Projet de Design C++ : réalisation d'une calculatrice

Option Géologie Numérique, ENSG 2013-2014
Guillaume Caumon, Thierry Valentin, Sophie Viseur.

Partie 1. Introduction

Une calculatrice peut être vue comme une boîte noire (Figure 1) qui prend en entrée une chaîne de caractère correspondant à une opération (i.e. : expression mathématique) et qui envoie en sortie le résultat (nombre) de cette opération.

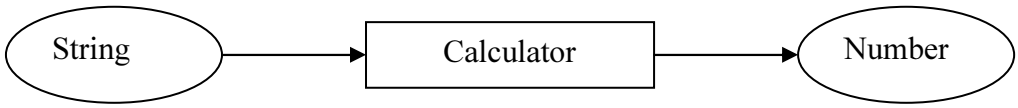


Figure 1: *Schéma d'une calculatrice, entrée et sortie.*

Afin de pouvoir calculer ce résultat, la calculatrice se doit de comprendre la structure et le sens de l'expression qui lui est donnée sous forme d'une chaîne de caractères. Ainsi, il est important de pouvoir définir quels types d'expressions (de phrases) pourront être donnés en entrée et selon quelles règles. Cette opération revient à établir un langage.

Les notions de langage informatique et celles qui en découlent vont donc être abordées dans une première partie. Dans une deuxième partie, ces notions seront appliquées dans le cas particulier de la mise en place d'un programme de réalisation d'une calculatrice.

Partie 2. Quelques notions sur le langage, la grammaire et la compilation

« L'intérêt d'un langage informatique est de véhiculer une sémantique, les aspects lexicaux et syntaxiques sont les maux nécessaires pour y parvenir ». (J. Menu, 1994)

2.1. Langages : vocabulaire, syntaxe et sémantique

Un langage est formé d'un **vocabulaire** (mots formés de caractères issus d'un **alphabet**), d'une **grammaire** et d'une **sémantique**. Par exemple, "chhat" n'est pas un mot du vocabulaire français (erreur lexicale). "la plage souris le chat" est constituée de mots valides mais n'est pas une phrase grammaticalement correcte (erreur syntaxique). "Le chat boit un bateau" est une phrase grammaticalement correcte mais sémantiquement incorrecte (ou pour le moins de sens obscur).

Un langage informatique est un formalisme pour la représentation et la manipulation des informations. Il suit les mêmes règles que les langages communs tels que le français : il est constitué d'un alphabet (i, +, *) formant un vocabulaire (for, i, while, 3.5), utilisé pour écrire des phrases (int i=2;) qui véhiculent une certaine information (sémantique).

Ainsi, trois aspects sont souvent distingués lors de l'analyse d'une phrase d'un langage : l'aspect lexical, syntaxique et sémantique.

2.1.1. Analyse lexicale :

Elle découpe la phrase d'entrée en une séquence de mots (unité lexicale) du vocabulaire considéré. L'intérêt est de vérifier si les caractères utilisés appartiennent à l'alphabet du langage et s'ils forment des mots autorisés par le langage.

Exemple de fautes lexicales :

- 2.5.5 en mathématique et C++ ;
- "arbritaire" en français ;
- Attention : 2,5 peut être une erreur lexicale si votre vocabulaire stipule que la virgule est spécifiée par un point '.' (convention anglaise) mais sera valide avec la convention française.

2.1.2. Analyse syntaxique :

La **grammaire** définit la syntaxe d'une phrase. Elle énonce les lois sur les ordonnancements possibles des mots du vocabulaire. L'analyse syntaxique vérifie donc que les séquences d'unités lexicales respectent une structure de phrase imposée par la **grammaire** du langage et rejette les constructions invalides.

Exemples de faute syntaxique :

- une parenthèse qui manque dans un programme
- "3++2=5" est une expression arithmétique incorrecte.

La syntaxe est un élément déterminant du langage, par exemple : "x++" est non acceptable selon la syntaxe arithmétique, mais acceptable pour la syntaxe d'un langage informatique tel que C++.

2.1.3. Analyse sémantique :

Elle vérifie le sens véhiculé par les phrases du langage.

Exemple:

- int i = 2.5; est une erreur de sémantique
- Attention!!!
  if(2+4 == 1) cout<<"Hello"<<endl;
  est sémantiquement correct en C++, le fait que 2+4 ne soit pas égal à 1 ne remet pas en cause la sémantique du langage C++.
  Par contre: "le chat" = = true est sémantiquement faux car le langage C++ n'autorise pas de comparer un booléen avec une chaîne de caractères.

NB : Selon la complexité ou la nature du langage, certaines phases d'analyse de la phrase seront plus ou moins prépondérantes.

