# Assignment 2: Input/Output

**TINBES02-2**

Bastiaan Teeuwen

Hogeschool Rotterdam

*April 16, 2019*

To make our operating system a little more interactive and user-friendly, we'll be writing a simple shell. A shell (like the name suggests) wraps around the operating system to provide users with a simple interface to manage programs, files, peripherals and the state of the system. User interfacing with the shell on PCs is usually achieved with the keyboard, mouse and monitor.

*Graphical user interfaces* (GUIs) like Microsoft Windows's shell (with the start menu, taskbar, explorer, etc.), X11 on Linux and macOS's Quartz are most popular nowadays due to their high ease of use.

*Command line interfaces* (CLIs) are still widely used on many servers and power users due to their low overhead and highly powerful features. GUIs contain a lot of unnecessary menus, windows, buttons which take up quite a bit of space on screen. They're more aesthetically pleasing but less expressive and powerful than a good old command line interface.

We will be writing a command line interface for our OS today. Fortunately for you, I have already provided the low-level interface drivers for the monitor and keyboard. All that's left for you to do is to write shell to wrap around them.

## 1 System library functions

Before you start coding, we'll cover a couple of functions that are part of the kernel that's already written for you.

### 1.1 Output

The output (monitor) part of the system library is available in these header files:

```
sys/print.h
sys/vga.h
```

Be sure to include these headers in your "`main.c`" file, which is where you'll program your CLI.

`sys/vga.h` is the low level interface for the VGA display (the standard output on the x86 architecture). Normally interfacing with the text terminal is done with ANSI escape sequences (see section 3) but for the sake of simplicity you can call the necessary functions directly. Call **`void vga_clear(void)`** to clear the screen.

VGA has a built-in cursor. The cursor is the location where text will be written to when `printf()` is called (after which it is automagically moved to the next column or line). The location of the cursor is represented with this structure:

```
struct vga_cursor {
        int x, y;
}
```

`x` cannot exceed `VGA_WIDTH` (80 columns) and `y` cannot exceed `VGA_HEIGHT` (25 lines), the driver ensures this. The cursor initially starts at `[0, 0]`and draws text from left to right. Text that exceeds the limits is automatically put on a new line and scrolling down also happens automatically.

Then there are two functions to set and get the cursor position.

```
struct vga_cursor vga_curget(void);
void vga_curset(struct vga_cursor cur, bool relative);
```

The `relative` boolean for `vga_curput()` controls whether the supplied cursor position relative (`true`) or absolute (`false`).

`sys/print.h` contains **`printf()`**. `printf()` is the standardized way to print text, number and such to the screen in C. I won't go into much detail in this document but you can look up the documentation for this function in your Linux or macOS manual pages with:

```
$ man 3 printf
```

or on this website for Windows users.

### 1.2 Input

The input (keyboard) part of the system library is available in this header file:

```
sys/ps2.h
```

PS/2 is the standard keyboard interface that is still used to this day. There's only one function you'll need: **`ps2_getch()`**. The function returns halts the system until a key is pressed after which a `char` will be returned. The *PS/2 driver* handles buffering internally so don't worry about missing a key when the system is not in a halted state waiting for input.

## 2 Escape characters

`ps2_getch()` returns a couple special characters when non-character keys are pressed. These special characters are actually part of the ASCII table. In C, they're represented with an escape character. An escape character is denoted with a backslash (\\). This is a list of escape character handled by the PS/2 driver:

| | | |
|---|---|---|
| **`\b`** | ⟵ | Backspace |
| **`\n`** | ↵ | Return |
| **`\t`** | ↹ | Tab |

If you've already read your manual pages you may note that these characters can also be used as input to `printf()`.

## 3 Escape sequences

Special characters that aren't available in the ASCII table are represented by an escape sequence. An ANSI escape sequence always starts with the (`\e`) escape character followed by a left-bracket (`[`). This is a list of escape sequences handled by the PS/2 driver:

| | | |
|---|---|---|
| **`\e[A`** | ↑ | Up arrow key |
| **`\e[B`** | ↓ | Down arrow key |
| **`\e[C`** | → | Right arrow key |
| **`\e[D`** | ← | Left arrow key |
| **`\e[F`** | End | End |
| **`\e[H`** | Home | Home |
| **`\e[P`** | Del. | Delete |
| **`\e[[`** | Esc | Escape |
| **`\e[*`** | Alt | + Character (for example Alt + P returns '`\e[p`') |

^* Ctrl + Character (for example Ctrl + Z returns '^Z')

^^ ^ Literal caret (same as Ctrl + ^)

ANSI escape sequences are unfortunately not handled by the VGA driver like they should be, you have my admiration if you implement this functionality.

You don't need to use all of these for your shell, but these might be handy to know when you start to work on the final assignment.

## 4 Command Line Interface

I won't give away a lot of information here as this part of the assignment is basic principle is that the user should be able to enter commands (with at least know: not that hard if you now understand how the VGA driver and the PS/2 work. The one argument separated by a whitespace character like a space) and run then when pressing enter. After the command is executed, the command line should prompt the user for a new command. There's a couple more important things you need to

Sending the backspace escape character to `printf()` automatically moves the cursor back one character and blanks the character under the cursor. Sending a newline automatically moves the cursor to the start of the next line. Just be sure to keep your command line buffer up to date and to keep track of the end of the buffer.

Also handle the left and right arrow keys correctly to move around the command line to insert and delete text (with ⟵ (Backspace) and Del.) in the middle of a line for example. You will also use this knowledge in your final assignment to move around. This task may be pretty difficult if you've never worked with C before. If you cannot get the arrow keys to work, ask for help or move on to the next section.

### 4.1 Commands

Let's implement some commands for our shell! Implement the following commands:

**halt** Pretty simple, simply return from `main()` and the system will halt itself.

**clear** Clear the display.

**echo** Echo the text typed after the command to the screen.

Use **`strtok()`** from `memory.h` to split commands up into a command and an argument. You'll need this for `echo`. We'll be adding more commands to our CLI in the next assignment (File Systems). Look this function up in the manual page if you don't know how it works on this website or on the command line:

```
$ man 3 strtok
```

Also print a message to the screen when an invalid command has been entered.

## 5 Bonus points

Here's some idea's:

- Handle multi-line commands correctly.
- Implement a buffer or history file to keep track of previously used commands. Navigate through the history with the up and down arrow keys.
- Implement escape sequences in the VGA driver to move the cursor around and clear the screen.