

## Module 3 : Structures de données

```
// structure décrivant une instance de Follow
struct follow {

    // table de hachage (UNE SEULE)

    struct text * pTextRef; // document de référence

    // document représentant le nouveau texte

    // document représentant la différence entre les deux

};
```

## Module 3 : Structures de données

```
// structure décrivant un document
struct text {

    char * text;// le texte brut

    struct token ** tokenizedText; // le texte découpé

    // longueur du texte brut, nb jetons total, nb jetons WORD,...

};
```

## Module 3 : Structures de données

```
// unité lexicale du texte
struct token {

    enum { WORD, SHORT_SPACE, SPACE, // espace : ' ', '\n' ou '\t'
          ERASE, INSERT, REPLACE, EMPTY } type;

    int textOffset; // position dans le texte (char*) d'origine

    union {

        char * word; // pointeur sur un mot de la table de hachage

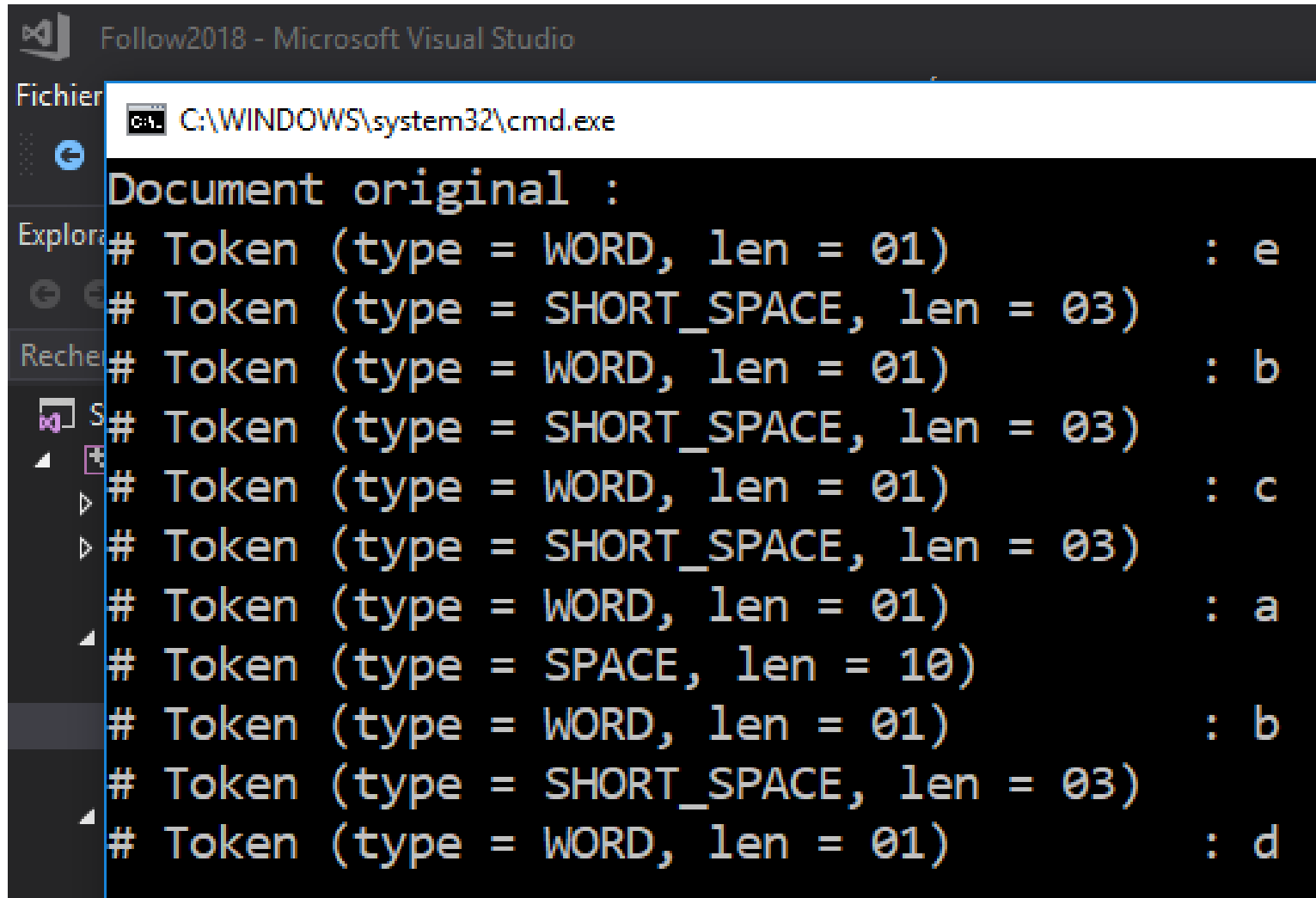
        char space[4]; // 4 délimiteurs max

    } data; // ( accès via data.word OU data.space selon le type )

};
```

