CS29206 Systems Programming Laboratory
Spring 2024
Lab Assignment: 7
Date: 04–Apr–2024

Shell programming using bash

Part 1: Continued fractions

$$a_0 + \cfrac{1}{a_1 + \cfrac{1}{a_2 + \ddots + \cfrac{1}{a_{n-1} + \cfrac{1}{a_n}}}}$$

A (finite simple) continued fraction is an expression of the form shown to the right. Here, a_0 is assumed to be a non-negative integer, whereas $a_1, a_2, \ldots, a_n$ are positive integers. We store this continued fraction as the array $(a_0, a_1, a_2, \ldots, a_n)$. Evidently, the value of this continued fraction is a (non-negative) rational number. For example, (2, 5, 1, 3) evaluates to

$$2+\cfrac{1}{5+\cfrac{1}{1+\cfrac{1}{3}}}=2+\cfrac{1}{5+\cfrac{1}{\frac{4}{3}}}=2+\cfrac{1}{5+\frac{3}{4}}=2+\cfrac{1}{\frac{23}{4}}=2+\frac{4}{23}=\frac{50}{23}\approx 2.1904761904.$$

Conversely, we can develop a continued-fraction expansion of a non-negative rational number a / b. In order to see how, let us first make the Euclidean division of a by b.

$$a=qb+r$$

Here, q is the quotient of division, and r is the remainder. If $r = 0$, we have $a / b = q$. On the other hand, if $r \neq 0$, then

$$\frac{a}{b}=q+\frac{r}{b}=q+\cfrac{1}{\frac{b}{r}}$$

Next, we divide b by r, and repeat. This gives us a procedure similar to the Euclidean gcd algorithm. For example, 225 / 80 can be expanded to a continued fraction as follows.

$225 = 2 \times 80 + 65$
$80 = 1 \times 65 + 15$
$65 = 4 \times 15 + 5$
$15 = 3 \times 5$

Therefore

$$\frac{225}{80}=2+\cfrac{1}{\frac{80}{65}}=2+\cfrac{1}{1+\cfrac{1}{\frac{65}{15}}}=2+\cfrac{1}{1+\cfrac{1}{4+\cfrac{1}{\frac{15}{5}}}}=2+\cfrac{1}{1+\cfrac{1}{4+\cfrac{1}{3}}}$$

Write an executable bash script *cfrac.sh* that does the following:

- Define a function *evalcfrac*() that reads (from the user) an array *A* standing for a continued fraction. It then evaluates and prints the continued fraction as a rational number of the form *a* / *b*. It finally prints the floating-point value of this fraction.

- Define a function *gencfrac*() that reads (from the user) a non-negative fraction *a* / *b*, and prints the continued-fraction expansion of this rational number as an array.

- Call *evalcfrac* and *gencfrac*.

Sample output

```
$ /cfrac.sh

Enter the array of coefficients: 3 7 15 1 292 1 1 1 2 1 3 1 14 2 1
The continued fraction evaluates to 245850922 / 78256779 = 3.1415926535

Enter fraction (a / b): 2468013579 / 1357924680
The continued fraction expansion of 2468013579 / 1357924680 is: 1 1 4 2 11 2 11 1 1 1 5 4 15 3 1 10
$
```

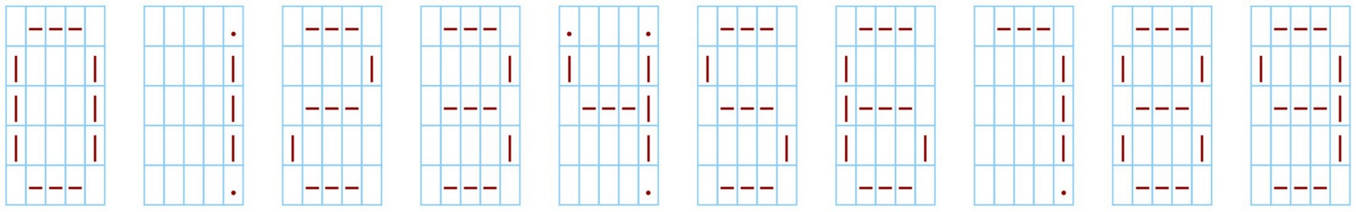
Note: In each of the two functions, read the user input in a single line (as demonstrated in the sample).

