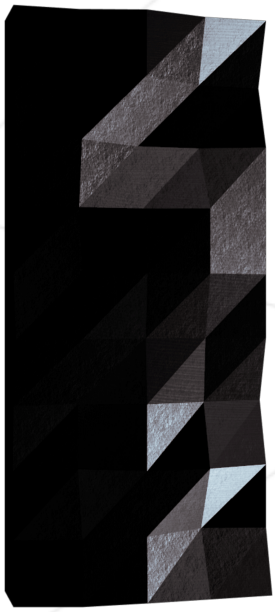

21sh

The vengence's return

Staff pedago pedago@42.fr

Summary: You'll have to start from your minishell and make it stronger to get little by littler closer to a real functionnal shell. You'll add couple of features such as multi-commande management, redirections as well as line edition that will allow you to use arrows for example.

Contents

I Préambule **2**
- I.1 The coder, the modern time Samurai! . . . 2
- I.2 Homarus . . . 2
  - I.2.1 Description . . . 2
  - I.2.2 Species . . . 2
  - I.2.3 Distribution . . . 3
  - I.2.4 Life cycle . . . 3

II Introduction **5**

III Objectives **6**

IV General Instructions **7**

V Mandatory part **9**

VI Bonus part **11**

VII Submission and peer correction **12**

Chapter I

Préambule

I.1 The coder, the modern time Samurai!

Do Kendo! At least you'll learn discipline.

I.2 Homarus

Homarus is a genus of lobsters, which include the common and commercially significant species Homarus americanus (the American lobster) and Homarus gammarus (the European lobster). The Cape lobster, which was formerly in this genus as H. capensis, was moved in 1995 to the new genus Homarinus.

I.2.1 Description

OHomarus is one of three extant genera of clawed lobsters to show dimorphism between claws – a specialisation into a crushing claw and a cutting claw. The other similar genera are Nephrops, which is much more slender, and has grooves along the claws and the abdomen, and Homarinus, the Cape lobster from South Africa, which is even smaller, and has hairy claws.

While analyses of morphology suggest a close relationship between Homarinus and Homarus, molecular analyses using mitochondrial DNA reveal that they are not sister taxa. Both genera lack ornamentation such as spines and carinae, but are thought to have reached that state independently, through convergent evolution. The closest living relative of Homarus is Nephrops norvegicus, while the closest relatives of Homarinus are Thymops and Thymopides.

I.2.2 Species

Eight extinct species are known from the fossil record, which stretches back to the Cretaceous, but only two species survive. These two species, the American lobster and the European lobster, are very similar and may have speciated as recently as the Pleistocene, during climatic fluctuations. The best characters for distinguishing them are the geographic distribution, with the American lobster in the western Atlantic and the European lobster in the eastern Atlantic, and by the presence of one or more teeth on the

underside of the rostrum in H. americanus but not in H. gammarus.

- Homarus gammarus: Homarus gammarus, known as the European lobster or common lobster, may grow to a length of 60 cm (24 in) and a mass of 6 kilograms (13 lb), and bears a conspicuous pair of claws. In life, the lobsters are blue, only becoming "lobster red" on cooking. Homarus gammarus is a highly esteemed food, and is widely caught using lobster pots, mostly around the British Isles.
- Homarus americanus: The American lobster, Homarus americanus, commonly ranges from 20–60 cm (8–24 in) in length and 0.5–4.1 kg (1.1–9.0 lb) in weight, but have been known to reach lengths of 64 cm (25 in) and weigh as much as 20 kg (44 lb) or more, making this the heaviest marine crustacean in the world. An average adult is about 23 cm (9 in) long and weighs 700–900 g (25–32 oz).
- Fossil species: The boundaries between Homarus and the extinct genus Hoploparia are unclear, and some species, such as Hoploparia benedeni have been transferred between the two genera. Eight species have been assigned to Homarus from the fossil record. They are:
  - Homarus americanus
  - Homarus brittonestris
  - Homarus davisi
  - Homarus hakelensis
  - Homarus lehmanni
  - Homarus mickelsoni
  - Homarus morrisi
  - Homarus neptunianus
  - Homarus travisensis

I.2.3 Distribution

The two extant species of Homarus are both found in the North Atlantic Ocean. H. americanus is found from Labrador to North Carolina in the western North Atlantic, while H. gammarus is found from Arctic Norway to Morocco, including the British Isles and the Mediterranean Sea.

