Chapitre IV.
Analyse syntaxique descendante

a.hettab@centre-univ-mila.dz

Introduction

- Dans **l'analyse syntaxique descendante**, l'arbre syntaxique est construit depuis la racine puis en descendant dans l'arbre.
- **Les méthodes descendantes** sont **simples à implémenter**, mais la classe des grammaires non contextuelles, avec lesquelles on peut construire ces analyseurs, **n'est pas très grande**.
- Nous allons étudier les deux version de la méthode d'analyse prédictive **LL** :
  - **l'analyse syntaxique LL(1).**
  - **l'analyse par descente récursive.**

Analyse syntaxique LL

- **l'analyse LL** est une **analyse syntaxique descendante** pour certaines grammaires non contextuelles, dites **grammaires LL**.
- **L'analyse LL** analyse un mot d'entrée **de gauche à droite** (**Left to right** : c'est **le premier L de LL**) et en construit une **dérivation à gauche** (**Leftmost derivation** : c'est **le deuxième L de LL**).

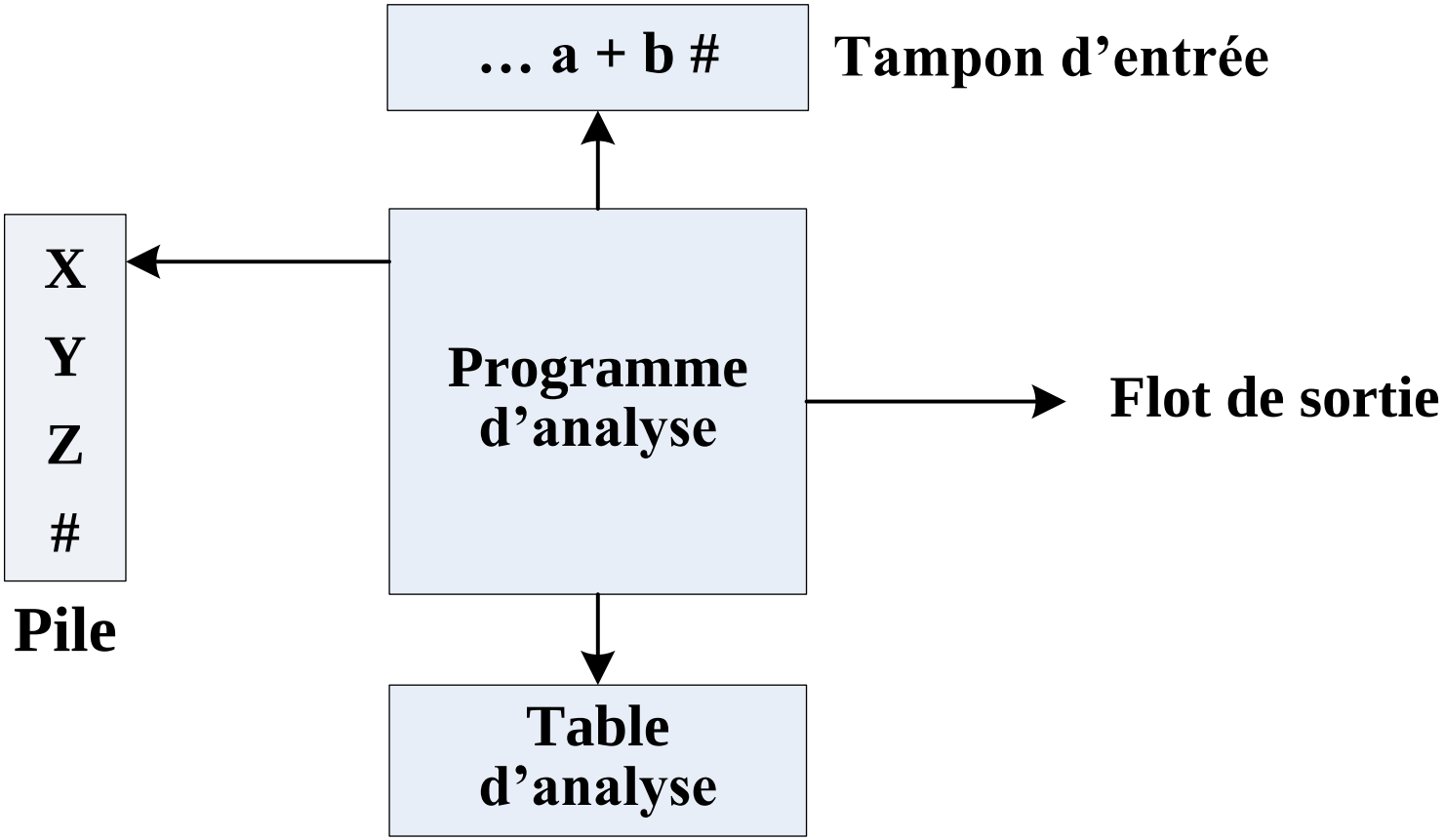
Analyse syntaxique LL(1)

