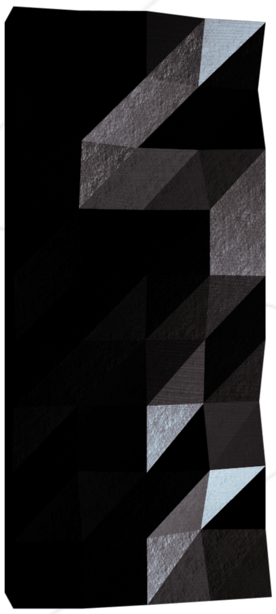


YASL

Yet Another Stack Language

Summary: YASL is an easy multiple-reversed-stacked language.

Version: 1

Contents

Chapter I

Forewords

Just in case you used to have, a very very long time ago, a calculator `HP 48GX`, you will be definitely in a familiar environment.

Chapter II

Appetizers

The goal of this project YASL is:

- Use an exotik coding language that you may not see again after this rush
- Solve a fun problem
- Make you confident in your adaptation capacity
- Make you think and code in a different way that you have done so far (in C, C++, ..)

YASL is a rather old coding language functionning with multiple reverted stacks. Details of this concept are described in the provided *man* with the yasl interpreter. You will need to rethink your way of storing and manipulating data.

As starters, you will face several small and academic exercises to get accointed with this language. A small part of the final mark but necessary for a successful completion of the following parts.
Then comes the main course: display of a base-64-encoded image in your terminal.
Just in case you're still hungry, you'll find some dessert at the end.

Chapter III

Starters

For these very first exercises, you need to handle carefully all possible errors, and avoid falling back to the error messages from the yasl interpreter (that directly stops your yasl program). The name of the script must be respected.

Exercise 0 : yasl_hw
Display the classic welcome message. Example from the *man* is probably a good start.

```
$> ./yasl_hw
Hello world
$>
```

Exercise 1 : yasl_aff_param
Display the command line parameters, one per line, in the order of the command line.

```
$> ./yasl_aff_param Hello World 42 "Be Cool"
Hello
World
42
Be Cool
$>
```

```
$> ./yasl_aff_param
$>
```

Exercise 2 : yasl_do <value1> <operator> <value2>
Makes the calculus or comparison between values. Operator can be:
+ - * / % < > <= >= == !=
No error management required on this exercise, all tests will provide a valid command line.

```
$> ./yasl_do 21 + 42
63
$>
```

```
$> ./yasl_do 84 ">" 24
1
$>
```

Exercise 3 : yasl_repeat <start_num> <value1> [<value2> [...]]
Display each parameter N times followed by a new line. N start with <start_num> and then increases by one for each subsequent parameter. Print an error message in case of wrong parameters. <start_num> must be strictly positive.

```
$> ./yasl_repeat 4 Bonjour "how are you ?"
BonjourBonjourBonjourBonjour
how are you ?how are you ?how are you ?how are you ?how are you ?
$>
```

```
$> ./yasl_repeat 1 "************" "******" "****" "---"
************
************
************
------------
$>
```

Exercise 4 : yasl_fact <num>
Classically computes the factorial of the number. Take the opportunity to do some recursivity, and check when you reach the overflow. You can face various incorrect situations with the parameter. You need to handle this correctly. Remember that 0! exists.

```
$> ./yasl_fact 10
3628800
$>
```

Exercise 5 : yasl_split <separator> <string>
Splits the provided string using the separator (a unique char), and display each part on a single line.
Correctly handle wrong separator.
If you can find the two ways of solving this exercise, you are reaching yasl mastery.

```
$> ./yasl_split a ljksdpoiapoipoialljlaiuoiu
ljksdpoi
poipoi
lljl
iuoiu
$>
```

```
$> ./yasl_split " " " Lorem ipsum dolor  sit amet, consectetur adipiscing elit. "
Lorem
ipsum
dolor
```

```
sit
amet,
consectetur
adipiscing
elit.
$>
```

Exercise 6 : yasl_interactive

Even if yasl interpretor already provides this feature, you need to create a script that reads on the standard input and then executes the yasl instructions you just received.