I.2.4 Life cycle

Mating in Homarus is complex and is accompanied by a number of courtship behaviours. Males build mating shelters or burrows, and larger males can attract more females, producing a polygynous mating system. A few days before moulting, a female will choose a mate, and will remain in his shelter until the moult. The male will then insert a spermatophore into the female's seminal vesicle, where it may be stored for several years.

The eggs of Homarus species are laid in the autumn, being fertilised externally as they exit, and are carried by the female on her pleopods.

The eggs generally hatch in the spring as a pre-larva, which rapidly develops into the first larval phase. This is followed by three zoeal phases, the total duration of which can vary from two weeks to two months, depending on the temperature. At the following moult, the young animal becomes a post-larva, with a gross form resembling the adult lobster. Although it can swim, using its pleopods, the post-larva soon settles to the bottom and lives as a juvenile for 3–5 years.

As adults, Homarus species moult increasingly infrequently. The size at sexual maturity varies with temperature; it is around 70 mm (2.8 in) for female H. americanus in southern New England, but 100 mm (3.9 in) around the Bay of Fundy. In H. gammarus, the size at sexual maturity is less well defined, but is in the range 80–140 millimetres (3.1–5.5 in).

Figure I.1: H. americanus

Figure I.2: H. gammarus

Chapter II

Introduction

Thanks to the `Minishell` project, you discovered a part of what is behind the scene of a shell, such as the one you use everyday. And more specifically the process' synchronisation creation with functions like `fork` and `wait`.

The `21sh` project will make you go further by adding, amoungst other things, inter-process communication using pipes.

You'll discover, or rediscover if you worked on the `ft_select` project, `termcaps`. This library will allow you to add to your shell a line edition feature. You'll then be able to edit a typo made on your command without having to retype it completely as well as repeat a previous command using history. Of course you'll have to code those features. Good news is `termcaps` will help you do it, bad news is it'll not do it for you!

Chapter III

Objectives

`Unix` programmation is great, The school's 3 shell projects allow you to discover a big part of the system's API and it can only be good for you.

However, the shell projects are commands interpreter above all and initiate you to a very important part of IT: compilation. Interpreter are programs that read and execute other programs, unline compilators which translate other programs into other langages. Interpreters and compilators have more in common than they have differences though: whether it comes to executing or translating a program, first we need to understand the program itself, and be able to detect and reject malformed programs.

You probably know this already, but the set of commands we can send to a shell form a langage. This langage has lexical, syntactic and semantical rules, that will have to be respected by your 21sh by going through a set of very precise steps well documented on the internet. For example the "2.10 Shell Grammar" section of this document.

The key to a successful `21sh`, and `42sh` later on is a clear and well managed code organisation. Be sure that a simple space based `split` on your command line will not do the trick here. To avoid losing time, consider this solution as a one way ticket to disaster.

Here are couple of key words that i suggest you to properly understand: "lexical analysis", "lexer", "syntactic analysis", "parser", "semantic analysis","interpreter", and of course "abstract syntax tree" (or "AST").