- **Une analyse LL** est appelée analyse **analyse LL(1)** lorsqu'elle réalise une seule passe sur le mot d'entrée.
- **Une analyse LL** est appelée **analyse LL(*k*)** lorsqu'elle réalise ***k*** passes sur le mot d'entrée.
- Le nombre ***k*** est dans la majorité des cas égal à **1**, car pour **k>1** l'analyse devient **moins intéressante**. C'est pour cette raison que nous utiliseront uniquement l'analyse **LL(1)** dans la suite du cours.

Architecture d'un analyseur LL(1)

- **L'analyseur LL(1) est composé de :**
- un **tampon d'entrée**, contenant la chaîne de caractères à analyser et doté de deux opérations: **lecture du caractère courant** et **passage au caractère suivant** ;
- une **pile** sur laquelle poser les **terminaux** et **non terminaux** de la grammaire qui restent à analyser ;
- une **table d'analyse** qui dit quelle règle utiliser (s'il y en a une) en fonction **des symboles au sommet de la pile** et du **caractère suivant**.

Architecture d'un analyseur LL(1)



Construction de la table d'analyse LL(1)

- Soit ***G=<N,T,S,P>*** une grammaire non contextuelle (où ***N*** désigne l'ensemble des variables ou **symboles non terminaux**, ***T*** l'alphabet terminal ou ensemble des **symboles terminaux**, ***P*** l'ensemble des **règles**, ***S*** **l'axiome** de la grammaire).
- Afin de calculer **la table d'analyse**, on introduit les deux fonctions: ***Premier*** et ***Suivant*** (***First*** et ***Follow*** en anglas).

fonctions: *Premier* et *Suivant*

Premier

Fonction Premier:

- Pour toute expression $\alpha \in (T \cup N)^+$, on définit ***Premier*** (α) comme étant l'ensemble des **terminaux susceptibles de commencer** un mot **dérivé** de **α**.
- Plus formellement :
  - $Premier(\alpha) = \{a \in T \mid \exists \beta \in (T \cup N)^*, \alpha \Rightarrow^* a\,\beta\}$.
  - Si $\alpha \Rightarrow^* \varepsilon$ alors $\varepsilon \in Premier(\alpha)$.
  - Si $\alpha = \varepsilon$ alors $Premier(\alpha) = \Phi$.

Calcul de l'ensemble Premier

Premier(X)

i) SI **X** est un terminal ALORS **Premier (X)=X**;

ii) SI **$X \rightarrow \varepsilon$** ALORS ajouter **ε à Premier (X)** ;

iii) SI **$X \rightarrow Y_1 Y_2 \ldots Y_n$** ALORS

- SI **$\varepsilon \notin$ Premier (Y1)**
  ALORS on ajoute uniquement **Premier (Y_1) à Premier (X)**;
- SI **$\varepsilon \in$ Premier (Y_1),…, Premier(Y_{i-1})** avec **$2 \leq i \leq n$**
  ALORS ajouter **Premier (Y_1),…, Premier(Y_{i-1}), Premier(Y_i) ε exclu à Premier (X)**
- SI **$\varepsilon \in$ Premier (Y_1), Premier(Y_2),…, Premier(Y_n)**
  ALORS ajouter **ε à Premier (X)**

Remarque :

- Ces règles sont appliquées jusqu'à ce qu'aucun terminal ni **ε** ne puisse être ajouté aux ensembles **Premier**.

fonctions: *Premier* et *Suivant*

Suivant

Fonction Suivant :