Part 2: A running clock

Write an executable bash script *clock.sh* that keeps on printing the current time at the center of the terminal in an ASCII format (not as a string), and updates the display at regular intervals (like every second or every half a second). The current time (and date) can be obtained by the Unix command `date`. Different machines may output the date and time in different formats. In this assignment, you run

```
date +"%A %d %B %Y %I:%M:%S %p %Z"
```

to get the format we need. You may read the man-page of `date` to know more about the command. Each digit in the current time is to printed as a 5 × 5 pattern illustrated below.



Also print a blinking : (consisting of two lower-case o's) between the hour and the minute and between the minute and the second. The program is not to be terminated by pressing control-c. Instead, whenever the user hits return, the program will terminate (and the shell prompt will return). If you resize the terminal window, the clock must also be repositioned to the center of the new window.

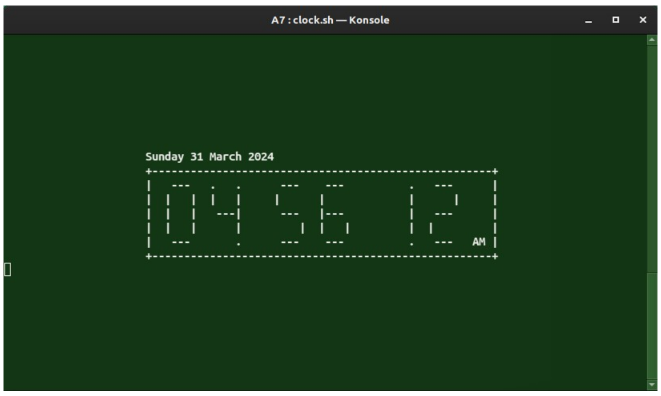
Implement the following algorithm:

```
Repeat forever:
    • Get the time and the date by calling date.
    • Separate the different fields from the output of date.
    • Clear the screen.
    • Print the date (as a string) and the time (as a text pattern) at the center of the window.
    • Wait for "some" time for the user to hit return.
```

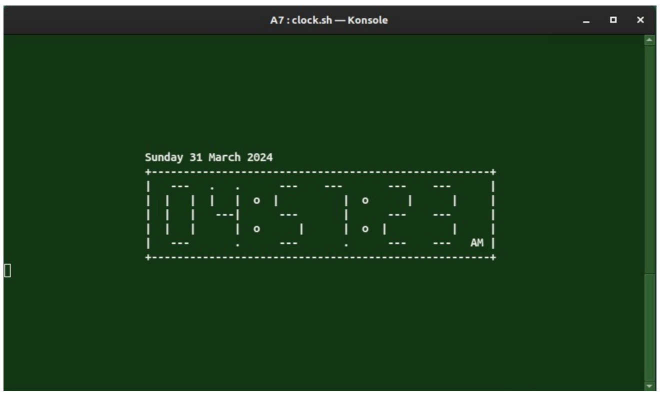
Submit the two files *cfrac.sh* and *clock.sh*.

Sample Output

We will provide you with a demo of the animation. Here, we are attaching a few screenshots.

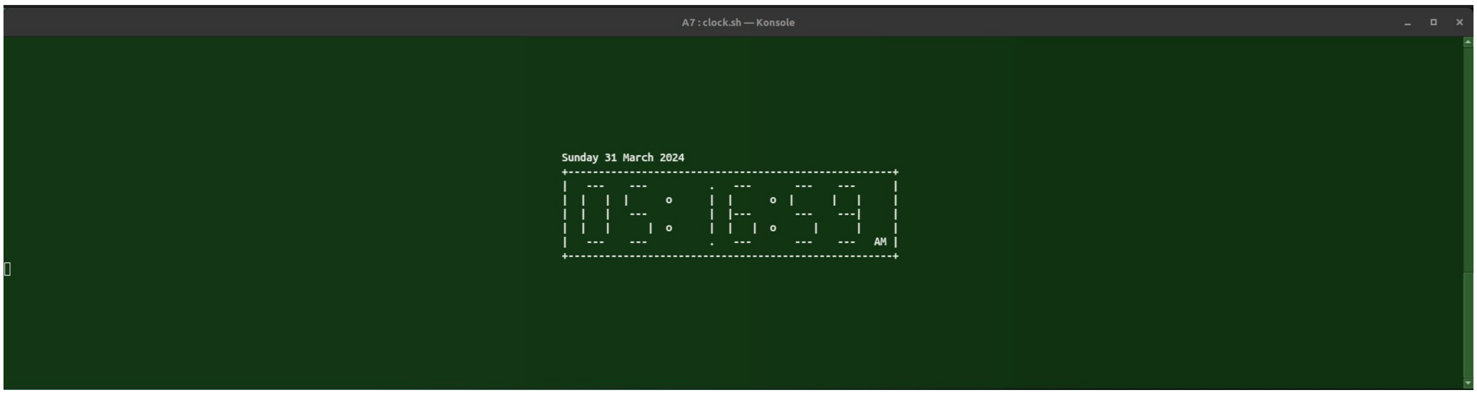


```
Sunday 31 March 2024
+----------------------------------------------------+
|   ---   .   .      ---    ---          .   ---     |
|  |   |  |   |     |      |             |      |    |
|  |   |   ---|      ---   |---          |   ---     |
|  |   |      |         |  |   |         |  |        |
|   ---       .      ---    ---          .   ---  AM |
+----------------------------------------------------+
```

