Rapport du projet de programmation impérative

Seydina Dieng

09 Janvier 2022

Table des matières

I. Démarche générale

I.1 Structures de données et opérations

Pile

La pile permet de stocker les différents états de l'automate. J'ai choisi l'implémentation par liste chainée dont chaque maillon a une valeur correspondant à un état.

```
struct stack_base{
        elem val ;
        struct stack_base* next ;
} ;
typedef struct stack_base* stack ;
typedef int elem ;
```

Les différentes opérations implémentées sur la pile sont :

- Initialiser une pile ;
- Ajouter un élément au sommet d'une pile ;
- Dépiler et/ou Retourner l'élément au sommet d'une pile ;
- Désallouer une pile.

Automat

J'ai choisi ensuite de récupérer les éléments de l'automate passé en argument dans une structure. Cette structure composée de 5 lignes tout comme les automates, récupère exactement chaque octet de chaque ligne d'automate. Je l'ai préféré à un stockage dans une matrice car il faudrait en créer une pour chaque ligne, là où automat peut contenir toutes les lignes du fichier.

```
struct automat{
        unsigned char* line1 ;
        unsigned char* line2 ;
        unsigned char* line3 ;
        unsigned char* line4 ;
        unsigned char* line5 ;} ;
typedef struct automat automat ;
```

Les différentes opérations implémentées pour automat sont :

- Retourner un élément d'une ligne d'un automat ;
- Rechercher un groupement d'élément sur une ligne d'un automat ;
- Récupérer une ligne particulière d'un automat ;
- Désallouer un automat .

Couple

La fonction réduit renvoie un couple d'éléments. J'ai donc implémenté une structure couple dont le premier élément est un int et le deuxième un char.

```
struct couple{
        int n ;
        char A ;
} ;
typedef struct couple couple ;
```

Les différentes opérations implémentées pour réduit sont :

- Récupérer un couple ;
- Récupérer le premier élément d'un couple ;
- Récupérer le deuxième élément d'un couple.

I.2 Récupération des données du fichier passé en argument

J'ai tout d'abord opté pour une relecture du fichier à chaque fois qu'on avait besoin d'une donnée. Par souci de temps d'exécution, j'ai alors décidé de stocker les données dans la structure automat. Ainsi j'ai créé une fonction « data » qui retourne un automat de même structure et de même composition que le fichier en argument.

Donc dans l'automat a retourné par data, on aura par exemple dans a.line1 tous les éléments de la ligne 1 du fichier passé en argument et rien que ceux-ci.

I.3 Les différentes fonctions nécessaires

Différentes fonctions ont été utilisées pour exécuter les actions nécessaires pour l'utilisation d'un automate LR(1).

- La fonction action (automat donnees, int s, char c) retourne l'élément numéro s*128+c de la ligne 2 ;

- La fonction reduit (automat donnees, int s) retourne l'élément numéro s de la ligne 3 et l'élément numéro s de la ligne 4 ;
- La fonction decale (automat donnees, int s, char A) retourne l'élément suivant un groupement (s, A) avant le premier groupement de 173, sur la ligne 5 ;
- La fonction branchement (automat donnees, int s, char A) retourne l'élément suivant un groupement (s, A) entre le premier et le deuxième groupement de 173, sur la ligne 5. Il a fallu donc trouver le premier groupement de 173 et se placer après pour pouvoir rechercher (s, A) avec une boucle while ;
- La fonction free_automat(automat donnees) libère la mémoire allouée pour le stockage de chaque ligne.

