# Rapport Livrable 2 - Assembleur PYTHON

*Paco LARDY–NUGUES, Abdourahmane MBAYE,*
*Idriss ABDOULWAHAB, Lorenzo AZERINE*
**SICOM**

**02/10/2023**

Lien vers dépôt **Gitlab**.

## 1 Tâches

### 1.1 Reprise de la fonction regexp-match.c (Responsable : Paco LARDY–NUGUES)

Travail réalisé :

- Réalisation de *regexp.c* ainsi que *regexp-match.c*. Il s'agissait d'une adaptation du programme de base. J'ai du adapter aussi la fonction *char_to_queue*

- Tests associés à cette fonction : *regexp*.

Avancés :

- J'ai des problèmes si un groupe de caractère contient un ou plusieurs caractères interdits. En effet, la fonction *read_bracket* qui lit ce qu'il y a entre crochets, renvoie si il y a une erreur *EXIT_FAILURE* avec un message. Cependant, pour que *re_match* marche, il faut qu'elle récupère quand même la position *end*, même si il y a une erreur dans l'expression régulière. Or dans le cas des crochets, je n'y arrive pas. Je suis entrain de travailler dessus.

- J'ai également des problèmes de fuites de mémoires.

Illustration :

**Code Listing 1** – Illustration erreur

```
./bin/regexp-match.exe "[+]" "a"
Erreur caractere special sans chaine
```

Ici, le message vient de la fonction *read_bracket*. Or ici, *end* ne va pas pointer sur *'a'*, j'ai donc essayé de faire des conditions pour quand même faire que dans la fonction *char_to_queue*, on puisse

récupérer cette erreur pour renvoyer à la fin ce que j'ai appelé une *file impossible*, afin que quand *re_match* la détecte, elle puisse agir en conséquence en renvoyant *end* pointant sur le début de *source*.

- Pour ce qui est du reste, je n'ai pour l'instant pas détecté d'erreurs.

**Code Listing 2** – Prototype

```
1
2  int re_match_except(int characters[256],list_t regexp,char *source,char **end);
3  int issFull(int characters[256]);
4  int re_match_zero_or_more(int characters[256],list_t regexp char *source,char**end,int complement);
5  int re_match(list_t regexp, char *source, char **end);
6  int re_match_empty(list_t regexp, char *source, char **end);
7  int re_match_one_or_more(int characters[256],list_t regexp,char *source,char **end,int complement);
8  int re_match_zero_or_one(int characters[256],list_t regexp,char *source,char **end,int complement);
```

## 1.2 Réalisation de la fonction lexem_read (Responsable : Lorenzo AZERINE)

Travail réalisé :

- Fonction *lexem_read* : prends le chemin d'un fichier .conf, lis ligne par ligne à l'aide de la fonction *fgets* et crée les lexèmes associés avec le type, le contenu, la ligne et la colonne elle renvoie un tableau de lexems.

La fonction lex_read(char *filename) prends en entrée le chemin du fichier source de définition des lexems et renvoie un tableau de lexem.

La fonction lit chaque ligne du fichier :

- Si la ligne commence par un "#" elle crée un lexem commentaire pour la ligne

- Sinon, elle lit le début de la ligne définissant le type du lexem,

Ensuite, séparé par une tabulation, elle récupère l'expression régulière Pour tester cette fonction, nous avons pris un fichier .conf organisé de facon similaire à celui présenté sur le sujet La difficulté principale pour implémenter cette fonction était la lecture de la ligne, j'ai utilisé fgets qui lit entièrement la ligne. Il fallait donc trouver un paramètre permettant de faire la distinction entre la définition du type et de l'expression régulière. Le choix a été fait de prendre la tabulation car l'espace semblait pouvoir être à l'origine de problème si jamais un type contenait un espace. Cette fonction n'emet aucune indication sur la défintion de l'expression régulière ou du type de lexem elle se contente de les lire et les regrouper

## 1.3 Implantation d'une sous fonction lex_def (Responsable : Idriss ABDOULWAHAB

Travail réalisé :

- Réalisation de la fonction *list_t lex_def(char * file* preenant en entrée un fichier texte et qui crée une liste de *lexem_t* comportant le type du lexème ainsi que sa valeur.

**Code Listing 3** – Prototype

```
1  list_t lex_def(char *file);
```

Avancés :

- Le prototype de la fonction est finie, les tests ne sont malheureusement pas concluant, (segmentation fault, et mauvaise lecture du fichier). J'ai eu un peu du mal à comprendre comment concevoir une telle fonction au début, mais je suis actuellement sous une phase de débuggage.

- Tests : J'ai tout d'abord créé un fichier *test_lexem.c* composée d'un main qui dans un premier temps affiche la liste créé par la fonction *lex_def* (afin de m'assurer que la fonction fonctionne bien).

## 1.4 Implantation de la fonction lex (Responsable : Abdourahmane MBAYE)

Avancés :

- Réalisation de la fonction *lex(char* regexp_file, char* source_file)*

Elle prend la liste renvoyée par *lex_read* puis essaye de matcher ces types dans le code source python pour identifier les différents lexems présents.