- Pour toute expression $\alpha \in (T \cup N)^+$, on définit ***Suivant*** (α) comme étant l'ensemble des ***terminaux*** **susceptibles** de **suivre** un mot **dérivé** de α.
- Plus formellement :
  - $Suivant(\alpha) = \{a \in Premier(\gamma) \mid \exists \beta, \gamma \in (T \cup N)^*, S \Rightarrow^* \beta \alpha \gamma\}$.
  - $Si\ \alpha = \varepsilon\ alors\ Suivant(\alpha) = \Phi$.
  - On ajoute aussi le symbole '#' à tous les $\alpha \in (T \cup N)^+ \mid \exists \beta \in (T \cup N)^*, S \Rightarrow^* \beta \alpha$, de façon à pouvoir indiquer la fin de la chaine d'entrée.

Calcul de l'ensemble SUIVANT

SUIVANT(X)

i) Mettre # (qui est le marqueur de fin de l'entrée à analyser) dans **SUIVANT(S)** où **S** est l'axiome de la grammaire;

ii) S'il y a une production $A \rightarrow \alpha X \beta$

ALORS ajouter **PREMIER(β) sauf ε à SUIVANT(X)** ;

iii) S'il y a une production $A \rightarrow \alpha X$ ou une production $A \rightarrow \alpha X \beta$ avec $\varepsilon \in PREMIER(\beta)$

ALORS ajouter **SUIVANT(A) à SUIVANT(X)**;

Remarque :

- Ces règles sont appliquées jusqu'à ce qu'aucun terminal ni # ne puisse être ajouté aux ensembles **SUIVANT**.

fonctions: *Premier* et *Suivant* *Exemples*

Exemple 1:

- Calculer les ensembles **Premier** et **Suivant** pour les non-terminaux de la grammaire dont les productions sont données ci-après :

S → aSBA | ε

A → aSb | b

B → bB | ε

	Premier	Suivant
S	a ε	# a b
A	a b	# a b
B	b ε	a b

fonctions: *Premier* et *Suivant* *Exemples*

Exemple2:

- Calculer les ensembles **Premier** et **Suivant** pour les non-terminaux de la grammaire dont les productions sont données ci-après :

S → ABSb | ε

A → aBb | b

B → bB | cS | ε

	Premier	**Suivant**
S	a b ε	# a b
A	a b	b c a
B	b c ε	a b

Remplissage de la table d'analyse

- La **table d'analyse** est une **matrice à deux dimensions**, dont les **lignes** sont indicées par des ***Non-terminaux*** et les **colonnes** par **des *Terminaux***.
- **Donnée :** Une grammaire non-contextuelle **G**.
- **Résultat :** Une table d'analyse **M** pour **G**;
- Les indices des lignes de **M** sont **les non-terminaux** de la grammaire ;
- les indices des colonnes sont **les terminaux** de la grammaire plus le caractère spécial # (# est le marqueur de fin du flot à analyser).

Remplissage de la table d'analyse Algorithme

Pour toute règle de la forme **X→α** **Faire**

  Pour tout **a ∈ Premier(α)** **Faire**

    Ajouter X→α à la case d'indice **M[X, a]**

    Si ε ∈ Premier(α) **Alors**

      Pour tout **b ∈ Suivant(X)** **Faire**

        Ajouter X→α à la case d'indice **M[X, b]**

      Fin pour

    Fin Si

  Fin pour

Fin pour

Grammaires LL(1)?

- Une **grammaire** est **LL(1)** lorsque **sa table d'analyse ne comporte pas plusieurs règles de production** pour une **même case**. Dans ce cas la table d'analyse est dite **mono-définie.** Dans le cas contraire la table d'analyse est dite **multi-définie.**
- Une grammaire **LL(1)** permet de faire une **analyse syntaxique descendante déterministe.** Au cours de toute dérivation, il existera au plus une possibilité pour remplacer un **non terminal** par une de ses **parties droites** selon le **caractère courant** de l'entrée à analyser.

Grammaires LL(1)?

- **Sans construire** de la **table d'analyse**, on peut dire qu'une grammaire est **LL(1)** ou **non**, si elle a **les propriétés suivantes**:
- S'il existe deux productions $A \rightarrow \alpha$ et $A \rightarrow \beta$ avec $(\alpha \neq \beta)$ dans une grammaire **LL(1)**, on a:
  - Il **n'existe** pas de terminal a tel que **α** et **β** génèrent tous deux des chaînes qui commencent par a;
  - Au plus un de **α** et **β** peut générer la chaîne **ε**;
  - Si **β ⇒* ε** alors **α ne peut pas générer** une chaîne dont le premier terminal est un élément de **Suivant(A)**.