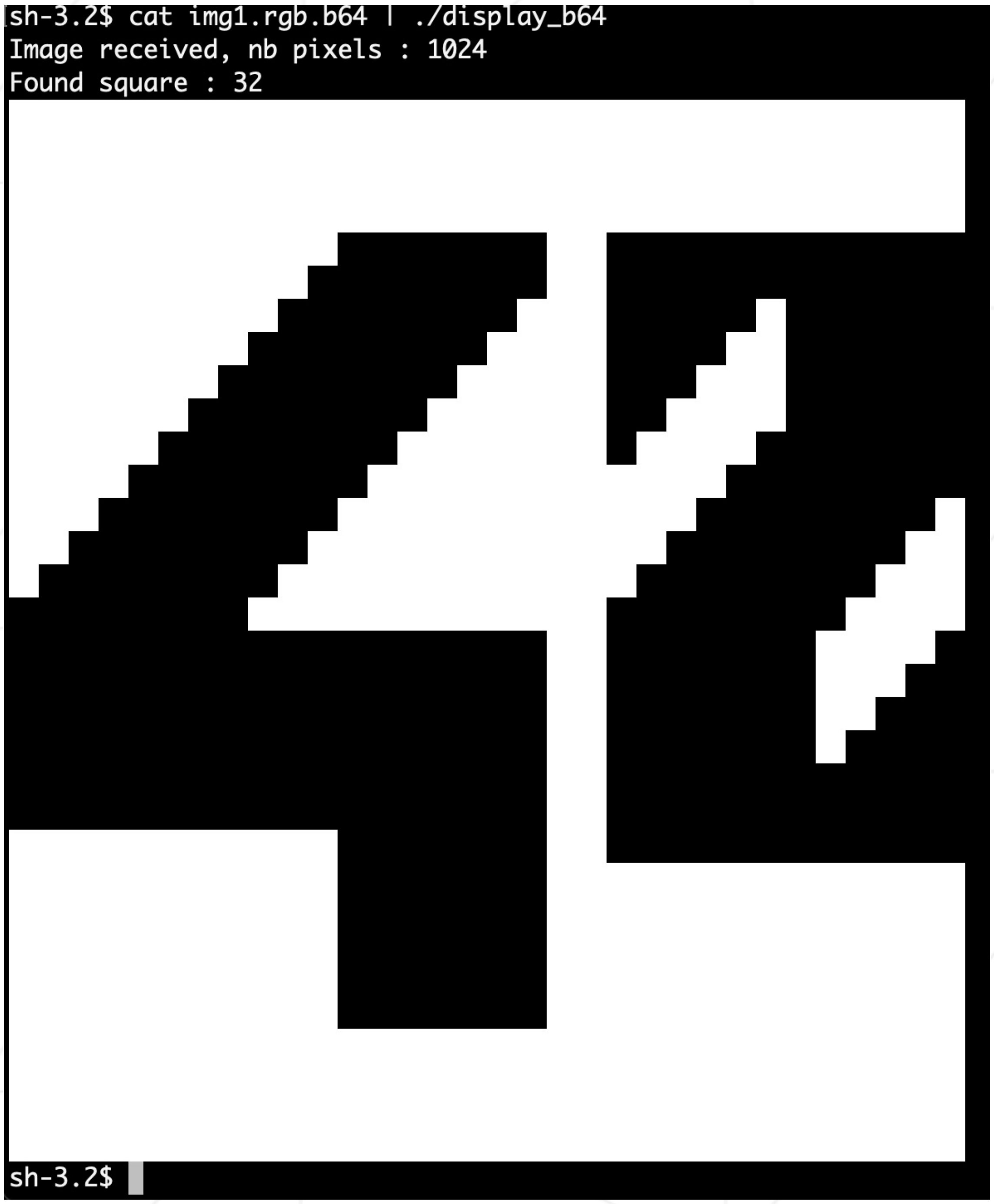
```
$> ./yasl_interactive
42
21
+ "\n" +
print
63
^D
$>
```

