INTERPRETEUR DE LANGAGE PSEUDO-CODE

Langage accepté par l'interpréteur

L'interpréteur simule l'exécution d'un langage de programmation simple avec des constructions classiques telles que les affectations, les conditions, les boucles et les entrées/sorties. Voici les principales caractéristiques du langage supporté :

1. Affectation : Syntaxe : Variable ← Expression

Exemple : X ← 3; affecte la valeur 3 à la variable X.

2. Opérations arithmétiques :

- Le langage permet des opérations de base : addition (`+`), soustraction (`-`), multiplication (`*`), division (`/`), et modulo (`%`).

- Exemple : `X ← 3 + 2;` affecte la valeur `5` à `X`.

3. Entrée : Syntaxe : Lire(Variable)

Exemple : Lire(Z); lit une valeur de type `double` et l'affecte à la variable `Z`.

4. Sortie : Syntaxe : Ecrire(Expression)

Exemple : Ecrire(X + 2) affiche le résultat de l'expression `X + 2`.

5. Conditions (Si) : Syntaxe : Si(Condition) { Instructions si vrai } Sinon {Instructions si faux}

Exemple : Si(X < 5) { Y ← X + 1; } Sinon { Y ← X - 1; }

6. Boucles (Tant que) : Syntaxe : Tq(Condition) { Instructions }

Exemple : Tq(X < 5) { X ← X + 1; }

7. Opérations booléennes :

- Le langage permet aussi d'utiliser des opérations booléennes : `&&` (ET), `||` (OU), et `!` (NON).

8. Commentaires :

- Il est aussi possible de rajouter des commentaires avec # et le commentaire jusqu'à la fin de ligne

Traitement effectué par l'interpréteur

L'interpréteur évalue et exécute les instructions du programme ligne par ligne. Il :

- Évalue les expressions arithmétiques et booléennes en utilisant les opérateurs standards.

- Gère les variables en utilisant une table des symboles (`ST_ARBRE`) qui garde une trace des valeurs affectées aux variables.

- Exécute les blocs conditionnels (`Si`) et répète les blocs d'instructions dans les boucles (`Tq`) tant que les conditions sont vraies.

- Affiche les résultats des opérations via la commande `Ecrire` et permet à l'utilisateur de saisir des valeurs via la commande `Lire`.

L'interpréteur s'assure également qu'aucune boucle infinie ne se produise en limitant le nombre d'itérations d'une boucle et en affichant un message d'erreur si le programme entre dans une boucle infinie.

Alphabet :

[0-9] ; [a-zA-Z] ; et [+, -, *, /, _]

LEXEMES :

MOT, // séquence de caractères

ENTIER, // séquence de chiffres

PLUS, MOINS, MUL, DIV, MODULO // +, -, *, / et %

INFERIEUR, SUPERIEUR, EGAL, NEGATION // <, >, = et !

ASSIGNATION // <-

PARO, PARF, ACCO, ACCF, // (,), { et }

SI, ALORS, SINON, TQ, ECRIRE, LIRE, DIESE

FIN_SEQUENCE // pseudo lexème ajoute en fin de séquence

Sémantique :

Pas de division par 0.

Prog → seq_inst

seq_inst → inst suite_seq_inst

suite_seq_inst → SEPINST seq_inst | ε

inst → seq_aff | lecture | ecrire | si | tq | eag

seq_aff → aff seq_aff | ε
aff → CHAINE AFF eag

lecture → LIRE PARO CHAINE PARF
ecrire → ECRIRE PARO eag PARF

si → SI PARO condition PARF ACCO seq_inst ACCF si_sinon
si_sinon → SINON ACCO seq_inst ACCF | ε

tq → TQ PARO condition PARF ACCO seq_inst ACCF

condition → PARO op_bool suite_condition PARF
suite_condition → ET op_bool suite_condition | OU op_bool suite_condition | ε

op_bool → eag OP_COMP eag | NON PARO eag OP_COMP eag PARF

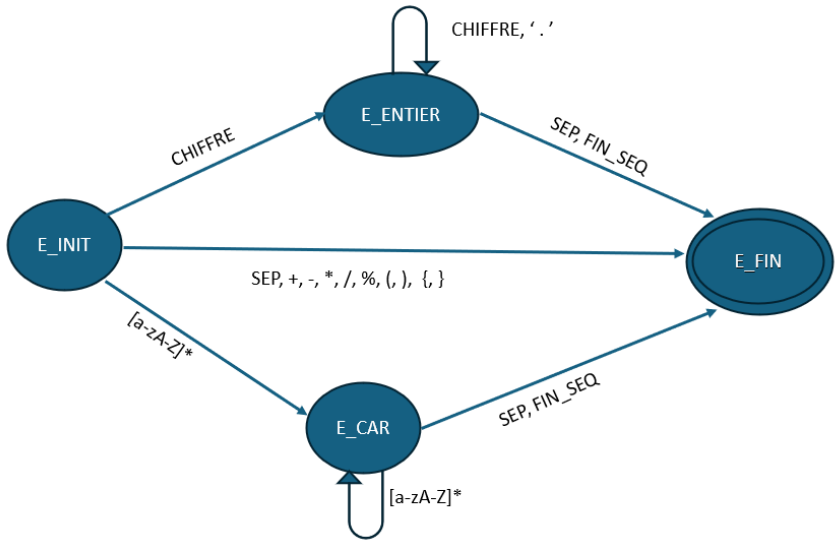
eag → seq_terme

seq_terme → terme suite_seq_terme
suite_seq_terme → op1 terme suite_seq_terme | ε

terme → seq_facteur
seq_facteur → facteur suite_seq_facteur
suite_seq_facteur → op2 facteur suite_seq_facteur | ε

facteur → ENTIER | CHAINE | PARO eag PARF
op1 → PLUS | MOINS
op2 → MUL | DIV | MODULO
op → PLUS | MOINS | MUL | DIV | MODULO
eaep → ENTIER | PARO eaep op eaep PARF
OP_COMP → SUP | INF | EQ | NEQ | SEQ | IEQ

Automate pour le Lexème :



Exemple :

```
1   # Ceci est un commentaire
2
3   Ecrire(7 + 8);
4
5   X <- 45;
6
7   Y <- X -20;
8
9   Si(X != Y){
10      Si(X < Y){
11          Ecrire(X);
12      } Sinon{
13          Lire(Z);
14          Ecrire(Z);
15      }
16  }
17
```

```
---------------------------- Analyse du PROGRAMME:----------------------------
Pas d'erreur! ✅
Pas d'erreur! ✅
Pas d'erreur! ✅
Pas d'erreur! ✅

---------------------------- Analyse terminee ----------------------------
Pas d'erreur! ✅

-----------------------Interpretation du PROGRAMME:-----------------------
Ecrire((7.000000+8.000000));
valeur = 15.000000

Entree la valeur a lire: 1
Z = 1.000000
Ecrire(Z);
valeur = 1.000000


-----------------------Interpretation terminee -----------------------
Pas d'erreur! ✅
```