2.2. Quelques remarques sur les langages informatiques

Les langages de programmation jouent un rôle essentiel dans la maîtrise de la complexité. En effet, en informatique comme en mathématique, la maîtrise de la complexité passe par l'abstraction et la conceptualisation des problèmes à résoudre. Les langages de programmation permettent ainsi de définir des briques logiques conceptuelles plus faciles à manipuler que les opérations élémentaires de bas niveau disponibles sur un ordinateur. D'une certaine façon, l'évolution des langages peut être vue comme un effort continu d'abstraction visant à échapper aux contraintes de la structure des machines et à forger les outils pour écrire des programmes dont la structure reflète celle des problèmes à résoudre et non pas celle des

machines utilisées. Ce faisant, les informaticiens ont rapproché les langages qu'ils utilisent des mathématiques et de la logique même si la courte histoire de l'informatique n'a pas encore permis d'arriver au niveau d'homogénéité et de fluidité qu'a atteint le langage mathématique après quelques millénaires d'existence.

Enfin, l'utilisation de langages suffisamment abstraits a d'abord pour effet de faciliter l'écriture des programmes, leur compréhension par d'autres personnes que le programmeur ainsi que leur assemblage pour réaliser des logiciels complexes. Elle ouvre aussi la possibilité de raisonner sur les programmes, d'en définir la sémantique, de faire des preuves de correction.

Deux styles de langage sont couramment utilisés : les langages interprétés font appel à un interpréteur pour traduire les instructions du langage en instructions pour le processeur. Cette traduction se fait au fur et à mesure de l'exécution (par exemple en python, caml, prolog, javascript…).

Afin de ganer du temps ; il est également possible d'utiliser un langage compilé, dans lequel les instructions processeurs sont générées avant l'exécution du programme par un compilateur. On parle alors de code objet. Les langages concernés ont fortran, C, C++.

Dans le cas d'une calculatrice, le langage créé (dont les phrases correspondent à toutes les expressions mathématiques acceptées par la calculatrice) est relativement simple. Une **interprétation** est donc suffisante.

2.3. Grammaire formelle

Les grammaires formelles sont à la base des techniques d'analyse textuelle et sont utilisées pour résoudre des problèmes diverses de reconnaissance des formes. Plusieurs types de grammaires formelles ont été de ce fait élaborés pour répondre à des problèmes différents de reconnaissance.

Dans le cas de la réalisation d'une calculatrice, la grammaire permet de définir quelles expressions mathématiques (construites à partir du vocabulaire considéré) sont autorisées (avec quelle syntaxe).

Une grammaire G est un quadruplet : **G(terminaux, non-terminaux, productions, axiome)**, où [1]:

- terminaux : ensemble de mots formant les phrases qui ne peuvent pas être remplacés ;
- "non-terminaux" : morceaux de phrases dits de transition qui correspondent à des notions non terminales et peuvent donc être remplacés par les règles de production ;
- axiome : un des éléments non-terminaux qui est la racine (ie l'expression de base) de la grammaire ;
- productions : règles grammaticales, qui permettent de définir par quel ensemble de terminaux et non-terminaux un non-terminal peut être remplacé. Les règles ont la forme tête $\Rightarrow$ corps, où "tête" est un non-terminal et "corps" peuvent être des séquences de terminaux et non terminaux. Chaque production permet de réécrire une phrase en remplaçant le terme de gauche par le ou les termes de droite. Par exemple, la règle Expr $\Rightarrow$ Expr + Expr | Expr – Expr | Nombre permet de définir une grammaire élémentaire pour décrire la structures de phrases du type 2+1-4+5-4.

[1] http://fr.wikipedia.org/wiki/Grammaire_formelle

2.3.1. Visualisation de la grammaire : Arbre syntaxique et sémantique

L'*arbre syntaxique* (Figure 2) correspond à l'arbre construit par l'analyse syntaxique. La racine de cet arbre correspond à l'axiome de la grammaire et ses feuilles à des terminaux. Les branches intermédiaires représentent les non-terminaux.

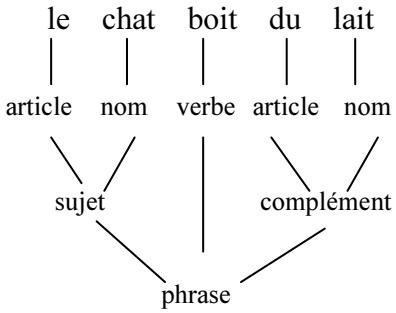


Figure 2 : *Arbre syntaxique correspondant à la phrase "le chat boit du lait"*

Cependant, par souci d'efficacité, ce n'est souvent pas un arbre syntaxique qui est construit mais un *arbre sémantique* (Fig. 3). L'arbre sémantique peut être vu comme un arbre syntaxique dont on n'a gardé que les éléments qui apportent un sens à la phrase.

