# Luigi

## Présentation

### Document concerné

**Luigi** est un langage de programmation destiné à construire une arborescence d’informations structurées sous n’importe quelle forme.

### Versions

31/05/2016 : version initiale

### Auteur

Invisible Media

## Projet traité

### Cadre du projet

### Ensemble des fonctionnalités

### Nom de la fonctionnalité traitée ici

Description du langage

## Besoin relatif

Il faut un système capable d’analyser syntaxiquement et de compiler et exécuter un exemplaire du langage **Luigi** de manière à être utilisable.

Le système réalisé est un logiciel appelé « Never ». C’est une librairie qui fournit un ensemble de fonctionnalités d’analyse syntaxique et grammaticale ainsi que de compilation et de conversion ; également, la librairie fournit un module d’analyse et d’écriture dans le langage souhaité.

## Description du langage

**Luigi** est un langage descriptif et fonctionnel. Des instructions permettent de décrire une arborescence d’informations structurées qui sont par la suite utilisable de façon fonctionnelle.

### Instructions

Quatre instructions principales :

1. set
2. mapper
3. literal
4. function

Toutes les instructions ont un nom donné à titre de nommage et d’utilisation.

Ainsi,

set first-set

mapper first-mapper

literal first-literal-id1

function myFunction

literal a\_literal

Le nom permet de les réutiliser dans le cadre de l’application de l’instruction.

Les noms sont donc associés en tant que « nom de type ». Lorsqu’un nom est utilisé plusieurs fois comme nom de type, le plus récent remplace l’ancien même si l’instruction a changée.

Par exemple,

set first-set

mapper first-set

set first-set

literal first-set

Le « nom de type » first-set est maintenant un literal.

#### Instruction set

Cette instruction construit une arborescence d’informations structurées. A partir de cette arborescence, une commande lui est adjointe permettant de convertir l’arborescence en un texte brut.

L’objectif de cette instruction est de proposer un ensemble de « noms de type » capable d’être appliqués de manière dynamique et souple à plusieurs sortes d’informations.

Chaque instruction a une commande permettant de convertir l’information structurée en un texte brut.

L’instruction est polymorphe, c’est-à-dire que les informations fournies peuvent être interprétées de manière différente selon le texte brut souhaité.

Aussi, pour une même arborescence d’informations structurées, la définition de chacune des instructions peut être remplacée en remplaçant la commande de conversion des informations en texte brut.

##### Conclusion

L’instruction set est une interface sémantique ; la commande associée à l’instruction est l’objet polymorphe.

##### Syntaxe

L’instruction set accepte un nombre indéterminé de paramètres. Chaque paramètre déclaré est typé si c’est un literal, un mapper ou un autre set. Des valeurs par défaut pour chaque paramètre peuvent être précisées.

set [nom de type] {

-name1 literal

%name2 mapper

@name3 set

}

Les paramètres fournis sont typés selon un caractère devant le nom du paramètre : - pour literal, % pour mapper, @ pour set.

#### Instruction mapper

Cette instruction associe une clé avec un literal. La déclaration offre la possibilité d’exprimer un nombre indéterminé de clés. Chaque clé est unique ; dans le cas contraire, la clé nommée une seconde fois met à jour sa valeur.

L’utilisation d’un mapper offre la possibilité d’exprimer la clé ce qui se traduit par l’écriture à l’exécution du literal associé à la clé.

##### Conclusion

L’instruction nomme des éléments « génériques » et offre la possibilité d’en changer leur expression finale.

##### Syntaxe

L’instruction accepte un nombre indéterminé de clés. A chaque clé, l’instruction associe un literal.

mapper [nom de type] {

a => literal1

b => literal2

…

}

#### Instruction literal

Cette instruction associe un nom de type avec un texte brut.

Des paramètres d’entrée peuvent être définis nommés et séparés par une virgule. Des espaces, tabulations ou des retours-chariots peuvent éventuellement être ajoutés. Cependant, l’édition du code les supprimera ou en ajoutera.

Pour utiliser ces paramètres dans le texte, il suffit de les faire précéder du caractère apostrophe suivi d’un espace ou d’une deuxième apostrophe. Le caractère apostrophe seul ou doublé peut être utilisé pour le texte.

##### Syntaxe

literal [nom de type] [mot de fin] [texte][mot de fin]

literal [nom de type](params…) [mot de fin] [texte][mot de fin]

#### Instruction function

Cette instruction a pour objectif de grouper plusieurs instructions en une seule. Une fonction comporte un nombre indéterminé de paramètres formels nommés. Les instructions groupées sont appliquées au moment de son utilisation en appliquant les paramètres passés par valeur. Cependant, la déclaration de la fonction doit être correctement formée.

##### Syntaxe

function [nom de type](params…) {

…

}

### Affectations

Comme pour tout langage de programmation, des variables sont applicables. Une variable est un moyen de structurer et dynamiser le résultat attendu.