Chapter IV

Main course

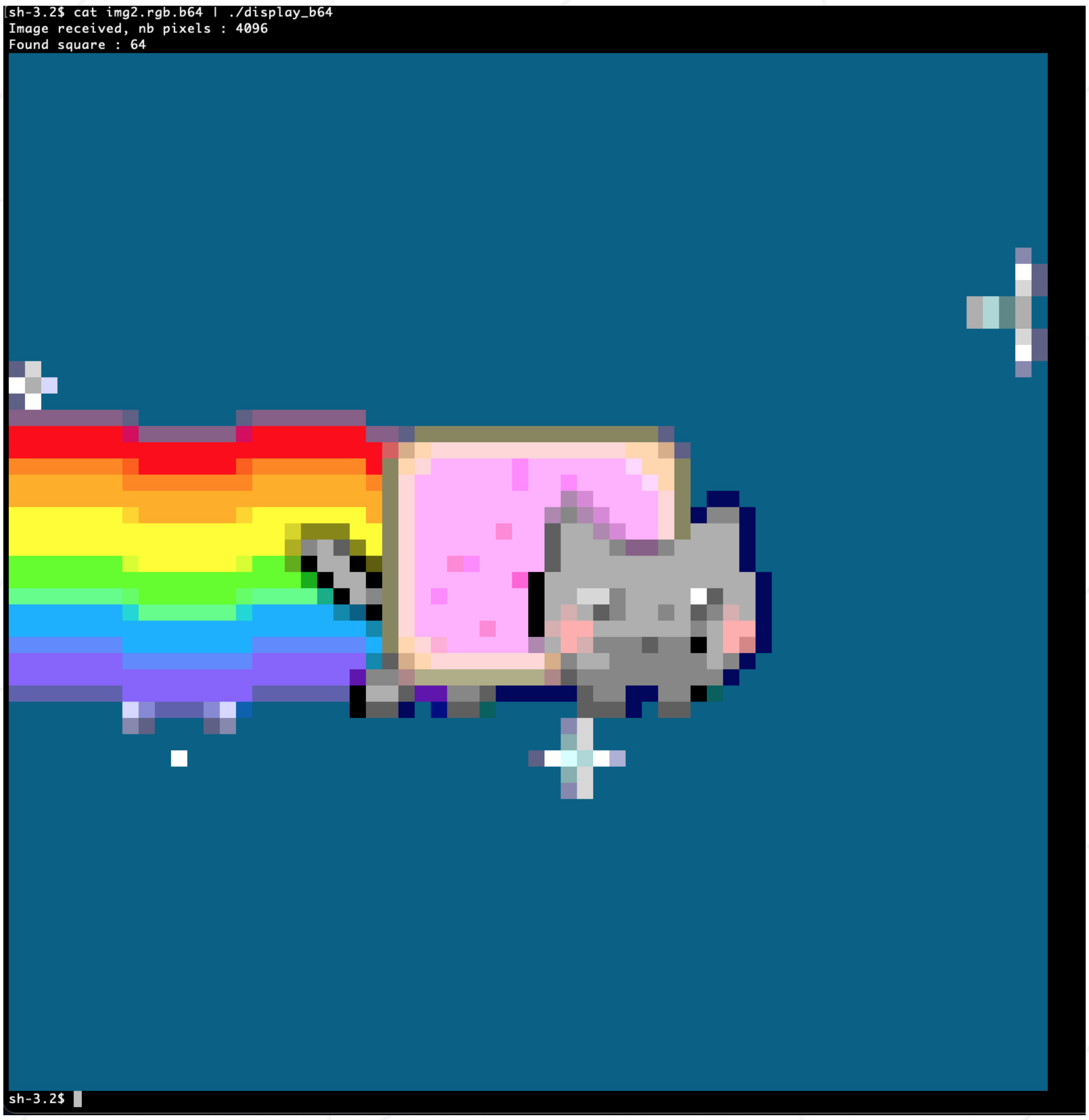
Create a yasl script that first reads from standard input a string. This string represents a base 64 encoded, 24 bits RGB, square, image. Then your script has to display the image on the terminal, using the standard 256 colors available on most implementations.