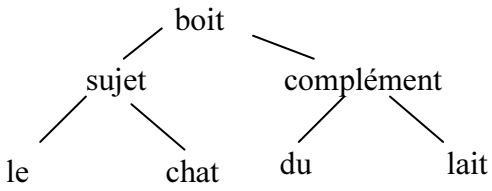
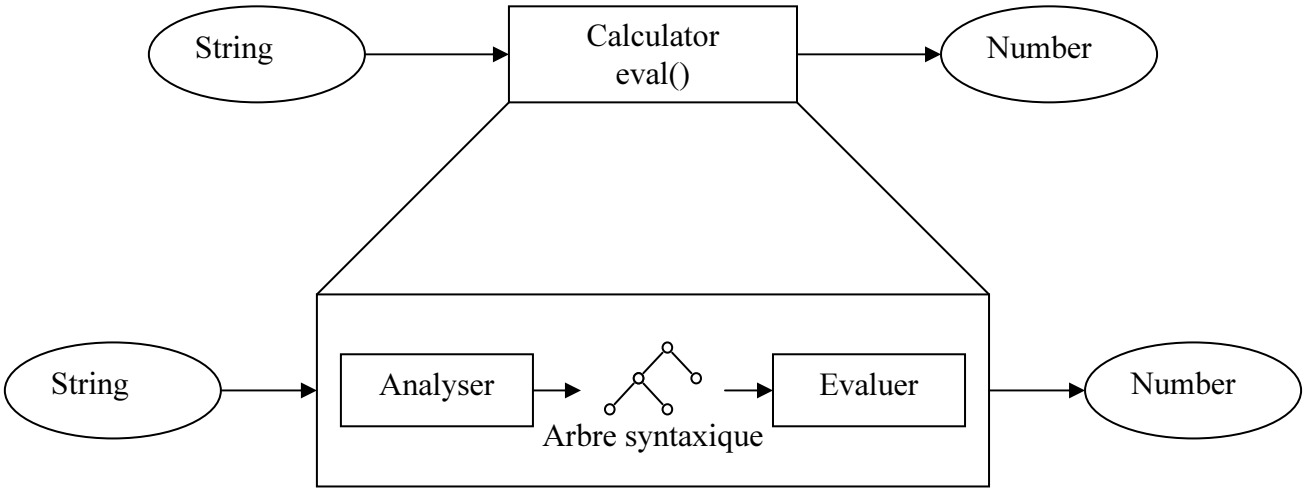


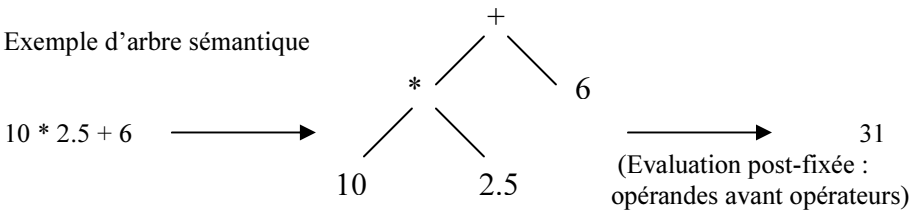
Figure 3 : *Arbre sémantique correspondant à la phrase "le chat boit du lait"*

Partie 3. Conception et design d'une calculatrice

L'évaluation d'une expression passe d'abord par l'analyse de la chaine de caractères qui génère un arbre. Celui-ci peut ensuite être passé à l'évaluateur:

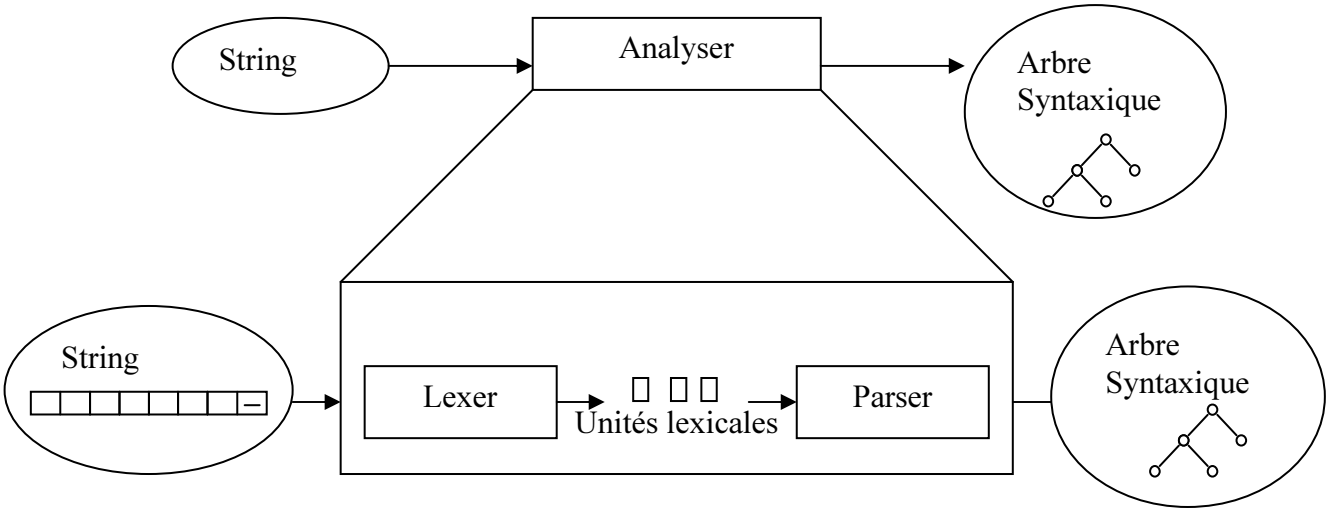


Exemple d'arbre sémantique

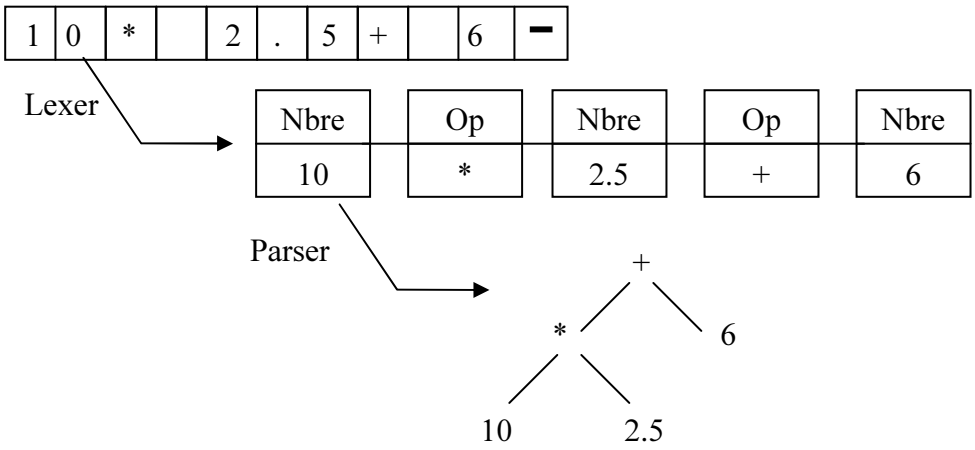


L'analyseur lui-même peut être décomposé en :

- un analyseur lexical (lexer) : qui divise la chaîne de caractère en Unités lexicales (mots) ;
- un analyseur syntaxique (parser) : qui produit l'arbre syntaxique à partir des mots et de leur signification grammaticale



Exemple :



Partie 4. Résumé de la calculatrice :

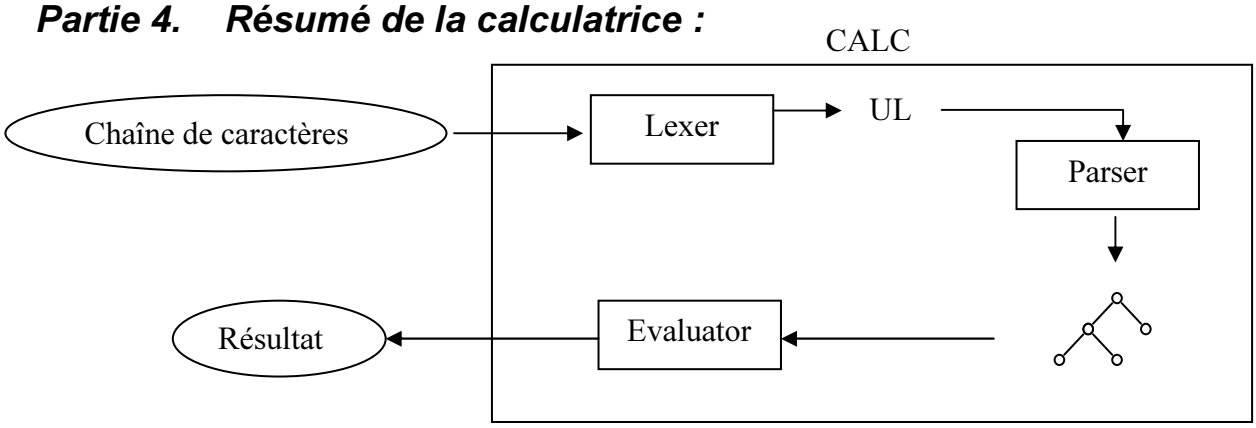


Diagramme de collaboration :