Rapport Livrable 1 - Assembleur PYTHON

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

26/09/2023

Lien vers dépôt **Gitlab**.

1 Tâches

1.1 Construction en mémoire de la structure de données représentant une expression régulière (Responsable : Lorenzo AZERINE)

Travail réalisé :

- Fonction qui construit en mémoire la structure de données représentant l'expression régulière. Cette fonction ne fait que lire l'expression régulière et la stock en mémoire telle quelle dans la structure de données (normalement fonctionnelle).

- Tests associés à cette fonction.

Avancés :

- Modifier la fonction précédente pour qu'elle stock un groupe de caractères lorsque c'est nécessaire plutôt que l'expression lue (ex : si chaine entrée '[a-z]' la fonction l'alphabet entier et pas [a-z]'.

- La fonction sera dans le module de la structure de donnée.

Code Listing 1 – Prototype

```
1
2 char_group list_regexp(char regexp[ ] );
```

1.2 Implémentation du type abstrait groupe de caractères ainsi que le type opérator (Responsable : Abdourahman MBAYE)

Travail réalisé :

- Implémentation du type abstrait groupe de caractères ainsi que le type opérator qui sera inclus

dans la liste chaînée.

- Tests unitaires pour vérifier le bon fonctionnement de ce type abstrait. Tests sur la création et la modification d'une variable de ce type.

Code Listing 2 – chargroup.h

```
1
2  #include <assert.h>
3  #include <stdio.h>
4  #include <stdlib.h>
5
6  #ifndef _CHARGROUP_H_
7  #define _CHARGROUP_H_
8
9  typedef enum {
10     ZERO_OR_ONE,
11     ZERO_OR_MORE,
12     ONE_OR_MORE,
13     ONE_TIME
14 } operator;
15 /* On utilise un type enumere avec la constante ZERO_OR_ONE qui equivaut a ?, la
16 constante ZERO_OR_MORE qui equivaut a * et enfin la constante ONE_OR_MORE qui
17 equivaut a + */
18
19
20 typedef struct {
21      int complement; // vaut 1 si il y a ^ devant le groupe de carateres et 0 sinon
22      operator op;
23      int characters[256]; // le groupe de caracteres
24    } char_group;
25
26
27 #endif /* _CHARGROUP_H_ */
```

1.3 Implantation de la fonction regexp-match (Responsable : Idriss AB-DOULWAHAB

Travail réalisé :

- Réadapter la fonction donnée par le sujet en fonction d'une chaine de caractères

- Créer une fonction int re_match_one_group(char_group * group,char caractere) qui compare un groupe de char d'une liste chaînée et un char (Utile pour comparer chaque élément d'une source).

- Créer des fonctions annexes pour les différentes opérations : $+, *, .$

- Concevoir la fonction finale : int re_match(list_t regexp , char *source , char **end) ; qui prend une liste chaînée transformée et comprenant les fonctions annexes.

- Faire un main() ; qui prend en argument dans le terminal les chaînes de caractères : l'expression régulière et la source.

- Test du main()

- Test sur les différents opérations

Problèmes :

- Pour l'instant la liste chainée est bien transformée, mais la fonction re_match() ne fonctionne pas dans son ensemble.

- Certaines opérations ne marchent pas, spécialement le $*$ et le $+$ (zero_or_more et one_or_more). L'itération ne semble pas s'effectuer (erreur sur le code des deux fonctions ?, doute sur la condition d'arrêt de ces deux fonctions)

- Les groupes de caractères [a-z] par exemple ne semble pas être reconnu

- De même pour le ^ qui signale une segmentation fault (erreur de memoire ?)

Avancés :

- La plupart des fonctions ont été conçues, je me retrouve actuellement dans la partie test et débogage de la fonction re$_m atch$.

1.4 Implantation du type file circulaire générique (Responsable : Paco LARDY–NUGUES)

Travail réalisé :

- Implantation du type file générique et réalisation des tests associés.

Code Listing 3 – queue.h

```
1
2  #ifndef _QUEUE_H_
3  #define _QUEUE_H_
4  #ifdef __cplusplus
5  extern "C" {
6  #endif
7  #include <pyas/list.h>
8    typedef struct link_t {
9    void *content;
10   struct link_t *next;
11 } link_t;
12
13 typedef struct {
14   link_t *head;
15   link_t *tail;
16 } fifo;
17
18 fifo *fifo_new(void);
19 int fifo_empty(fifo *q);
20 void fifo_enqueue(fifo *q, void *object);
21 list_t *fifo_to_list(fifo *q);
22 void fifo_print(fifo *q, void (*print_callback)(void *));
23 void print_char_callback(void *data);
24 void print_int_callback(void *data);
25 void *fifo_dequeue(fifo *q);
26 int fifo_length(fifo *q);
27 void *fifo_first(fifo *q);
28 void *fifo_last(fifo *q);
29 void fifo_delete(fifo **q);
30 void print_char_group_callback(void *data);
31 #ifdef __cplusplus
32 }
33 #endif
34 #endif /* _QUEUE_H_ */
```

Avancés :

L'implantation a permis de commencer à travailler sur la fonction *regexp-read*. Il a fallut créer des fonctions de callback pour l'utiliser dans fifo_print, notamment pour afficher le type *chargroup*.

1.5 Implantation de regexp-read (Lorenzo AZERINE & Paco LARDY–NUGUES)

Travail réalisé :

- Ajout dans la fonction *queue.c* des fonctions permettant de lire des groupes de caractères, et aussi de transformer une chaîne de caractère en groupe de caractère.

Code Listing 4 – queue.h - Ajouts

```
1
2  #ifndef _QUEUE_H_
3  #define _QUEUE_H_
4  #ifdef __cplusplus
5  extern "C" {
6  #endif
7  #include <pyas/list.h>
8
9  char_group * read_bracket(int k,  char * p_regexp, int * k_final);
10 char_group * read_point(char_group * regexp2) ;
11 operator is_special(int k, char * p_regexp) ;
12 fifo * char_to_queue(char regexp[]) ;
13
14 #ifdef __cplusplus
15 }
16 #endif
17 #endif /* _QUEUE_H_ */
```

- Création de la fonction *regexp-read.c*.

Avancés :

Les fonctions sont implantées et marchent bien. Nous avons rajouté des conditions en cas d'erreur de syntaxe dans une expression régulière.

2 Conclusion Livrabe 1

À terme de ce premier livrable, nous avons réalisé comlètement **une** fonction sur **deux**, ainsi que les tests associés à chaque module.

On a essayé plusieurs combinaisons, notamment pour la gestion des tabulations et des retours à la ligne, mais par exemple dans ce cas, on obtient :

Code Listing 5 – Cas particulier de regexp-read

```
1
2  ./regexp-read.exe '[a-z\t]+'
3  One in "\abcdefghijklmnopqrstuvwxyz", one or more times.
```

On remarque donc que le caractère dd'échappement à l'intérieur d'un crochet est mal interprété. C'est parce que la position du caractère $'\backslash'$ dans la table ASCII est située aant le caractère $'a'$.

Malheuresement, les objectifs du livrable 1 ne sont pas atteint, nous redoublerons d'effort pour avoir la fonction regexp-match.c le plus vite possible.