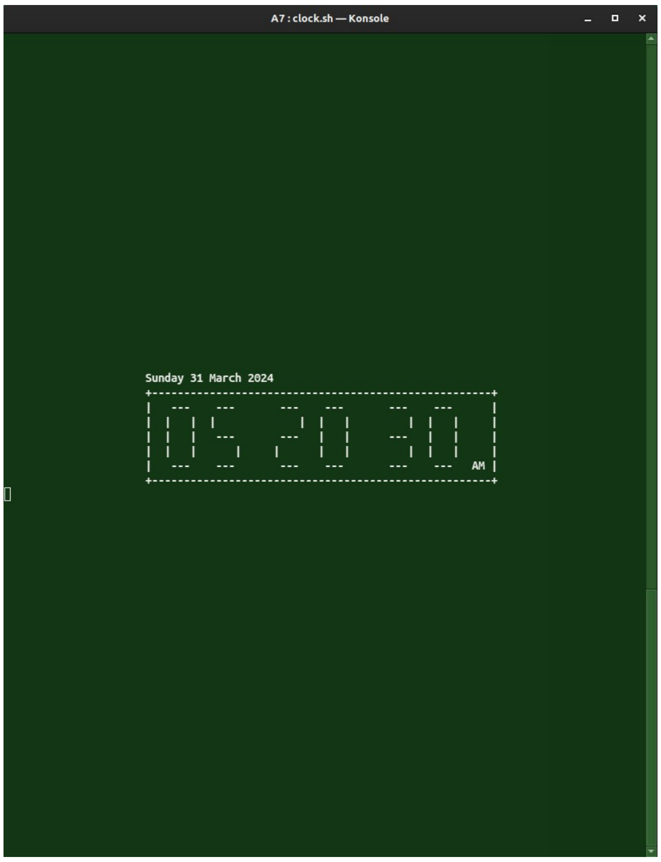


```
Sunday 31 March 2024
+----------------------------------------------------+
|   ---   .   .      ---    ---       ---    ---     |
|  |   |  |   |  o  |          |  o      |      |    |
|  |   |   ---|      ---       |      ---    ---     |
|  |   |      |  o      |      |  o  |          |    |
|   ---       .      ---       .      ---    ---  AM |
+----------------------------------------------------+
```

This is what should happen if you resize the window.



```
Sunday 31 March 2024
+----------------------------------------------------+
|   ---    ---          .   ---       ---    ---     |
|  |   |  |      o      |  |      o  |      |   |    |
|  |   |   ---          |  |---       ---    ---|    |
|  |   |      |  o      |  |   |  o      |      |    |
|   ---    ---          .   ---       ---    ---  AM |
+----------------------------------------------------+
```



```
Sunday 31 March 2024
+----------------------------------------------------+
|   ---    ---       ---    ---       ---    ---     |
|  |   |  |             |  |   |         |  |   |    |
|  |   |   ---       ---   |   |      ---   |   |    |
|  |   |      |     |      |   |         |  |   |    |
|   ---    ---       ---    ---       ---    ---  AM |
+----------------------------------------------------+
```

Hint: You can obtain usage instructions of bash builtin commands as: `help -m command`