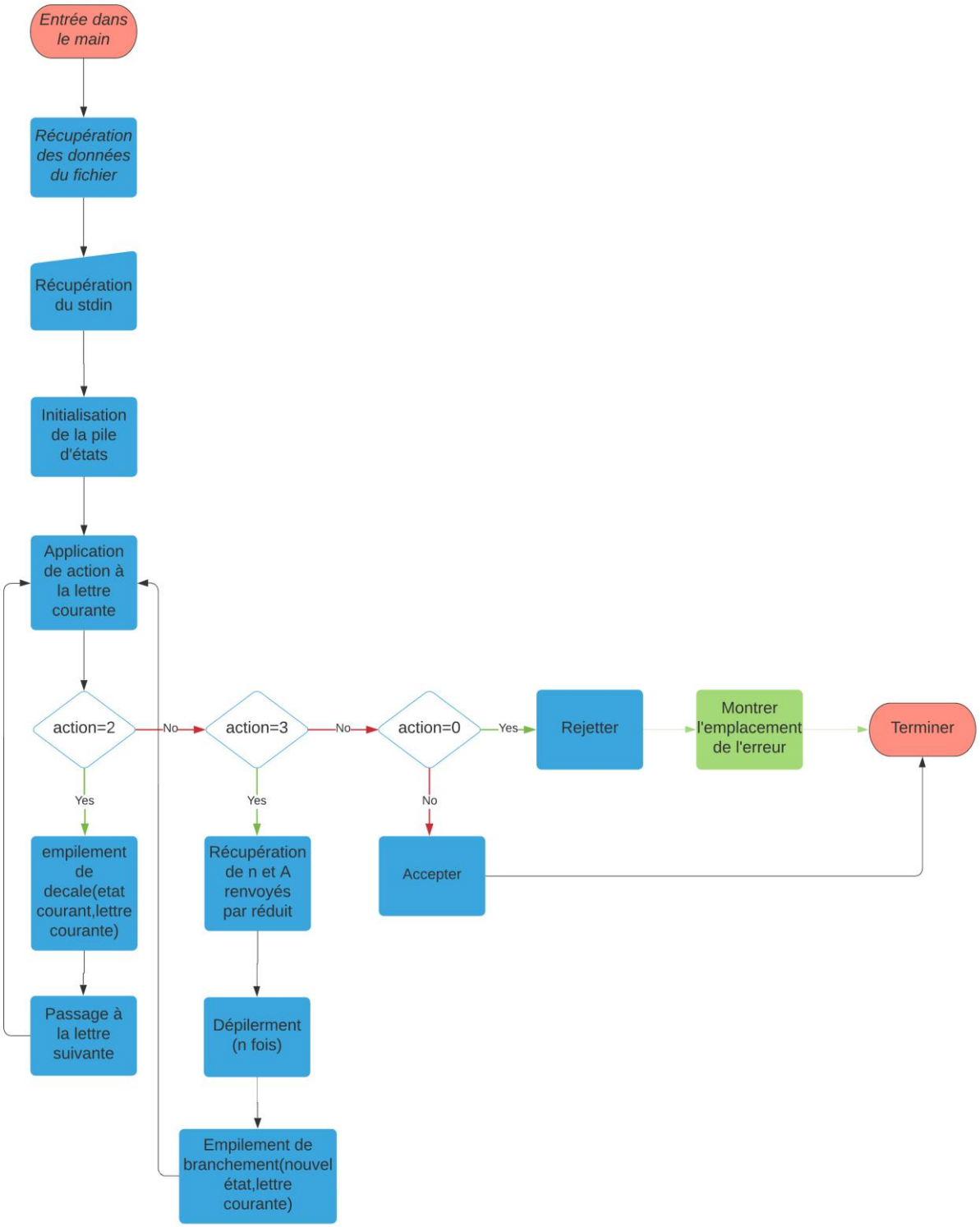
I.4 Exécution des actions

Chaque résultat d'action nous amène à faire une manipulation particulière. Après avoir récupéré la ligne sur l'entrée standard avec la fonction fgets, on parcourt la chaine de caractère et on passe chaque caractère et l'état courant en argument de la fonction action. On récupère la valeur retournée par action et on traite les cas comme suit :

- Si action=0 écrire « Rejected ! » sur le stdout ;
- Si action=1 écrire « Accepted ! » sur le stdout ;
- Si action=2 appeler la fonction 'push' pour empiler decale(etat_courant, caractère) et passer à la lettre suivante (en incrémentant l'entier i parcourant le buffer qui contient la chaine de caractère entrée par l'utilisateur) ;
- Si action=3 récupérer les deux composantes renvoyées par réduit, dépiler avec 'pop' la pile d'états n+1 fois (où n est la valeur retournée par réduit_1) pour récupérer le nouvel état courant qu'on va rempiler, puis empiler branchement(etat_courant, caractère) avec 'push'.

II. Fonctionnement général du programme

Le fonctionnement général du programme est le suivant :



III. Bonus

Pour montrer l'emplacement de l'erreur, j'ai isolé le cas où l'erreur se trouvait au niveau d'un '\n' car cela veut dire que l'agencement des mots n'est pas correct.

IV. Limites du programme et idées d'optimisation

Pour récupérer l'état courant je fais un pop puis un push. J'aurais aimé trouver une autre solution pour ce faire, sans pour autant faire des dépilements et empilements successifs. Toutefois je n'ai malheureusement pas eu d'idées pour l'éviter.
Aussi je voulais que l'utilisateur puisse vérifier plusieurs entrées au lieu de relancer l'exécutable à chaque fois. Mais mon programme ne fonctionne pas si j'utilise une boucle infinie pour demander plusieurs entrées utilisateurs.