

Rapport du projet de programmation fonctionnelle :  
**DERNIÈRES NOUVELLES : LES  
EXTRA-TERRESTRES EXISTENT**

Loïc DUBARD

4 mai 2019

**Table des matières**

<b>1</b>	<b>Architecture du projet</b>	<b>3</b>
<b>2</b>	<b>Phase 1</b>	<b>3</b>
2.1	Objectifs . . . . .	3
2.2	Fonctionnement général du script . . . . .	3
<b>3</b>	<b>Phase 2</b>	<b>4</b>
3.1	Objectifs . . . . .	4
3.2	Fonctionnement général du script . . . . .	4
<b>4</b>	<b>Idées d'optimisation du code</b>	<b>5</b>

## Introduction

Le but de ce projet est d'implanter un protocole de communication avec les habitants des mondes lointains.

Plus précisément, il s'agira de prendre un message entrée et de donner la suite d'instructions permettant de piloter une (ou plusieurs) antenne(s) afin d'envoyer ledit message.

Dans l'urgence de cette nouvelle, nous avons développé un système d'antennes rudimentaire composées chacune d'une roue comportant les 26 lettres de l'alphabet et du caractère espace. Nous pouvons donc voir l'ensemble des symboles utilisables comme un cycle. Le caractère suivant A est B, ..., le caractère suivant Y est Z, le caractère suivant Z est espace suivi de A.

Toutes les antennes sont initialement positionnées sur le caractère espace.

Les commandes disponibles pour manipuler une antenne sont au nombre de 3 :

- "N" : passe au caractère suivant. ;
- "P" : revient au caractère précédent ;
- "E" : envoie la commande d'émission du caractère courant.

Nous devons commencer rapidement à envoyer des messages afin d'ouvrir au plus vite la voie à nos ingénieurs financiers et autres analystes de données. Le projet sera donc découpé en trois phases (une antenne, plusieurs antennes et compression du signal).

Durant toutes ces phases vos programme devront lire deux lignes sur leur entrée standard.

la première ligne comportera un entier  $n$  donnant le nombre d'antennes disponibles. Pour la première phase, cet entier sera toujours égal à 1. la deuxième ligne comportera le message lui-même.

Le programme affichera donc une suite de commandes permettant d'émettre le message.

# 1 Architecture du projet

```
> ipf_s2/  
| > doc/  
|   | rapport.pdf  
|   | rapport.tex  
| > src/  
|   | main.ml  
|   | tests.ml  
|   | user.ml  
| [> bin /]  
|   | main.out  
|   | tests.out  
| [> archive /]  
|   | loic_dubard_ipf.tar.gz  
| Makefile  
| README.md  
| (...)
```

Les dossiers entre crochets [] sont des dossiers créés par la compilation (partielle ou complète), les noms entre parenthèses () sont là pour indiquer que des dossiers/fichiers ne sont pas précisés.

## 2 Phase 1

### 2.1 Objectifs

Dans cette phase nous avons accès à une seule antenne, on analyse uniquement a seconde entrée standard.

### 2.2 Fonctionnement général du script

Le principe est le suivant : je défini la roue comme une liste des caractères (en majuscule) :

```
let roue = [' ','A','B','C','D','E','F','G','H','I','J','K','L';  
            'M','N','O','P','Q','R','S','T','U','V','W','X','Y','Z']
```

Puis, je demande à l'utilisateur le message à traduire grâce aux fonctions [parse\_input ()] et [split\_string s] fournies par le professeur.

Le couple (n :int,l :char list) ainsi obtenu est passé dans la fonction [**commande** (n,l) roue].

On introduit alors la fonction [**translate** c roue] qui se charge de renvoyer la commande correspondant à l'envoi de la lettre c :char, sous forme de string et le nombre de déplacement de la roue à faire. (Il suffit pour cela de récupérer l'indice (noté i) du caractère c dans roue et de renvoyer i fois 'N' si i<14 sinon 27-i fois 'P').

On introduit aussi la fonction [**tourner** c n roue] dont le but est de retourner le nouvel état de la roue en tournant n fois (par récurrence) vers la gauche si c='N' et vers la droite si c='P'.

La fonction **commande** agit par récurrence sur la liste de caractères l de cette manière :

- à la première itération, on traduit le premier caractère de la liste en sa commande (une string) grâce à la fonction **translate** puis tourne la roue dans le bon sens et le bon nombre de cran avec la fonction **tourner**.
- à la  $n^{ieme}$  itération, le  $n^{ieme}$  caractère est traduit en sa commande (en une string) en utilisant la roue obtenue après la  $(n - 1)^{ieme}$  itération puis tourne la roue de même que à la première itération.

## 3 Phase 2

### 3.1 Objectifs

Le changement de place d'une roue ('N' ou 'P') prenant 3 secondes et la sélection d'une antenne ('Sn' où n est le numéro de l'antenne à activer) prenant 1 seconde, l'objectif devient donc d'optimiser le temps de calcul pour l'envoi d'un message. (l'émission 'E' d'une commande à une antenne prend 5s)

La première entrée sera le nombre d'antennes disponibles et la seconde entrée restera le message à envoyer (en majuscule).

### 3.2 Fonctionnement général du script

J'introduit d'abord la fonction [**quelle\_roue** l c i] où l est la liste de roue actuelle, c le caractère à traduire et i l'indice de la roue sur laquelle on est à l'entrée de la fonction. Cette fonction choisi dans l la roue qui donne le plus petit nombre de cran à tourner pour avoir c peu importe le sens.

Une autre fonction à introduire est la fonction [**tourner\_list** c n i l] qui en utilisant **tourne** de la phase 1 tourne n fois la roue d'indice i dans le sens de c ('N' ou 'P') dans l.

Pour cette phase j'ai donc créé la fonction [**commande2** (n,m)] où n est le nombre d'antennes et m le message à traduire. Cette fonction crée une liste de n roues en phase initiale puis fait le calcul par la récurrence décrite ci-dessous :

- à la première itération, on choisi la roue à utiliser grâce à la fonction **quelle\_roue** on traduit le premier caractère de la liste en sa commande (une string) grâce à la fonction **translate** de la phase 1 puis tourne la roue choisie dans la liste de roues dans le bon sens et le bon nombre de cran avec la fonction **tourner\_list**.
- à la  $n^{ieme}$  iteration, le  $n^{ieme}$  caractère est traduit en sa commande (en une string) de même que pour la première itération en utilisant la liste de roues obtenue après la  $(n - 1)^{ieme}$  itération puis envoie à l'itération suivant la liste de roues bien modifiée.

## 4 Idées d'optimisation du code

Au lieu de choisir la roue à utiliser lettre après lettre, on peut tracer un arbre  $n$ -aire (où  $n$  est le nombre de roues) et trouver le chemin le plus court pour la phrase à traduire. On optimiserai le temps d'exécution du programme non pas lettre par lettre mais pour toute la phrase.

Cependant, cette manière de calculer le programme à envoyer aux antennes me semble très coûteuse en temps et en mémoire. C'est pour cela que j'ai choisi de ne pas l'implémenter.