RASCOL Laura | AKIKI Melissa | DUCHADEAU Romain

Compte rendu : livrable 1

Ce compte rendu permet de mettre en lumière comment tester les fonctions écrites ainsi que le déroulement en équipe de la réalisation du livrable 1.
Nous sommes globalement satisfaits de l'avancement de ce livrable même si nous n'avons pas pu le terminer à temps. Le travail à été relativement bien réparti et nous avons pu apprendre de nos erreurs pour en tirer des leçons pour les prochains livrables.

I) Travail réalisé: le readme

##Projet sei AKIKI-DUCHADEAU-RASCOL
##---

#data_base.txt
But: fichier regroupant tous les types avec les definitions associées; c'est le dictionnaire complet de lexem

#data_base.txt.test
But: Dictionnaire de lexem réduit pour les tests

#queue.c
But: Définir toutes les fonctions nécéssaire à l'utilisation des queue

Avancement: Tout fonctionne parfaitement

Pour tester: Tapez d'abord "make" dans le terminal pour compiler, Tapez ensuite la commande " ./bin/unit/test-queue.exe -v ".
Cette commande va tester les fonctions : queue_new, queue_empty, enqueue, queue_to_list, queue_peek, dequeue, queue_length.

#list.c
But: Définir toutes les fonctions nécéssaire à l'utilisation des queue

Avancement: Tout fonctionne parfaitement

Pour tester: Tapez d'abord "make" dans le terminal pour compiler, Tapez ensuite la commande " ./bin/unit/test-list.exe -v "

#char_group.c
But: Définir toutes les fonctions nécéssaire à l'utilisation des chargroup

Avancement: Tout fonctionne parfaitement

Pour tester: Tapez d'abord "make" dans le terminal pour compiler, Tapez ensuite la commande " ./bin/unit/test-char_group.exe -v "

#database.c
But: Lire le data_base.txt, et inclure les informations triée dans une queue

Avancement: Tout fonctionne parfaitement

Pour tester: Tapez d'abord "make" dans le terminal pour compiler, Tapez ensuite la commande " ./bin/unit/test-database.exe data_base.txt.test ".

#regexp.c
But: Comparer une source char* avec une expression régulière

Avancement: 3 tests sur 5 fonctionnel

Pour tester: Tapez d'abord "make" dans le terminal pour compiler, Tapez ensuite la commande " ./bin/unit/test-regexp.exe ".

#regexp-cg.c
But: Comparer une source char* a un char_group

Avancement: non fonctionnel

Pour tester: Tapez d'abord "make" dans le terminal pour compiler, Tapez ensuite la commande " ./bin/unit/test-regexp-cg.exe ".

#parse.c
But: Prendre un char* en argument et mettre les différentes unités logiques dans une liste de chargroup

Avancement: Compile mais segmentation fault

Pour tester: Tapez d'abord "make" dans le terminal pour compiler, Tapez ensuite la commande " ./bin/unit/test-parse.exe text_a_trier ".
le programme va prendre le text_a_trier et mettre les différents char-group dans une liste avant d'imprimer la liste

#lexer.c

But: Elément final: Lire un fichier assembler .pys et renvoyer une queue de lexem

Avancement: Base certainement fonctionnelle mais incomplète car nécessite de finir toutes les fonctions préalablement.

Pour tester: Tapez d'abord "make" dans le terminal pour compiler, Tapez ensuite la commande "./bin/unit/test-lexer.exe ".

II) Répartition des tâches

Ici vous retrouverez, les tâches et sous tâches permettant de valider l'analyse lexicale demandé pour le 04 Octobre 2022.

— T.1 "Parsing des expressions régulières" : Découpage d'une expression régulière en la liste de ses composants ; prendre en main le sujet, le bootstrap fourni pour démarrer et les outils fournis dans ce bootstrap pour soutenir les tests.

