strlen(str): returns length of the string str (upto NULL)

- NULL character not counted in length

strcpy(dest, src): copies string src to string dest

- Careful: characters after NULL character are not copied
- If you want them copied too, do so yourself! Mr C stops at NULL character
- Make sure dest array has enough length (at least strlen(src) + 1 for NULL)

strcat(dest, src): appends src at the end of dest and moves NULL to the end of combined string

- Anything after NULL in dest will be lost – WARNING!
- Make sure dest array has enough length to absorb new characters

Note: strrev and some other functions non-standard

# Useful string functions – string.h

strlen(str): returns length of the string str (upto NULL)

- NULL character not counted in length

strcpy(dest, src): copies string src to string dest

- Careful: characters after NULL character are not copied
- If you want them copied too, do so yourself! Mr C stops at NULL character
- Make sure dest array has enough length (at least strlen(src) + 1 for NULL)

strcat(dest, src): appends src at the end of dest and moves NULL to the end of combined string

- Anything after NULL in dest will be lost – WARNING!
- Make sure dest array has enough length to absorb new characters

Note: strrev and some other functions non-standard

Already seen how to reverse a string in tutorial ☺

# Useful string functions – string.h

# Useful string functions – string.h

strcmp(str1, str2): returns 0 if strings equal (till the first NULL) else returns the difference in the ASCII values of first character that differs in the two strings.

## Useful string functions – string.h

strcmp(str1, str2): returns 0 if strings equal (till the first NULL) else returns the difference in the ASCII values of first character that differs in the two strings.

str**n**cpy(d, s, n), str**n**cat(d, s, n), str**n**cmp(d, s, n): does the operations but only for the first n characters

## Useful string functions – string.h

strcmp(str1, str2): returns 0 if strings equal (till the first NULL) else returns the difference in the ASCII values of first character that differs in the two strings.

str**n**cpy(d, s, n), str**n**cat(d, s, n), str**n**cmp(d, s, n): does the operations but only for the first n characters

str**case**cmp(d, s) compares strings in case-insensitive way

## Useful string functions – string.h

strcmp(str1, str2): returns 0 if strings equal (till the first NULL) else returns the difference in the ASCII values of first character that differs in the two strings.

str**n**cpy(d, s, n), str**n**cat(d, s, n), str**n**cmp(d, s, n): does the operations but only for the first n characters

str**case**cmp(d, s) compares strings in case-insensitive way

str**ncase**cmp(d, s, n) compares first n chars case-insensitive

## Useful string functions – string.h

strcmp(str1, str2): returns 0 if strings equal (till the first NULL) else returns the difference in the ASCII values of first character that differs in the two strings.

str**n**cpy(d, s, n), str**n**cat(d, s, n), str**n**cmp(d, s, n): does the operations but only for the first n characters

str**case**cmp(d, s) compares strings in case-insensitive way

str**ncase**cmp(d, s, n) compares first n chars case-insensitive

strupr(s)/ strlwr(s) convert to upper/lower case

# Useful string functions – string.h

strcmp(str1, str2): returns 0 if strings equal (till the first NULL) else returns the difference in the ASCII values of first character that differs in the two strings.

str**n**cpy(d, s, n), str**n**cat(d, s, n), str**n**cmp(d, s, n): does the operations but only for the first n characters

str**case**cmp(d, s) compares strings in case-insensitive way

str**ncase**cmp(d, s, n) compares first n chars case-insensitive

strupr(s)/ strlwr(s) convert to upper/lower case

strstr(d,s) search for substring s in d. Return "index" of first occurrence (actually returns something called a pointer)

# Useful string functions – string.h

strcmp(str1, str2): returns 0 if strings equal (till the first NULL) else returns the difference in the ASCII values of first character that differs in the two strings.

str**n**cpy(d, s, n), str**n**cat(d, s, n), str**n**cmp(d, s, n): does the operations but only for the first n characters

str**case**cmp(d, s) compares strings in case-insensitive way

str**ncase**cmp(d, s, n) compares first n chars case-insensitive

strupr(s)/ strlwr(s) convert to upper/lower case

strstr(d,s) search for substring s in d. Return “index” of first occurrence (actually returns something called a pointer)

strchr(d,c) search for char c in d. Return pointer to location

## Practice Problem

# Practice Problem

*In the first line, user specifies manner in which 3 integers would be input using a format string. In second line, user gives three integers in that format. In third line, user specifies the manner in which the three integers are to be printed. Read the 3 integers in the manner user wants and then print them in the manner the user wants.*

## Practice Problem

*In the first line, user specifies manner in which 3 integers would be input using a format string. In second line, user gives three integers in that format. In third line, user specifies the manner in which the three integers are to be printed. Read the 3 integers in the manner user wants and then print them in the manner the user wants.*

The format string we use in scanf, printf, is also a string

# Practice Problem

*In the first line, user specifies manner in which 3 integers would be input using a format string. In second line, user gives three integers in that format. In third line, user specifies the manner in which the three integers are to be printed. Read the 3 integers in the manner user wants and then print them in the manner the user wants.*

The format string we use in scanf, printf, is also a string

This means, we can take the format string from user too!