

# **TLC-S7: Documentation utilisateur**

Pour le 26/01/2025

*Responsable: LAMARCHE Fabrice*

## Sommaire

1. Ecriture d'un programme en langage WHILE .....	3
1.1. Brève introduction .....	3
1.2. Types de données .....	3
1.2.1. Arbres binaires .....	3
1.2.2. Simulation de types .....	3
1.3. Commandes et structures de Contrôle .....	3
1.3.1. Commandes simples .....	3
1.3.2. Structures conditionnelles .....	3
1.3.3. Boucles .....	4
1.3.4. Définitions de fonctions .....	4
1.3.5. Programme principal .....	5
2. Compilation d'un programme en langage WHILE .....	6

# 1. Ecriture d'un programme en langage WHILE

## 1.1. Brève introduction

Le langage While est conçu pour manipuler uniquement des arbres binaires. Il permet de simuler d'autres types de données (entiers, booléens, chaînes de caractères etc.) grâce à des encodages spécifiques.

## 1.2. Types de données

### 1.2.1. Arbres binaires

Comme dit en introduction, le langage While manipule uniquement des arbres binaires. Voici les éléments principaux pour travailler avec ce type de données :

- `nil` : Arbre vide.
- `(cons)` : Equivalent à `nil`
- `(cons A)` : Crée une copie de `A`
- `(cons A B)` : Crée un arbre binaire avec `A` comme fils gauche et `B` comme fils droit.
- `(cons A B C)` : Equivalent à `(cons A (cons B C))`
- `(list T1 T2 ... Tn)` : Crée une liste chaînée d'éléments.

### 1.2.2. Simulation de types

- Booléens : `true` qui est encodé comme tout sauf un arbre sans enfants (exemple : `(cons nil nil)`) et `false` qui est encodé comme `nil`.
- Entiers : Un entier `n >= 0` est encodé comme un arbre de `n-1` nœuds à droite. Les nœuds à gauche ne sont pas comptabilisés.

Exemple :

- `nil = 0`
- `(cons nil nil) = 1`
- `(cons (cons nil nil) nil) = 1` ( car seul les nœuds à droite comptent )
- `(cons nil (cons nil nil)) = 2`
- `(cons nil nil nil) = 2`

- Chaînes de caractères : Encodées comme un arbre où chaque nœud possède une chaîne de caractères. L'interprétation de l'arbre se fait alors comme la concaténation du `fils gauche + chaîne du nœud actuel + fils droit`

## 1.3. Commandes et structures de Contrôle

Voici les principales commandes et structures de contrôle :

### 1.3.1. Commandes simples

- `nop` : Ne fait rien (utile pour tester).
- `Vars := Exprs` : Associe des expressions à des variables.

### 1.3.2. Structures conditionnelles

- `if E then C1 fi` : Exécute `C1` si `E` est vrai.
- `if E then C1 else C2 fi` : Exécute `C1` si `E` est vrai, sinon `C2`.

### 1.3.3. Boucles

- `while E do C od` : Répète `C` tant que `E` est vrai.
- `for E do C od` : Répète `C` un nombre déterminé de fois.
- `foreach X in E do C od` : Parcourt chaque élément de `E`.

### 1.3.4. Définitions de fonctions

Un programme While est composé de plusieurs fonctions. Voici la grammaire générale d'une fonction en While pour comprendre la syntaxe :

```
Program → Function Program | Function
Function → 'function' Symbol ':' Definition
Definition → 'read' Input '%' Commands '%' 'write' Output
```

Avec :

- `Input` : Liste des variables d'entrée
- `Commands` : Liste de commandes ou de structures de contrôle
- `Output` : Liste des variables de sortie

Par exemple, voici une fonction While “générique” décrivant bien les spécificités de la grammaire.

```
function Symbol :
  read I1, ..., In
%
  Commands
%
  write O1, ..., Om
```

- `Symbol` : Nom de la fonction.
- `I1, ..., In` : Paramètres d'entrée.
- `Commands` : Commandes exécutées.
- `O1, ..., Om` : Valeurs de sortie.

#### 1.3.4.1. Exemple : Fonction pour ajouter deux entiers

```
function add :
  read A, B
%
  if A =? nil then
    Result := B
  else
    Result := (cons nil (add (tl A) B))
  fi
%
  write Result
```

*Remarques :*

- Une fonction peut avoir zéro paramètre d'entrée.
- Elle doit toujours avoir au moins une valeur de sortie.

- Les variables sont locales à la fonction.

### 1.3.5. Programme principal

Le programme principal est défini dans une fonction appelée `main`. C'est la fonction principale. Voici un exemple :

```
function main :  
  read X, T  
  %  
  Result := (cons int (add X Y))  
  %  
  write Result
```

- Entrées : `X`, `Y`.
- Commandes :
  - Appel de la fonction `add` avec `X` et `Y`.
  - Stockage du résultat de la fonction dans `Result` avec un pretty printing en int
- Sortie : `Result`.

## 2. Compilation d'un programme en langage WHILE

Pour utiliser notre compilateur pour générer, à partir d'un code source WHILE, un fichier exécutable, voici les commandes à exécuter :

Pour Linux et MacOS :

```
java -jar <path-to-while_compiler-all.jar> -i <path-to-while-file> -o <target-cpp-file>
clang++ <target-cpp-file>
./a.out <args>
```

Pour Windows :

```
java -jar <path-to-while_compiler-all.jar> -i <path-to-while-file> -o <target-cpp-file>
clang++ <target-cpp-file>
./a.exe <args>
```

⚠ : Seuls les arguments sous forme d'entier sont supportés