L’affectation d’une variable est obtenue par l’opérateur =

[nom de variable] = [instruction]

Une fois l’affectation déclarée, ce qui suit peut réutiliser cette variable. La variable ne contient sans doute pas l’instruction mais uniquement son résultat ; c’est également dans cette circonstance qu’une variable utilisée est interprétée immédiatement et non à postériori.

#### Portée des variables

Les variables sont créées successivement tout au long de la séquence du traitement des opérations. Exception, pour les fonctions, la portée des variables est acheminée par les paramètres et, de ce fait, toute variable créée dans une fonction constitue un paramètre pour cette fonction. Aussi, une fonction dans une autre, ne verra jamais les paramètres de la fonction parente. Pour communiquer un paramètre en entrée à une fonction, il suffit de fournir le paramètre.

Nous pouvons éventuellement considérer qu’une fonction possède l’ensemble des variables qu’elle utilise pendant son traitement comme des paramètres. Autrement dit, les variables utilisées sont acceptées en tant que paramètre même si elles ne sont pas présentes dans la liste des paramètres au moment de la construction de la fonction. La présente liste des données en entrée en tant que paramètre est ici, fait pour faciliter la lecture du code et non pas pour permettre au programme de fonctionner.

### Commande de conversion

La commande de conversion est l’exploitation des informations structurées sous la forme de l’écriture d’un texte brut.

Les nombres, les opérations usuelles, les valeurs booléennes et les expressions booléennes sont prise en compte.

#### Ensemble des commandes

La syntaxe des commandes est la suivante

[nom de la commande]([paramètre], …)

##### Concat

Cette commande réalise la concaténation ordonnée des paramètres partant de la gauche vers la droite. Le nombre de paramètres est indéfini. Chaque paramètre est séparé par une virgule. Les paramètres sont effectifs ; ce sont des valeurs qui sont représentées dans chaque paramètre.

##### String

Cette commande utilise la même syntaxe qu’un literal.

##### Space

La commande space affiche autant d’espaces que le nombre indiqué en paramètre.

##### Return

La commande return affiche autant de retours-chariots que le nombre indiqué en paramètre.

##### Indent

Cette commande indente le texte. Deux commandes sous-jacentes lui sont attribuées

Indent++++

Indent----

##### Chars

Cette commande prend deux paramètres : un nombre et une séquence de caractères. La séquence de caractère doit faire exactement le même nombre que celui donné.

##### If

Elle est composée de trois termes :

1. une expression booléenne
2. une commande
3. une autre commande

##### Loop

Elle est composée de quatre termes :

1. un iterateur
2. une commande

### Méthodes

Chaque instruction possède des méthodes.

#### Print

La méthode print() affiche le texte brut. Elle peut comporter un paramètre qui indique le nom du fichier ou une variable pointeur.

Cette méthode est applicable aux instructions set, mapper, literal et function.

#### insert, remove, select

L’instruction set est une instruction à plusieurs instances.

#### Paramètres

Les paramètres nommés peuvent être utilisés pour accéder à un élément dans l’ensemble arborescent des informations structurées.

## Exemples

### Exemple 1 : structures d’opérations arithmétiques

## Implémentation

J’utilise un sous-programme pour effectuer les opérations. Le sous-programme s’appelle **Accu**.

C’est un langage basé sur un accumulateur. Une arborescence fonctionne selon des variables définies et des sous-listes de variables.

L’utilisation de ce sous-programme est pratique pour créer rapidement des éléments d’informations structurés et arborescente. C’est pourquoi **Accu** est adapté pour le langage **Luigi**.

J’utilise également un autre sous-programme appelé **Printer**. C’est un sous-programme permettant de personnaliser du texte en utilisant un ensemble de variables de configuration applicable en totalité du contenu et un ensemble de variables simples ou complexes ; les variables simples sont des couples (nom, valeur) alphanumériques. Les variables complexes acceptent un nom de fichier de type **Printer** pour réutiliser des commandes personnalisées existantes.

Le langage **Printer** et le langage **Accu** sont deux étapes déterministes qui simplifient la réalisation du résultat et la réalisation du code source visible.

Utiliser le langage **Accu** pour réaliser l’objectif de traitement du langage **Luigi** est rendu possible par la propriété *Value* du langage. C’est cette propriété qui permet de considérer chaque commande du langage Luigi comme une expression.

Puisque le langage **Accu** réalise une arborescence, le paramétrage au moment de l’application des valeurs effectives et des valeurs formelles est rapidement mis en place sans alourdir la programmation.

Deux options complètent ce langage : une référence et un appel de méthode. La référence permet de citer des noms pour les réutiliser. L’appel de méthode se propose d’associer un nom et un ensemble de paramètres dont la réalisation de la méthode utilise les paramètres pour effectuer les opérations.