The script will be named `display_b64`.



```
sh-3.2$ cat img1.rgb.b64 | ./display_b64
Image received, nb pixels : 1024
Found square : 32
```

```
sh-3.2$
```



```
sh-3.2$ cat img2.rgb.b64 | ./display_b64
Image received, nb pixels : 4096
Found square : 64
```

```
sh-3.2$
```

Chapter V

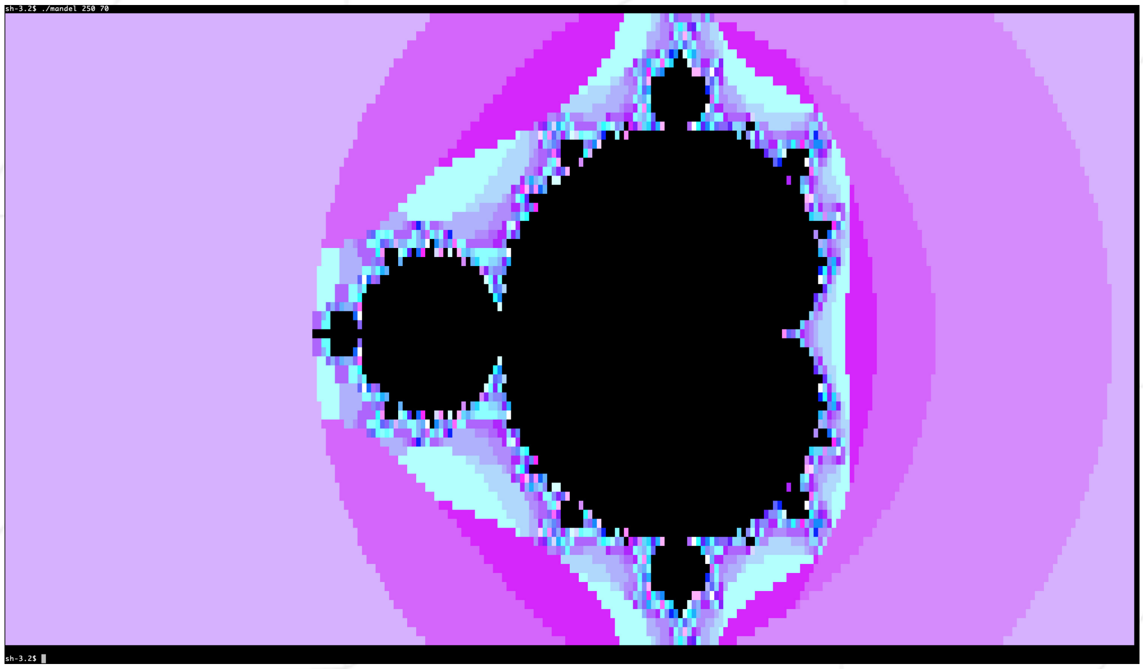
Dessert

aka Bonuses

Just in case you did find this main course not enough for your meal, you can get the dessert.

Create a yasl script named `mandel` that display in the terminal the Mandelbrot set (yes the fractal). You'll get more points if:

- you can specify the size of the displayed set as command line parameters
- you add some color



```
sh-3.2$ ./mandel 250 70
```

```
sh-3.2$
```

That's all for this rush !!