# INF203 - Travaux pratiques, séance 12
# Développement d'un interpréteur de commandes (2/2)

## 1 Etat des lieux et objectifs

Après le dernier TP, l'interpréteur de commandes est capable de traiter :
— des instructions "simples", sans variables ;
— les variables : leur affectation, et les instructions où il leur est fait référence ;
— les instructions qui font référence aux arguments de la ligne de commande.
L'interpréteur que vous récupérez au début de ce TP correspond à cet état d'avancement.
L'objectif est maintenant de traiter des scripts shell qui comportent des instructions conditionnelles ou des boucles while non imbriquées.

Tout au long de ce TP, vous devrez créer vos programmes de test : des fichiers `.sh` avec des instructions conditionnelles ou avec des while, et vérifier que leur comportement est bien celui attendu.
**[a]** Joignez ces fichiers à votre compte-rendu. ■

## 2 Quelle sorte de ligne ?

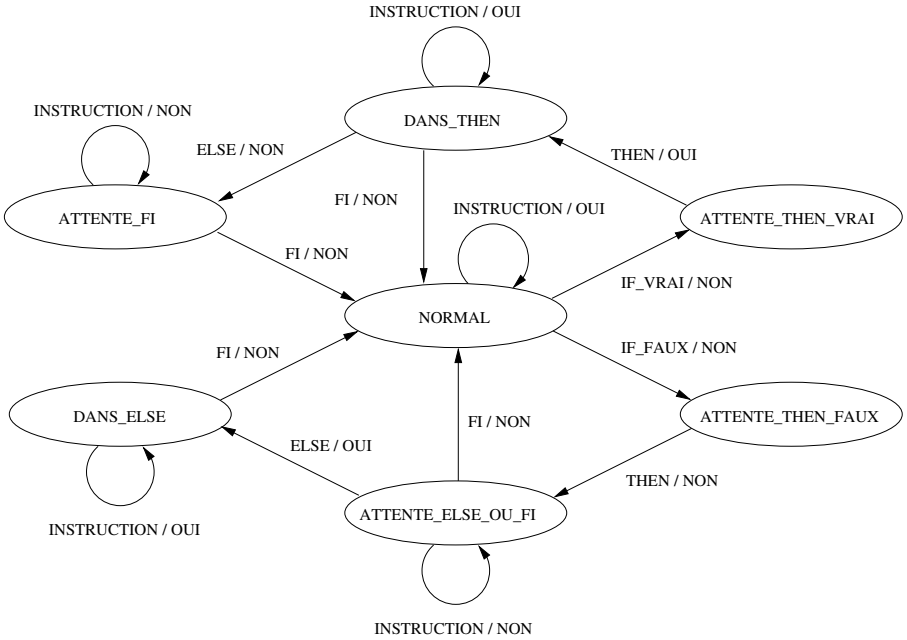
Les symboles d'entrée de l'automate (du if ou du while) représentent la "nature" de la ligne courante. Celle-ci est déterminée par son premier mot : soit c'est un mot clé (ou mnémonique) parmi l'ensemble {`if, then, else, fi, while, do, done`} et la ligne est alors une commande interne pour notre shell, soit (par défaut), c'est une instruction.
**[b]** A quel endroit la fonction `decode_entree` est-elle appelée ? Complétez cette fonction à l'endroit indiqué. ■

## 3 Automate d'une instruction conditionnelle

L'automate permettant de gérer une instruction conditionnelle dans un script, tel que vous l'avez vu en TD, est représenté sur la page suivante. Pour ne pas surcharger l'ensemble, l'état d'erreur et les transitions qui y mènent ne figurent pas sur le dessin. mais il faut bien comprendre que dans la définition de l'automate, il y a une transition et une sortie pour chaque couple (etat, entrée) ! Toutes les transitions manquantes mènent à l'état d'erreur avec une sortie à NON. A l'aide du dessin de l'automate, complétez la fonction `init_automate_commandes` du fichier `commandes.c` (uniquement pour ce qui concerne le if).

Complétez également la fonction `analyse_commande_interne` : calcul de la sortie `code_sortie`, et changement d'état en fonction de `etat_courant` et de `entree`.
**[c]** Vous joindrez le texte du fichier `commandes.c` à votre compte rendu (imprimez, ne recopiez pas). ■
**[d]** Retrouvez dans la fonction `main` de l'interpréteur, à quel endroits sont appelées les fonctions définies dans `commandes.c` et indiquez les dans votre compte rendu. ■

Compilez et testez votre interpréteur avec les jeux de test que vous avez préparés, et corrigez les erreurs si nécessaire ...