Grammaires LL(1)?

Remarque :

- Les trois conditions précédentes peuvent être résumées par la formule ci-après.

✓ Une grammaire **G=<N,T,S,P>** est **LL(1)** si et seulement si pour toute paire de règles $A \rightarrow \alpha \mid \beta$ on a :

$$\text{Premier}(\alpha.\text{SUIVANT}(A)) \cap \text{Premier}(\beta.\text{SUIVANT}(A)) = \varnothing$$

Algorithme d'analyse

Initialement la pile contient # et au dessus **l'axiome** de la grammaire;

répéter Soit **X** le symbole en **sommet de pile** et Soit **a** le symbole d'entrée courant

Si **X** est un **non terminal** **alors**

Si **M[X, a]= X → Y1 ….Yn** **alors**

Dépiler **X** de la pile et **Empiler** **Yn** **puis** **Yn-1**

puis…… puis **Y1** dans la pile

Sinon /*case vide dans la table*/

Ecrire (‘**ERREUR**’)

Finsi

Sinon /*X est un terminal*/

(1) **(2)**

Algorithme d'analyse

(1) (2)

Si X = # **alors**

  Si a = # **alors**

    Ecrire (‘**ACCEPTER**’)

  Sinon

    Ecrire (‘**Erreur**’)

  Fin si

Sinon /*X est un terminal ≠ #*/

  Si X = a **alors**

    Dépiler X de la pile et **Avancer la lecture** sur le symbole suivant.

  Sinon /*X est un terminal ≠ a*/

    Ecrire (‘**Erreur**’)

  Finsi

Finsi

Finsi

Fin Répéter

Exemple d'analyse LL(1)

- Considérer la grammaire **G = <{E,E',T,T',F},{+,*,(,),id},P,E>**:

E → TE'
E' → +TE' | ε
T → FT'
T' →* FT' | ε
F → (E) | i

✓ Ensembles Premier et SUIVANT :

	Premier	SUIVANT
E	(i	#)
E'	+ ε	#)
T	(i	+ #)
T'	* ε	+ #)
F	(i	* + #)

Exemple d'analyse LL(1)

- Table d'analyse pour la grammaire précédente :

$E \rightarrow TE'$
$E' \rightarrow +TE' \mid \varepsilon$
$T \rightarrow FT'$
$T' \rightarrow * FT' \mid \varepsilon$
$F \rightarrow (E) \mid i$

	+	*	(	)	I	#
E			$E \rightarrow TE'$		$E \rightarrow TE'$	
E'	$E' \rightarrow +TE'$			$E' \rightarrow \varepsilon$		$E' \rightarrow \varepsilon$
T			$T \rightarrow FT'$		$T \rightarrow FT'$	
T'	$T' \rightarrow \varepsilon$	$T' \rightarrow * FT'$		$T' \rightarrow \varepsilon$		$T' \rightarrow \varepsilon$
F			$F \rightarrow (E)$		$F \rightarrow i$	

La grammaire précédente est LL(1)

Exemple d'analyse LL(1)

- Analyse de la chaine **i + i * i #**

Contenu de la pile	Restant à analyser	Action
#E	i + i * i #	Dépiler(E) et Empiler(E'T)
#E'T	i + i * i #	Dépiler(T) et Empiler(T'F)
#E'T'F	i + i * i #	Dépiler(F) et Empiler(i)
#E'T'i	i + i * i #	Avancer
#E'T'	+ i * i #	Dépiler(T‘)
#E'	+ i * i #	Dépiler(E') et Empiler(E'T+)
#E'T+	+ i * i #	Avancer
#E'T	i* i #	Dépiler(T) et Empiler(T'F)
#E'T'F	i * i #	Dépiler(F) et Empiler(i)
#E'T'i	i * i #	Avancer
#E'T'	* i #	Dépiler(T') et Empiler(T'F*)
#E'T'F*	* i #	Avancer
#E'T'F	i #	Dépiler(F) et Empiler(i)
#E'T'i	i #	Avancer
#E'T'	#	Dépiler(T‘)
#E'	#	Dépiler(E‘)
#	#	« Chaine acceptée »