## Module 3 : exemple parsing char\* vers tokens

Document original contient : **e b c a**      **b d**



The screenshot shows a Windows command prompt window titled "Follow2018 - Microsoft Visual Studio". The command prompt is running "C:\WINDOWS\system32\cmd.exe". The output of the command is as follows:

```
Document original :  
# Token (type = WORD, len = 01)      : e  
# Token (type = SHORT_SPACE, len = 03)  
# Token (type = WORD, len = 01)      : b  
# Token (type = SHORT_SPACE, len = 03)  
# Token (type = WORD, len = 01)      : c  
# Token (type = SHORT_SPACE, len = 03)  
# Token (type = WORD, len = 01)      : a  
# Token (type = SPACE, len = 10)  
# Token (type = WORD, len = 01)      : b  
# Token (type = SHORT_SPACE, len = 03)  
# Token (type = WORD, len = 01)      : d
```

## Module 3 : exemple parsing char\* vers tokens

Nouveau document contient : **a b c d**

Nouveau document :


```
# Token (type = WORD, len = 01)      : a
# Token (type = SHORT_SPACE, len = 03)
# Token (type = WORD, len = 01)      : b
# Token (type = SHORT_SPACE, len = 03)
# Token (type = WORD, len = 01)      : c
# Token (type = SHORT_SPACE, len = 03)
# Token (type = WORD, len = 01)      : d
```

## Module 3 : PLSC $\Leftrightarrow$ Plus Longue Sous-chaine Commune



E B C A B D  
A B C D

Exemple 1



A B C D E  
C E I J

Exemple 2

Éléments communs doivent être dans le même ordre,  
mais pas nécessairement consécutifs

## Notations

$A_i, \quad i = 1, \dots, m$

$B_j, \quad j = 1, \dots, n$

$PLSC(i, j) = \langle c_1, \dots, c_k \rangle$

$lg(i, j) = k$

## Propriétés :

$lg(i_1, j) \leq lg(i_2, j) \text{ si } i_1 \leq i_2$

$lg(i, j_1) \leq lg(i, j_2) \text{ si } j_1 \leq j_2$

Si  $A_i = B_j$ ,

$PLSC(i, j) = PLSC(i-1, j-1) + \langle A_i \rangle$

$lg(i, j) = lg(i-1, j-1) + 1$

Sinon

*si*  $PLSC(i, j-1) = \langle c_1, \dots, c_k, A_i \rangle$

$lg(i, j) = lg(i, j-1) = k + 1$

*si*  $PLSC(i-1, j) = \langle c_1, \dots, c_k, B_j \rangle$

$lg(i, j) = lg(i-1, j) = k + 1$

$lg(i, 0) = 0$

$lg(0, j) = 0$

$$lg(i, j) = \begin{cases} lg(i-1, j-1) + 1 & \text{si } A_i = B_j \\ \max(lg(i, j-1), lg(i-1, j)) & \text{sinon} \end{cases}$$

