UFR d'Informatique
L3 Informatique, Math-Info et Japonais-Info

Grammaires et analyse syntaxique
Année 2024-2025

TP n°2

Ocamllex

Mise en route

Nous allons utiliser le générateur d'analyseur lexical `ocamllex`. On rappelle que `ocamllex` prend en entrée un fichier de spécification (dont le nom se termine par `.mll`) ayant trois/quatre parties :

- un prologue entre accolades `{` et `}` contenant du code Ocaml (typiquement des définitions utilisées dans la suite) qui sera placé au début du code engendré ;
- une séquence de définitions d'expressions rationnelles ;
- plusieurs points d'entrée de l'analyse lexicale qui regroupent une séquence d'expressions rationnelles avec les actions associées.
- un possible épilogue entre accolades `{` et `}` contenant du code OCaml qui sera placé à la fin du code engendré.

La façon la plus simple d'utiliser `ocamllex` est d'avoir un unique fichier `.mll` contenant tout pour obtenir un programme exécutable. Prenons comme exemple le fichier `partie_1/lexeur1.mll` qui contient un analyseur lexical qui prend un fichier en entrée, imprime tous les chiffres, calcule leur somme et l'imprime.

```
{
  exception Eof
  let somme = ref 0
}

let digit = ['0'-'9']

rule lexeur = parse
  | digit as c  { print_char c;
          somme := !somme + (int_of_char c) - (int_of_char '0') }
  | _           { lexeur lexbuf }
  | eof         { raise Eof }

{
  let ch = open_in (Sys.argv.(1)) in
  let lexbuf = Lexing.from_channel ch in
  try
    while true do
      lexeur lexbuf
    done
  with Eof -> (print_newline(); print_int !somme; print_newline()
     )
}
```

La génération d'un programme OCaml se fait avec `ocamllex lexeur1.mll`
la compilation ensuite avec `ocamlc -o main lexeur1.ml`
et l'exécution finalement avec `./main test.txt`

Exercice 1 *Examiner le programme `lexeur1.ml` engendré par ocamllex et repérer le prologue et l'épilogue.*

Une façon plus avancée d'utiliser ocamllex est de considérer les tokens. Pour cela on définit un type `token` dans un fichier à part (par exemple `token.ml`) qui contient les différents tokens que le lexeur peut utiliser et le lexeur produira une suite de tokens utilisée par exemple dans un programme principal `main.ml`. Par exemple, les trois fichiers `lexeur2.ml`, `token.ml` et `main.ml` ont la même fonctionnalité que l'unique fichier `lexeur1.ml`.

La compilation se fait soit laborieusement

```
ocamllex lexeur2.mll
ocamlc token.ml
ocamlc lexeur2.ml
ocamlc -o main lexeur2.cmo token.cmo main.ml
```

soit via un projet `dune` (fourni)

```
dune build main.exe
```

(l'exécution se fait dans ce cas avec `_build/default/main.exe test.txt`).

Exercice 2 *Modifier les fichiers de sorte qu'on calcule la somme de tous les entiers (suite de chiffres) d'un fichier. On ne traite que des entiers positifs sans `"+"` devant.*
On n'hésitera pas à traiter de nouveaux types de jetons dans token.ml si besoin. Tester la solution obtenue sur le fichier `test.txt`.

On devrait trouver 6801.

En route vers Mars

En l'an 10 000, les hommes ont enfin pu coloniser Mars. Il était temps, il ne restait plus vraiment beaucoup de régions habitables, l'industrie spatiale ayant grandement contribué à cette situation. Les humains habitant Mars sont appelés les Martiens. Le fichier fourni représente une liste de Martiens célèbres. Chaque ligne du fichier a le format suivant : le nom et le(s) prénom(s) : plusieurs suite de lettres alphabétiques et d'espaces ; un espacement ; la date de naissance suivie par un tiret `"-"` suivi ou non par la date de décès ; un espacement suivi ou non par un "commentaire" c'est-à-dire du texte (quelconque) entre crochets `"[....]"`.

Par "espacement", on entend n'importe quel suite non vide d'espaces et de tabulations. Pour simplifier, il n'y a pas de lettres accentuées, ni de lettres qui ne soient pas dans l'alphabet anglais.

Nous vous donnons un fichier `mars.mll` qui contient la définition des espacements et des sauts de lignes (gérées différemment dans les fichiers DOS et Linux). Il reconnaît les mots composés de lettres alphabétiques et indique leurs positions avant de les afficher, un par ligne.

Exercice 3 *Dans un premier temps, écrire un programme qui repère uniquement la partie date de naissance et de mort, et les affiche une par ligne. Une ligne est donc soit du genre* `10020-10100` *soit* `10020-`. *Ensuite faire la même chose mais en précisant "encore vivant" quand c'est le cas.*

Exercice 4 *Afficher les lignes entières concernant les personnes décédées et ayant un commentaire. On suppose que le fichier fourni est conforme au schéma, on ne cherchera donc pas à contrôler les parties non concernées par la requête.*

Exercice 5 *Vérifier que toutes les lignes sont conformes au schéma et afficher les mauvaises avec leur numéro de ligne. Il devrait y en avoir trois qui ne le sont pas : les corriger à la main pour faire l'exercice suivant.*

S'il reste du temps

Exercice 6 *Si on voulait faire une analyse plus structurée du texte, il faudrait dans un premier temps engendrer les tokens correspondants aux éléments du texte. On voudrait donc comme dans la fin de la mise en route, reconnaître des tokens pour*

1. *les noms et prénoms (le même token sera utilisé)*
2. *les dates*
3. *les tirets*
4. *les commentaires*
5. *les fins de ligne*

On fera en sorte d'afficher les tokens successifs. Il est possible de diminuer le nombre de tokens reconnus, puis en fonction du temps à disposition, l'augmenter progressivement.