# 4 Boucle while

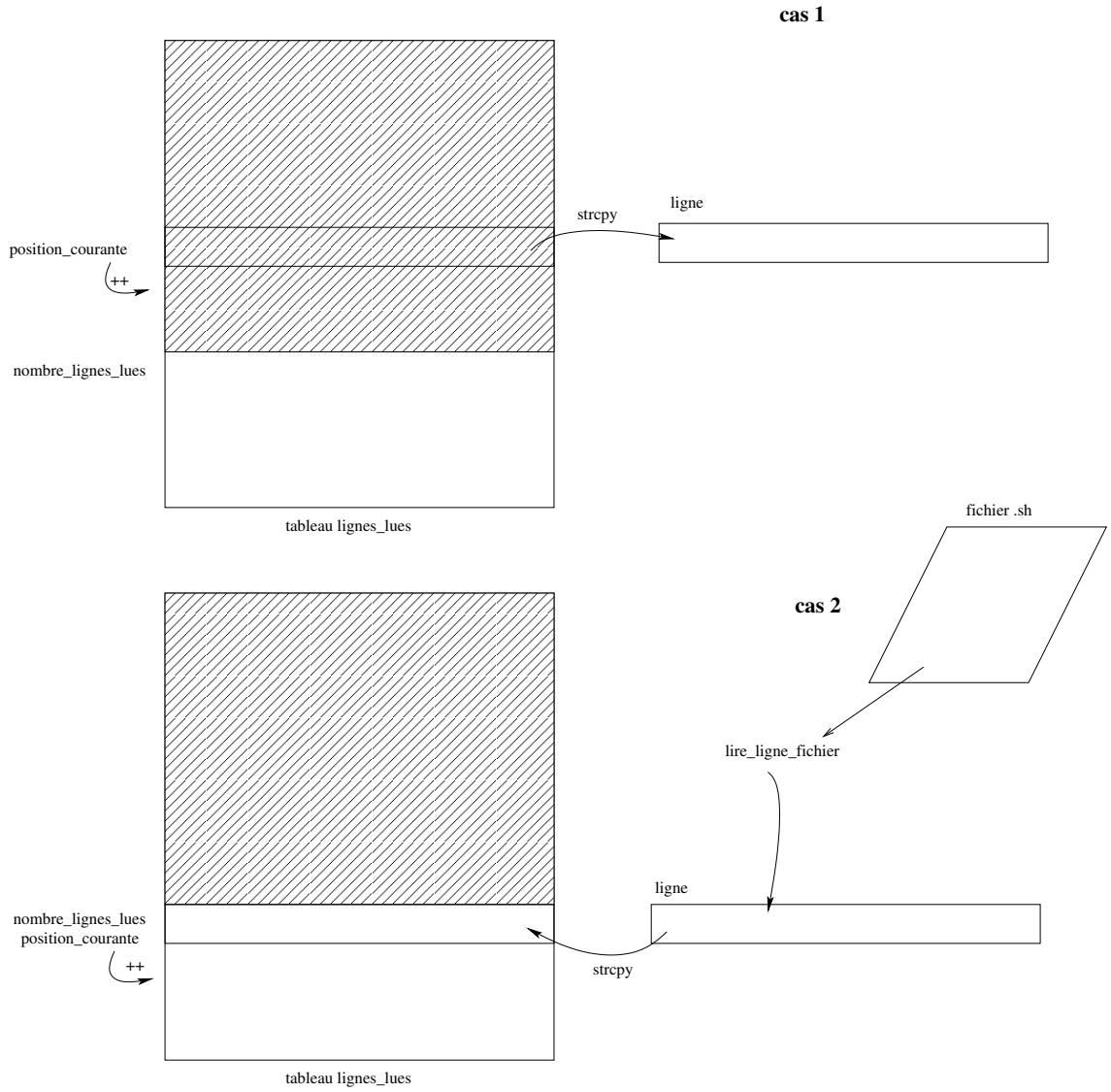
Pour la boucle while, les choses se compliquent. Il va falloir, en plus de la gestion de la structure de contrôle elle-même, être capable de réexécuter des lignes déjà lues. Nous découperons le travail à réaliser en deux parties.

## 4.1 Retours en arrière

Pour être capable de réexécuter durant plusieurs itérations un même ensemble de lignes, il va falloir les numéroter et les stocker. Pour cela éditez le fichier `lignes.c` et ajoutez y les éléments vus en TD :

- des déclarations de variables globales : un tableau permettant de stocker les lignes lues, un entier permettant de comptabiliser le nombre de lignes déjà lues, un entier permettant de connaître le numéro de la ligne courante (ou de la prochaîne ligne à lire), et un descripteur de fichier dans lequel lire les lignes non encore lues.
- une fonction `init_historique` permettant d'initialiser les variables de l'historique.
- une fonction `lire_ligne` prenant en argument une chaîne de caractères. Cette fonction devra :
  - si la ligne courante a déjà été lue, copier dans la chaîne donnée en argument la ligne stockée correspondant à la position courante (**cas 1** de la figure)
  - sinon, utiliser `lire_ligne_fichier` pour lire la prochaine ligne à exécuter et la stocker (**cas 2** de la figure).
- modifiez le programme principal pour initialiser votre structure de stockage des lignes et utliser `lire_ligne` au lieu de `lire_ligne_fichier` dans la boucle de l'interpréteur.
- ecrivez une fonction `obtenir_numero_ligne` permettant de connaître le numéro de la ligne courante, une autre `aller_a_la_ligne` permettant d'effectuer un retour à une ligne déjà lue de numéro donné

**[e]** Comment pouvez-vous tester votre mécanique de retour arrière ? ■




## 4.2 Structure de contrôle

Pour gérer la structure de contrôle du while, nous utiliserons la même technique que dans le cas de la structure conditionnelle : récupération des mots clé directement sur la ligne de commande lue, détermination de l'entrée associée et utilisation d'un automate pour suivre l'évolution du `while/do/done`. En plus des sorties OUI/NON, nous ajoutons une sortie BOUCLE indiquant qu'il faut revenir en début de boucle, c'est-à-dire lors de l'occurence d'un `done`, après un test positif pour le `while` associié.

**[f]** Dessinez l'automate permettant de gérer la structure `while/do/done`. ■

**[g]** Complétez le code de `commandes.c` afin de :

- gérer les `if/then/else/fi` et les `while/do/done` avec l'unique automate de l'interpréteur
- lors de l'occurence d'un `while`, sauvegarder la ligne courante pour pouvoir y revenir plus tard (en utilisant la fonction `obtenir_numero_ligne`) et exécuter le reste de la ligne de commande pour déterminer le résultat du test
- lors de l'occurence d'un `done`, après un test positif pour le `while` associé (sortie BOUCLE), revenir à la ligne du while (en utilisant la fonction `aller_a_la_ligne`).

■