Chapter IV

General Instructions

- This project will be corrected by humans only. You're allowed to organise and name your files as you see fit, but you must follow the following rules.
- The executable file must be named `21sh`.
- Your `Makefile` must compile the project and must contain the usual rules. It must recompile and re-link the program only if necessary.
- If you are clever, you will use your library for your `RTv1`. Submit also your folder `libft` including its own `Makefile` at the root of your repository. Your `Makefile` will have to compile the library, and then compile your project.
- Your project must be written in accordance with the Norm. Only norminette is authoritative.
- You have to handle errors carefully. In no way can your program quit in an unexpected manner (Segmentation fault, bus error, double free, etc).
- Your program cannot have memory leaks.
- You'll have to submit a file called `author` containing your username followed by a '\n' at the root of your repository.

```
$>cat -e author
xlogin$
```

- Within the mandatory part, you are allowed to use only the following libc functions:
  - `malloc`, `free`
  - `access`
  - `open`, `close`, `read`, `write`
  - `opendir`, `readdir`, `closedir`
  - `getcwd`, `chdir`
  - `stat`, `lstat`, `fstat`
  - `fork`, `execve`
  - `wait`, `waitpid`, `wait3`, `wait4`
  - `signal`, `kill`
  - `exit`
  - `pipe`
  - `dup`, `dup2`
  - `isatty`, `ttyname`, `ttyslot`
  - `ioctl`
  - `getenv`
  - `tcsetattr`, `tcgetattr`
  - `tgetent`
  - `tgetflag`
  - `tgetnum`
  - `tgetstr`
  - `tgoto`
  - `tputs`
- You are allowed to use other functions or other librairies to complete the bonus part as long as their use is justified during your defense. Be smart!
- You can ask your questions on the forum, on slack...

Chapter V

Mandatory part

To begin with, every `minishell` features are implicitly part of the `21sh` mandatory part. Furthermore you'll have to add the following new features:

- a line edition feature using the `termcaps` library Check the following description below.
- the "`;`" command line separator
- pipes "`|`"
- the 4 following redirections "`<`", "`>`", "`<<`" et "`>>`"
- file descriptor aggregation, for example to close the standard error:

```
$> ls
riri
$> rm riri; cat riri 2>&-
```

Here is a representativ example of commands your `21sh` must be able to execute correctly:

```
$> mkdir test ; cd test ; ls -a ; ls | cat | wc -c > fifi ; cat fifi
.   ..
      5
$>
```

Regarding the line edition, you must at least manage the followin features. The keys to be used are used as examples, you're free to use other ones as long as your shell remains logical and intuitive. The corrector will decide what's logical and intuitive, so be careful not to get carried away with creativity.

- Edit the line where the cursor is located.
- Move the cursor left and right to be able to edit the line at a specific location. Obviously new characters have to be inserted between the existing ones similarly to a classic shell.

- Use up and down arrows to navigate through the command history which we will then be able to edit if we feel like it (the line, not the history)
- Cut, copy, and/or paste all or part of a line using the key sequence you prefer.
- Move directly by word towards the left or the right using `ctrl+LEFT` and `ctrl+RIGHT` or any other reasonable combination of keys.
- Go directly to the beginning or the end of a line by pressing `home` and `end`.
- Write AND edit a command over a few lines. In that case, we would love that `ctrl+UP` and `ctrl+DOWN` allow to go from one line to another in the command while remaining in the same column or otherwise the most appropriate column.
- If the part between parenthesis of the command is not closed before you press `return`, the shell will return to the line and await for the end of the command. About the parenthesis part we're referring to command line including a part between quotes, doubles quotes, back quotes, parenthesis, hooks, accolades etc...
- Implement the `ctrl+D` and `ctrl+C` combination feature (knowing that `ctrl+C` to stop a program is a good thing too).

Chapter VI

Bonus part

We will look at your bonuses if and only if your mandatory part is EXCELLENT. This means that your must complete the mandatory part, beginning to end, and your error management must be flawless, even in cases of twisted or bad usage. If that's not the case, your bonuses will be totally IGNORED.

There are quite a few features that will appear in 42sh. Here is however a list of bonuses that you can implement immediately:

- Search through history using `ctrl+R`
- Implement a hash table for binary files
- Simple or advanced completion using `tab`.
- `Emacs` and/or `Vim` binding mode freely activable or deactivable.
- Syntactic shell coloration freely activable or deactivable.
- Any additional bonus that you will feel useful.

Chapter VII

Submission and peer correction

Submit your work on your `GiT` repository as usual. Only the work on your repository will be graded.

Good luck to all and don't forget your author file!