— T.1.1 "Ecriture de queue.s" (resp. RASCOL Laura) : Analyse des besoins nécéssaire lors de l'utilisation de file. (espéré 4 jours, réel 1 semaine)
— T.1.2.1 "Ecriture des fonctions queue_new, queue_empty, enqueue, queue_to_list, queue_peek, dequeue, queue_length" (resp. RASCOL Laura) : Analyse des besoins nécéssaire lors de l'utilisation de file ;
— T.1.2.1.1 "Ecriture de la fonction list_add_last" (resp. RASCOL Laura) : Fonction nécessaire pour la fonction queue_to_list.
— T.1.2.1.1 "Test de la fonction list_add_last" (resp. RASCOL Laura) :Test avec unitest, ajouté aux tests précédents pour les fonctions dans list.c.
— T.1.2.1 "Test des fonctions queue_new, queue_empty, enqueue, queue_to_list, queue_peek, dequeue, queue_length" (resp. RASCOL Laura) : Tests réalisés avec unitest;

— T.1.2 "Ecriture de regexp.c et sa version adapter regexp_cg.c" (resp. AKIKI Melissa) (espéré 1 semaine, réel 2 semaines).

— T.1.2.1 "Fonction rematch" (resp.AKIKI Melissa)
rematch est la fonction de base avec récursivité qui va lire des expressions régulières 'regexp' et qui a été retravaillé après sous le nom 'rematch_cg' pour intégrer les char_group. notamment rematch est une fonction récursive qui fera appel aux fonctions qui suivent.

— T.1.2.1.1 "Fonction re_match_zero_or_more" (resp.AKIKI Melissa)
Fonction qui s'assure de la bonne syntaxe au cas d'une '*'.c-a-d s'assure que l'expression régulière figure dans la source zéro ou plusieurs fois.
— T.1.2.1.2 "Fonction re_match_one_or_more" (resp.AKIKI Melissa)
Fonction qui s'assure de la bonne syntaxe au cas d'un '+'.c-a-d s'assure que l'expression régulière figure dans la source au moins une fois.
— T.1.2.1.3 "Fonction re_match_zero_or_one" (resp.AKIKI Melissa)
Fonction qui s'assure de la bonne syntaxe au cas d'un '?'.c-a-d s'assure que l'expression régulière figure dans la source une fois ou aucune.
— T.1.2.1.4 "Fonction re_match_negation" (resp.AKIKI Melissa)
Fonction qui traite le complément de l'expression régulière si elle est présente dans la source.
— T.1.2.2 "Test de regexp" (resp.AKIKI Melissa) Analyse des tests unitaires Fournis dans tests/unit/test-regexp.c et des tests d'intégration fournis dans le répertoire 01_test_regexp_parse
Pour le parsing d'expressions régulières ; écriture d'autres tests ; validation en commun des tests écrits ;

— T.1.2.3 "Fonction rematch_cg" (resp.AKIKI Melissa)
Fonction qui prend une source et une expression régulière de type char_group. et selon le type d'opérateur appelle la fonction correspondante de traitement.(les cas 'NULL' et " " ont été pris en considération).
A noter que regexp est une fonction nécessaire pour d'autres fonctions, notamment "Parse.c" lui fera appele pour lire le fichier .pys.

— T.1.2.3.1 "Fonction re_match_zero_or_more" (resp.AKIKI Melissa)
Fonction qui s'assure de la bonne syntaxe au cas d'une '*'.c-a-d s'assure que l'expression régulière de type char_group figure dans la source zéro ou plusieurs fois.
— T.1.2.3.2 "Fonction re_match_one_or_more" (resp.AKIKI Melissa)
Fonction qui s'assure de la bonne syntaxe au cas d'un '+'.c-a-d s'assure que l'expression régulière type char_group figure dans la source au moins une fois.
— T.1.2.3.3 "Fonction re_match_zero_or_one" (resp.AKIKI Melissa)
Fonction qui s'assure de la bonne syntaxe au cas d'un '?'.c-a-d s'assure que l'expression régulière type char_group figure dans la source une fois ou aucune.
— T.1.2.3.4 "Fonction re_match_negation" (resp.AKIKI Melissa)
Fonction qui traite le complément de l'expression régulière type char_group si elle est présente dans la source.
— T.1.2.3.5 "Fonction is_equal" (resp.AKIKI Melissa)
Fonction intermédiaire à qui toutes les autre rematch vous appeler pour pouvoir tester l'égalité entre deux char*.(pris en considération le caractère '.').
— T.1.2.4 "Test de regexp_cg" (resp.AKIKI Melissa) Analyse des tests unitaires Fournis dans tests/unit/test-regexp_cg.c et des tests d'intégration fournis dans le répertoire 01_test_regexp_parse
Tester des expression les plus intéressantes notamment des cas particulier