Transformation d'une grammaire

- **Théorème** : Une grammaire **ambiguë**, ou **récursive à gauche** ou **non factorisée à gauche**, **n'est pas** une **grammaire LL(1)**.
- il **n'existe** malheureusement **pas** de méthode **pour rendre une grammaire LL(1)**:
  - **une grammaire ambiguë** est une grammaire **mal** conçue. Il faut tenter de refaire la conception de la grammaire (tenir compte de la priorité des opérateurs, associer le premier else au dernier if…).
  - **Eliminer la récursivité à gauche** si cela est nécessaire.
  - **Factoriser les règles de production à gauche** si cela est nécessaire.

Factorisation à gauche d'une grammaire

- Des productions non factorisées à gauche d'une grammaire du type:

$$A \rightarrow \alpha\beta_1 \mid \alpha\beta_2 \mid \ldots \mid \alpha\beta_n \mid \gamma \text{ où } \alpha \neq \varepsilon$$

- Sont remplacées par les productions suivantes:

$$A \rightarrow \alpha A' \mid \gamma$$
$$A' \rightarrow \beta_1 \mid \beta_2 \mid \ldots \mid \beta_n$$

Factorisation à gauche d'une grammaire

- **Exemple**:

Soit la grammaire G1 ayant les règles de production suivantes :

S → aC | abD | aBc

Après factorisation à gauche, on obtient une grammaire G'1 équivalente à G1, dont les règles de production sont :

S → aS'

S' → C | bD | Bc

Elimination de la récursivité à gauche d'une grammaire

- Une grammaire est dite **récursive à gauche** si elle contient un non-terminal **A** tel que:

$$A \Rightarrow^{*} A\alpha$$

  - Où α est une chaine quelconque.
- **Elimination de la récursivité à gauche immédiate :**
- Les règles sous la forme suivante :

$$A \rightarrow A\alpha_1 | A\alpha_2 | \dots | A\alpha_m | \beta_1 | \beta_2 | \dots | \beta_n$$

  - où β ne commence pas par **A**.
  - sont remplacées par les productions :

$$A \rightarrow \beta_1 A' | \beta_2 A' | \dots | \beta_n A'$$

$$A' \rightarrow \alpha_1 A' \mid \alpha_2 A' \mid \dots \mid \alpha_m A' \mid \varepsilon$$

Elimination de la récursivité à gauche d'une grammaire

- **Exemple:**

Soit la grammaire G1 ayant les règles de production suivantes :

E → E+T | T

T → T*F | F

F → (E) | id

Après l'élimination de la récursivité à gauche, on obtient une grammaire G'1 équivalente à G1, dont les règles de production sont :

E → TE'

E' → +TE' | ε

T → FT'

T' → *FT' | ε

F → (E) | id

Elimination de la récursivité à gauche indirecte

Grammaire sans cycle (dérivations $A \Rightarrow^{+} A$) et sans production vide ($A \rightarrow \varepsilon$)

Ordonner les non-terminaux $A_1, A_2, \ldots, A_n$;

Pour i:=1 **à** n **Faire**

  Pour j:=1 **à** i-1 **Faire**

    Remplacer chaque production de la forme $A_i \rightarrow A_j\gamma$

    Par les productions $A_i \rightarrow \delta_1\gamma \mid \delta_2\gamma \mid \ldots \delta_k\gamma$

    Où $A_j \rightarrow \delta_1 \mid \delta_2 \mid \ldots \mid \delta_k$ sont les productions A_j courantes.

