Travaux dirigés

Introduction à la Compilation

1-Analyse lexicale

Jean-Christophe Le Lann, Quentin Ducasse

Janvier 2023

Exercice 1 : Identification de lexèmes

L'objectif de cet exercice est de vous familiariser à la découpe d'un texte en *lexèmes* (en anglais *Tokens*). Dans ce premier exercice, nous procédérons "à la main", par exemple avec papier et crayon, et sans utilisation d'outils informatiques particuliers. Il s'agira pour vous de lire différents codes sources, et d'établir la séquence des ses lexèmes.

Pour rappel, un lexème est un regroupement de trois informations différentes :

- un identifiant du `type` du lexème.
- une `valeur` correspondant au texte capturé.
- une `position` en ligne et colonne, autorisant un reporting d'erreur pertinent.

La difficulté tient au choix du nommage du type de lexème. Il n'existe pas de convention particulière, mais ce nommage doit être judicieux, car les lexèmes seront utilisés lors de la phase suivante du compilateur. Un nommage intuitif vous facilitera votre travail.

Il existe plusieurs catégories de types de lexèmes. Voici quelques exemples de nommage de type des lexèmes :

- mots clés : `kw_if`
- signes de ponctuation : `comma`, `dot`, `semicolon`...
- identifiant : `ident`
- valeurs litérales : `int_lit`, `float_lit`, `str_lit`

Question a : Etablir la liste des lexèmes du code Python suivant.

```python
import numpy as np
from logging import Logger

class NumberLogger(Logger):
    def __init__(self, file):
        self.file = file


    def log_nb(self, nb):
        self.log("INFO", np.format_float_scientific(nb))

if __name__ == "__main__":
    lg = NumberLogger()
    lg.log_nb(1.245)
}
```

Question b : Etablir la liste des lexèmes du code C suivant.

```c
#include <stdio.h>

typedef struct complex {
    float real;
    float imag;
} complex;

complex add(complex n1, complex n2) {
    complex temp;
    temp.real = n1.real + n2.real;
    temp.imag = n1.imag + n2.imag;
    return temp;
}


int main() {
    complex n1 = {.float = 1, .imag = 3};
    complex n2 = {.float = 2, .imag = 4};
    complex result;

    result = add(n1, n2);

    printf("Sum = %.1f + %.1fi", result.real, result.imag);
    return 0;
}
```

Question c : Etablir la liste des lexèmes d'un code personnel, d'une langage de votre choix, que vous avez eu l'occasion d'écrire dans un autre cours.

Exercice 2 : Expressions régulières

L'analyse lexicale repose sur la reconnaissance de *motifs* dans un texte, réalisée à l'aide d'*expressions régulières*. Ces expressions régulières sont un outil puissant, présent dans tous les langages de programmation, soit de manière native (décrit dans la grammaire du langage), soit à travers des bibliothèques. Les expressions régulières se présentent elles-mêmes sous forme textuelle, dans un langage dédié, qui possède donc sa propre syntaxe...La formulation d'une expression régulière n'est pas standardisée, et varie légèrement d'un langage à un autre.
Une expression régulière est transformée (compilée !) en automate. Cet automate transite d'un état à un autre en consommant les caractères d'un texte les uns à la suite des autres. Si le motif décrit par l'expression régulière est *reconnu*, l'automate se retrouve dans un état dit *terminal* et la reconnaissance d'arrête.

Question a : A l'aide du site web "Pythex" (ou "Rubular"), établir l'expression régulière unique décrivant les nombres en virgule flottante. On pourra bien entendu y inclure la notation scientifique, comme dans `-123.12345e-42`

Question b : Un fichier contenant des mots français est fourni sous Moodle. Ecrire un programme Python permettant de lister l'ensemble des mots vérifiant les motifs suivants :

- Mots se terminant par `ion`.
- Mots possédant le motif `at`, non-suivi de 't', ni de 'r', ni de 'i'.

Exercice 3 : Lexer simple

Sous Moodle, un lexer simple, écrit en Python, est fourni. Il pourra vous servir dans votre futur projet. Il découpe le texte en une suite de lexèmes, qu'il retourne à l'utilisateur sous la forme d'un tableau de lexèmes.
Il procède relativement naïvement, en testant successivement toutes les expressions régulières ; dès qu'une expression régulière reconnaît, dans le texte source, le motif décrit, un lexème est crée, ajouté dans le tableau. Le motif est consommé dans le texte source, ce qui lui permet de passer au lexème suivant, etc. Nous verrons à l'exercice 4, comment il est possible d'optimiser cette étape de l'analyse lexicale. Toutefois, notre lexer naïf fonctionne avec suffisamment de rapidité pour les cas d'usage courant.

Question : Compléter la liste des expressions régulières permettant d'automatiser l'analyse lexicale réalisée (à la main) dans l'exercice 1.

Exercice 4 : Lexer optimisé

Le lexer de l'exercice précédent n'est pas optimisé : il tente plusieurs expressions régulières, avant de tomber sur celle qui `match` la chaîne courante. Imaginons par exemple plusieurs mots clés qui commencent par une même séquence de lettres (prefixe) : dans ce cas, chaque expression régulière commence par la recherche du même préfixe, ce qui est manifestement inefficace. Les lexers optimisés combinent plusieurs expressions régulières et élaborent un seul et même automate, qui évite l'écueil précédent. Les algorithmes de ces lexers permettent de passer de la combinaison "ou" d'une liste d'expressions régulières à un automate performant (DFA : deterministic Finite Automation).

Le but de l'exercice est ici de construire manuellement un tel lexer optimisé et de comparer sa performance par rapport au lexer naïf de l'exercice précédent.

Considérons ici l'ensemble des mots clés suivants : [**à décider**] `if`,`while`, `input`, `func`, `for`, `in`, `body`, `begin`, `end`, `entity`

Question 1 A l'aide de l'outil `https://cyberzhg.github.io/toolbox/nfa2dfa`, entrez ces mots clés combinés par un "ou" (barre verticale) et visualisez l'automate de reconnaissance.

Question 2 Coder un automate unique de reconnaissance des lexèmes précédents.

Question 3 On cherche à mesurer le gain en performance du lexer optimisé par rapport au lexer naïf. Ecrire un script Python (ou autre) qui génère aléatoirement un fichier contenant un grand nombre des lexèmes précédent. Mesurer la performance respective des deux lexers en lexèmes / seconde.