Why do we use the enter key to terminate scanf?

Asked 1 year, 5 months ago Modified 1 year, 3 months ago Viewed 251 times

0

Lets say I have a simple code in c where I am asked to enter my age and then the program will print out my answer. When I run the program it will say "Enter your age: " and then (using scanf) wait for me to enter a number, say 40. I press the 4 and 0 keys on my keyboard. But then I have to press the enter key in order for the program to register what I have written and to terminate scanf. Why do I have to press enter? Why not the space key or even the shift key? I thought scanf("%d", &age); would consume and store all digits until it finds a non-digit, and then terminate. But if I write 40a or 4 0 instead of 40 I will still have to press enter for the scanf function to end and the let the program continue. Why? Is it just a definition, just as when you search something on google you have to press enter in the search-field? I hope you understand what I mean!

If I had to guess I would say it has something to do with a definition, but again, I don't know. I read another answer on a similar question but I didn't understand half of it.

c scanf

Share Edit Follow Flag

asked Apr 11, 2023 at 14:38
Alexander Jonsson
179 10

1 Because "Enter" means - "Enter the typed text as an input for the processing". – Eugene Sh. Apr 11, 2023 at 14:42

1 The `stdin` input stream is line-buffered by default. – Weather Vane Apr 11, 2023 at 14:44

1 It's not really related to `scanf`, its related to the stdin input stream, other input functions like `fgets` also needs enter. It acts as kind of trigger to signal that the input is complete and ready for processing. It's also not in the control of the program, it's the operating system that controls stdin buffering. – anastaciu Apr 11, 2023 at 14:45

Why not a space or shift-key to terminate? `scanf` might be inputting string data, or satisfying several format specifiers.. – Weather Vane Apr 11, 2023 at 14:47

Some terminals allow you to emit EOF character with ^D (Control + D). – jxh Apr 11, 2023 at 17:17

1 Answer

Sorted by: Highest score (default)

It's nothing to do with the program at all. When you type at a terminal, there are two modes in which it can operate: raw mode or cooked mode. In raw mode, the terminal driver will send

3

characters to stdin as they are typed. In cooked mode, it will wait until it sees a carriage return and then send the whole line. Cooked mode also does other things like interpret cursor keys and other special characters.

By default, your terminal is in cooked mode.

This is generally a good thing. If you type `40` for your number and then realise you made a mistake and should have typed `30`, in cooked mode you can hit backspace twice and type `30` and the program will just see `30` when you hit the return key. In raw mode, you have to handle backspace yourself, your program will see `40<backspace><backspace>30`.

On Unix-like operating systems, you use tcgetattr() and tcsetattr() to set raw mode on a terminal. Here is some code that does that. It's written in Swift but it should be fairly understandable to a C programmer.

```
// Creates a new empty termios structure
var originalTermios: termios = termios()

// Gets the current settings for the terminal. Note: fileHandle is a Unix
// file descriptor, not a FILE*   For stdin, it is conventionally 0
// It's assumed that you already know the file descriptor is a
// terminal.
if tcgetattr(fileHandle, &originalTermios) == -1 {
    throw LineNoise.Error.generalError("Could not get term attributes")
}

// defer is a Swift keyword that makes the following block run at
// the end of the scope. In this case it restores the original settings
defer {
    // Disable raw mode
    _ = tcsetattr(fileHandle, TCSAFLUSH, &originalTermios)
}
// Make a copy of the existing terminal settings
var raw = originalTermios
// Set/reset the necessary bits to go into raw mode.
// Other stuff happens here too, e.g. signal handling stuff.
#if os(Linux) || os(FreeBSD)
    raw.c_iflag &= ~UInt32(BRKINT | ICRNL | INPCK | ISTRIP | IXON)
            raw.c_oflag &= ~UInt32(OPOST)
            raw.c_cflag |= UInt32(CS8)
            raw.c_lflag &= ~UInt32(ECHO | ICANON | IEXTEN | ISIG)
#else
    raw.c_iflag &= ~UInt(BRKINT | ICRNL | INPCK | ISTRIP | IXON)
            raw.c_oflag &= ~UInt(OPOST)
            raw.c_cflag |= UInt(CS8)
            raw.c_lflag &= ~UInt(ECHO | ICANON | IEXTEN | ISIG)
#endif

raw.c_cc.16 = 1
// Set the attributes
if tcsetattr(fileHandle, Int32(TCSAFLUSH), &raw) < 0 {
    throw LineNoise.Error.generalError("Could not set raw mode")
}

// Here, do stuff with the file descriptor in raw mode.
// Note that using standard buffered IO functions might be unpredictable
```

Share Edit Follow Flag   edited Jun 7, 2023 at 9:17   answered Apr 11, 2023 at 14:47

JeremyP
86.3k 15 128 162

Some more info: en.wikipedia.org/wiki/Terminal_mode – Aykhan Hagverdili Apr 11, 2023 at 14:57

It is important to realize that the terminal is a character device and it chooses how to present its data to the program. The program is basically unaware of anything you do in the terminal until the terminal decides to let it know. By default many terminals will line-buffer the input, which means they will hide your input from the program until the line is terminated with a newline character. Thus, scanf is completely unaware of what's going on. – Aykhan Hagverdili Apr 11, 2023 at 15:04

Is the getch function an example of raw mode? – Alexander Jonsson Jun 6, 2023 at 16:34

@AlexanderJonsson That's a curses API function, isn't it? Maybe not, but curses probably has the ability to easily set raw mode. To set raw mode on Unix like OS's without curses, you need to use `tcsetattr()` and it's a bit of a mess. – JeremyP Jun 7, 2023 at 8:58