  Fait ;

  Eliminer les récursivités gauches immédiates des productions A_i;

Fait ;

Elimination de la récursivité à gauche indirecte

- **Exemple**:
- Eliminer la récursivité à gauche de la grammaire suivante:

  S → Ab | b | Sb

  A → a | AS | Sb
- Ordre **S , A**
- **Etape 1:** Elimination de la récursivité à gauche directe:

  S → AbS' | bS'

  S' → bS' | ε
- **Etape 2:** Substitution

  A → a | AS | AbS'b | bS'b
- Elimination de la récursivité gauche directe:

  A → aA' | bS'bA'

  A' → SA' | bS'bA' | ε

Analyse syntaxique par descente récursive

Analyse syntaxique par descente récursive

- L'analyse syntaxique **par descente récursive** est une méthode d'analyse syntaxique **descendante** dans laquelle on exécute un ensemble **de procédures récursives** pour traiter la chaîne d'entrée.
- Une **procédure** est associée à chaque **non-terminal** d'une grammaire.
- Une analyse syntaxique **récursive LL(1)** se décompose en deux parties :
- **(i) construction de l'analyseur** : on écrit une procédure d'analyse pour chaque non-terminal
- **(ii) reconnaissance d'un mot** Une suite des appels de ***procédure* récursive** pour reconnaître un mot du langage.

Construction de l'analyseur

- **Condition:** la grammaire **doit vérifier** les **conditions LL(1)**.
- **Les étapes de construction**:
  - On ajoute la règle de production suivante : **Z → S#** où **S** est l'axiome de la grammaire et **#** le marqueur de fin de la chaine à analyser;
  - A chaque **non terminal** de la grammaire correspond **une procédure**;

Ecriture des procédures

- On utiliser les variables **tc** et **ts** pour désigner, respectivement, le **symbole courant** de la chaîne d'entrée à analyser et son **symbole suivant**;
- Le traitement d'un membre droit d'une règle de production $A \rightarrow \alpha_i$ se fera comme suit :
- **Chaque terminal** en partie droite est comparé au symbole courant de l'entrée:
  - **Si égalité** lire le symbole suivant
  - **Sinon** déclencher l'impression d'une erreur ;
- **Chaque non-terminal** en partie droite conduit à son appel.

Ecriture des procédures

Le traitement d'un membre droit d'une règle de production $A \rightarrow \alpha_i$ se fera comme suit :

- **Chaque terminal** en partie droite est comparé au symbole courant de l'entrée:
  - **Si égalité** lire le symbole suivant
  - **Sinon** déclencher l'impression d'une erreur ;
- **Chaque non-terminal** en partie droite conduit à son appel.

Exemple d'analyse

- Considérer la grammaire
  G = <{S,A}, {a,b,c},S,P> suivante:
  S → a A b | ε
  A → c A | ab
- On ajoute la règle suivante :
  Z → S #
- Ensembles **PREMIER** et **SUIVANT** :

	PREMIER	SUIVANT
S	a ε	#
A	c a	b

- La grammaire **G** précédente vérifie les conditions **LL(1)**

Ecriture des procédures

Procédure Z( )
Début
S( );
Si tc = '#'
Alors "Chaine syntaxiquement correcte"
Sinon "Erreur"
FinSi;
Fin.

Procédure S( )
Début
Si tc = 'a'
Alors
tc = ts;
A( );
Si tc = 'b'
Alors tc=ts
Sinon "Erreur"
FinSi;
FinSi;
Fin.

Ecriture des procédures

```
Procédure A( )
Début
Si tc='c'
Alors
    tc = ts ;
    A( );
Sinon
Si tc = 'a'
    Alors tc=ts
        Si tc ='b'
        Alors tc =ts
        Sinon "Erreur"
        FinSi;
    Sinon "Erreur"
    FinSi;
FinSi;
Fin.
```

Exemple d'analyse (1)

- Analyse de la chaine : **accabb#**

Contenu de la Pile	Restant à analyser	Action
Z	a c c a b b #	**Appel S**
ZS	a c c a b b #	**Avancer ; Appel A**
ZSA	c c a b b #	**Avancer ; Appel A**
ZSAA	c a b b #	**Avancer ; Appel A**
ZSAAA	a b b #	**Avancer ; avancer ; retour A**
ZSAA	b#	**retour A**
ZSA	b#	**retour A**
ZS	b#	**avancer ; retour S**
Z	#	**"Chaine Correcte"**

Exemple d'analyse (2)

- Analyse de la chaine : **acaabb#**

Contenu de la Pile	Restant à analyser	Action
Z	a c a a b b #	**Appel S**
ZS	a c a a b b #	**Avancer ; Appel A**
ZSA	c a a b b #	**Avancer ; Appel A**
ZSAA	a a b b #	**Avancer ; "Erreur"**
ZSAA	a b b #	**"Chaine incorrecte"**