---

— T.1.3 "Ecriture de data_base.txt" (resp. AKIKI Melissa)

(espéré 1 jour, réel 2 jours).
Contient la définition des expressions régulières utilisée par le language assembleur.

— T.1.4 "Ecriture de database.c" (resp. DUCHADEAU Romain et RASCOL Laura) : Le but est ici de pouvoir lire la database et stocker les types avec les définitions liées. (espéré 1 semaine, réel 1 semaine)
— T.1.4.1 "Créer la structure data_t" (resp. RASCOL Laura) : Le but de cette structure est de pouvoir stocker les informations, de def et de type de manière liée, ceci simplifiera donc la lecture de ces information, contenues dans une queue.
— T.1.4.2 "Ouvrir le fichier database.txt, et en extraire les infos type et def" (resp. DUCHADEAU Romain)
— T.1.4.3 "Stocker les infos type et def dans une queue" (resp. RASCOL Laura)
— T.1.4.4 "Tester la fonction database.c" (resp. RASCOL Laura) : Tests réalisés avec unitest;

— T.1.5 "Ecriture de chargroup.c" (resp. DUCHADEAU Romain) : Choix de la structure la plus adapté, pour stocker les informations et les fonction associées à cette structure.(espéré 1 jour, réel 1 jour)

— T.1.5.1 "Créer la structure la plus efficace" (resp. DUCHADEAU Romain) : Le but était destocker à la fois les valeurs admises et leur occurence sous une forme lisible par regexp.
— T.1.5.2 "Ecrire les fonctions new_char_group, delete_char_group, char_group_print" (resp. DUCHADEAU Romain)
— T.1.5.1 "Tester les fonctions : new_char_group, delete_char_group, char_group_print " (resp. RASCOL Laura) : Tests réalisés avec unitest;

— T.1.3 "Ecriture de parse.c" : Permet de lire un data_t et de le rendre la définition lisible par un algorithme comme re_match
(espéré 1 semaine, pas fini).

— T.1.3.1 “Ecriture de read_crochet” (resp. DUCHADEAU Romain): le but de cette fonction est de lire l’intérieur des crochets et de stocker ces valeurs sous forme de char group, elle est utilisée dans la fonction re_read.
— T.1.3.2 “Ecriture de re_read” (resp. DUCHADEAU Romain): le but de cette fonction est de lire un char* et de stocker les valeurs “une à unes” dans une liste de char groupe.
— T.1.5.1 "Tester les fonctions : new_char_group, delete_char_group, char_group_print " (resp. RASCOL Laura) : Tests réalisés avec unitest; pas encore fait car premier jet ne fonctionne pas encore.

— T.1.3 "Ecriture de lexer.c" : Fonction finale du livrable 1
(espéré 1 semaine, pas fini).-en cours de travail
— T.1.3.1 “Ecriture de find_regexp” (resp. DUCHADEAU Romain): le but de cette fonction est de lire un fichier assembleur et de renvoyer une queue de lexem. En soit c’est la fonction finale.

III) Bilan

Nous ne sommes pas loin d’avoir terminé ce livrable. Nous pensons qu’il faudra encore 2 personnes jusqu’à la fin de la semaine pour complètement le terminer pendant qu’une troisième pourra se concentrer sur l’avancement du livrable 2.