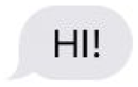
Question 1 of 3

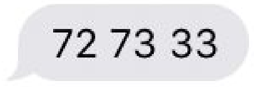
a) Suppose that you've just received the text message below.



HI!

Assuming the text message is represented using ASCII, what sequence of bits (or, if you prefer, decimal digits) did you actually receive?

b) Suppose that you've just received the below text message instead.



72 73 33

Assuming the text message is represented using ASCII, what sequence of bits (or, if you prefer, decimal digits) did you actually receive?

c) Why are those sequences not the same?

Answers

a) 72 73 33 (0)
b) 55 50 32 55 51 32 51 51 (0)
c) The first message is the ASCII value for the character 'H', the character 'I', then the character '!'. The second message is the ASCII value for the character '7', the character '2', the character ' ', etc. In the latter message, "72" is in the actual text of the message and so each character has an ASCII value, whereas in the former message the number 72 is the ASCII value for the first character.

Question 2 of 3

Recall that, in lecture, we saw how to write programs in C that support command-line arguments. To do so, we modified the program's `main` function to take two arguments: `argc` and `argv`.

a) What's a program you've used already in CS50 that accepts command-line arguments? Name the program and describe what the command-line arguments are used for.
b) What is stored in `argc`? What is its type?
c) What is stored in `argv`? What is its type?

Answers

a) Answers may vary. Possibilities include:
   i) `clang`, which accepts the name of the source code file to compile, as well as the name of the program to generate as command-line arguments.
   ii) `make`, which accepts the name of the program to generate as a command-line argument.
   iii) `help50`, which accepts the command that is producing an error as a command-line argument.
   iv) `style50`, which accepts the name of the file to check style for as a command-line argument.
   v) `check50` or `submit50`, which accepts a submission slug as a command-line argument.
b) `argc` stores the number of command line arguments provided to the program, and is an `int`.
c) `argv` stores the command line arguments themselves as an array of `string`s.

Question 3 of 3

Recall that, in lecture, we saw the following two `for` loops, both of which print the characters of a string, `s`, one character per line.

Version 1	Version 2

`for (int i = 0; i < strlen(s); i++)`
`{`
`    printf("%c\n", s[i]);`
`}`
	`for (int i = 0, n = strlen(s); i < n; i++)`
	`{`
	`    printf("%c\n", s[i]);`
	`}`

a) In your own words, what is a `string`?
b) In both of the above functions, the function `strlen` is used to get the length of the string `s`. Given what you know about how strings are represented in C, how does `strlen` likely compute the length of the string?
c) Which of these two versions of the code is more efficient? In what way is it more efficient?

Answers

a) A `string` is an array of characters.
b) `strlen` can look through the characters of the string one at a time, keeping track of how many characters it has seen, until it hits the `\0` (null terminator) character, which marks the end of the string. `strlen` can then output the number of characters it had to iterate over before it arrived at the null terminator.
c) Version 2 is more efficient: it computes the length of the string once and reuses that value later, rather than re-calculating the length of the string after every iteration of the loop.