Compte Rendu du Livrable 1

I- Répartition des rôles et organisation:

1- *Répartition des rôles*:

Yassine: Chef de Projet
Jala et Siryne: Développeurs

2- *Organisation:*

Nous avons un groupe chat dans lequel nous nous informons de l'avancée de notre travail et des actions faites sur Git. Yassine a divisé le travail de manière homogène et s'est intéressé aux difficultés rencontrées par les développeurs par l'intermédiaire de réunions Zoom régulièrement afin de constater l'avancée du travail et de débloquer certains points.

II- Pour la répartition des tâches dans le projet :

1- *Création par Yassine de la structure dictionnaire:*

Il a commencé par définir une structure qui permet de stocker le **nom**, l'expression régulière **exp_str** qui est en chaîne de caractères, puis l'expression régulière en forme de **liste** de **char group**. Cette même structure est intégrée dans une **file** de type **FIFO** qui va permettre de stocker l'ensemble des expressions enregistrées dans le fichier configuration. L'ajout de certaines fonctionnalités pour le type queue a été nécessaire, notamment la fonction qui permet d'afficher l'ensemble des expressions "extraites" et enregistrés dans le "dictionnaire" (le dictionnaire est en réalité la liste contenant l'ensemble des expressions) et la fonction qui permet de supprimer et libérer l'espace alloué au dictionnaire. À noter qu'il était possible de convertir la queue en liste et ensuite utiliser les fonctions données pour print et delete, ce choix n'a pas été fait suite à des problèmes rencontrés (non résolus) avec les listes génériques.

Le code qui permet de tester la fonction pour "remplir" le dictionnaire est donné dans le fichier **test-lecture-config.c**.

Les principaux problèmes rencontrés étaient en rapport avec l'espace mémoire alloué (segmentation fault) et des problèmes liés à un conflit de structures avec les listes génériques. Les erreurs de segmentation fault ont été réglées grâce à l'utilisation de l'outil Valgrind et les problèmes de conflit avec les listes génériques a été résolu en créant une fonction qui affiche et supprime mais de façon non générique.

2- *Création par Siryne de la fonction de création des listes de char_group pour modéliser les lexèmes puis les comparer au code source*

Siryne a créé un fichier de configuration qui présente le nom des lexèmes ainsi que leur définition sous la forme d'expressions régulières qui peuvent être trouvés au sein d'un fichier assembleur. Ces lexèmes ont été placés dans ce fichier dans un ordre particulier, de haut en bas. En effet, ce fichier permet de créer un dictionnaire de lexèmes qui sera utilisé

pour reconnaître les lexèmes contenus dans le fichier assembleur source. Cela implique qu'il faille d'abord prendre en compte les expressions régulières les moins générales puis celles qui le sont de plus en plus (de haut en bas).

Ensuite, Siryne a effectué la fonction queue_t **split_regexp**(char*regexp) qui prend en argument une expression régulière extraite du fichier de configuration. Cette fonction étant assez longue, des petites fonctions ont été implémentées pour en faciliter la lecture. Elle a pour objectif de créer une queue puis une liste génériques dont le champ **void*content** de chaque maillon pointe sur une structure **char_group**, elle-même contenant un tableau de 256 caractères qui comporte les caractères '0' ou '1' dans les cases représentant le code ASCII de différents caractères utilisés dans les expressions régulières prises en argument et un typé énuméré définissant le nombre d'apparition d'un caractère ou groupe de caractères en fonction de la présence de différents symboles tels que '+', '*', '?'. Cette liste permet de représenter des groupes de caractères pour une seule regexp grâce à plusieurs tableaux reliés entre eux sous forme de queue. Nous avons d'abord utilisé des queue, car elles permettent de respecter l'ordre adopté pour les lexèmes dans le fichier de configuration (les maillons étant enfilés à la fin et non au début de la queue comme cela aurait été le cas pour les listes). Il existe différents cas à prendre en compte lors de la prise en compte de l'expression régulière par la fonction. On parcourt alors à l'aide du pointeur regexp chacun des caractères constituant l'expression régulière et on les compare à différents cas. On distingue les cas où l'on a un groupe de caractères entre crochets qui se trouveront alors spécifiés dans le même tableau de char_group et les caractères qui se trouvent à l'extérieur et qui seront dans différents tableaux formant des listes de structures pointant sur des char_group. Lorsqu'un caractère est reconnu, on place un '1' dans la case du tableau group[256] correspondant à son code ASCII et '0' dans les autres. On initialise alors ce tableau avec des '0'. Quand un caractère particulier est reconnu, on crée un tableau initialisé, on le modifie puis on alloue un nouveau maillon dont le champ content pointe vers ce nouveau tableau. Afin de pouvoir avancer dans l'expression régulière, on implémente le pointeur sur cette expression pour passer au caractère suivant. Ensuite, on rappelle récursivement la fonction afin de pouvoir bénéficier de tous les cas possibles pour reconnaître le nouveau caractère pointé.

Comme les fonctions utilisant les **char_group** prend en argument des types liste_t, la queue_t renvoyé par la fonction est ensuite convertie en list_t grâce à la fonction **queue_to_list** effectuée par Siryne, qui manie les pointeurs de **queue_t** pour en faire une **list_t**.

Les principaux problèmes rencontrés étaient la compréhension de la formalisation du problème, comprendre comment nous allions mettre en place ces définitions et comment manier les listes génériques. Aussi, les différentes disjonctions de cas pour interpréter les différents caractères possibles dans la regexp n'a pas été simple.

3- *Création par Jalal de la fonction de lecture du fichier assembleur source*

Jalal a commencé par écrire le code implémentant les fonctions **re_match_one_or_more** (qui represente l'operateur '+') et **re_match_zero_one_or_one** (qui represente l'operateur '?'). La première version du code prenait en entrée des chaînes de caractères ce qui posait problème pour les groupes de caractères (comme par exemple "[a-f]"). Donc il était obligé de modifier le code des fonctions pour qu'il soit adapté à la nouvelle structure de données qui permet de représenter les groupes de caractères.

III- Travail restant:

Une segmentation fault détectée par Valgrind demeure. Nous devons encore régler ce problème.