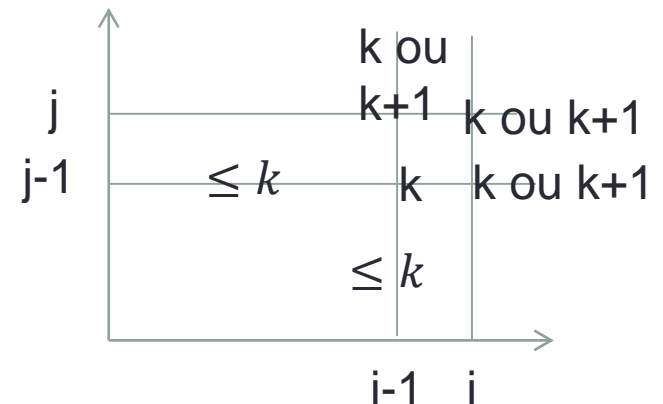
## Algorithme

Calcul de la distance d'édition

PLSC : Plus Longue Sous-chaine Commune

## Exemple

D	0	0	1	2	2	2	3
C	0	0	1	2	2	2	2
B	0	0	1	1	1	2	2
A	0	0	0	0	1	1	1
	0	0	0	0	0	0	0
	E	B	C	A	B	D	



xToken Et j => Nouveau texte



yToken

Et i

=

Ancien texte

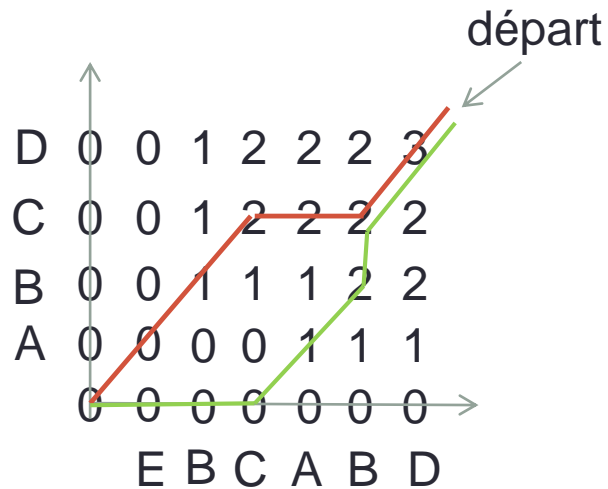
PLSC Matrix :

		a	b	c	d
e	0	0	0	0	0
b	0	0	1	1	1
c	0	0	1	2	2
a	0	1	1	2	2
b	0	1	2	2	2
d	0	1	2	2	3

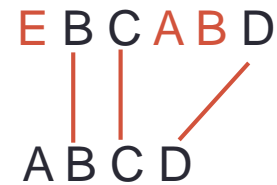
matrix\_lg[i][j]



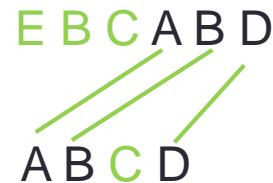
Construction de PLSC : remplacement prioritaire sauf si la suppression ou l'insertion donnent une PLSC plus longue



Suppression prioritaire



Insertion prioritaire



Document résultat (après chaînage arrière PLSC) contient : **e b c a b c d**

Document issu de la comparaison :

```
# Token (type = ERASE, len = 01)      : e
# Token (type = SHORT_SPACE, len = 03)
# Token (type = ERASE, len = 01)      : b
# Token (type = SHORT_SPACE, len = 03)
# Token (type = ERASE, len = 01)      : c
# Token (type = SHORT_SPACE, len = 03)
# Token (type = WORD, len = 01)       : a
# Token (type = SHORT_SPACE, len = 03)
# Token (type = WORD, len = 01)       : b
# Token (type = SHORT_SPACE, len = 03)
# Token (type = INSERT, len = 01)     : c
# Token (type = SHORT_SPACE, len = 03)
# Token (type = WORD, len = 01)       : d
```

- > Dans le TP, on compare des tokens (unités lexicales/mots), et non des lettres !
- > Pour le calcul de la longueur (int\*\*) de PLSC, on tient compte seulement des tokens de type WORD. Il faudra donc déclarer i,j, xToken et yToken
- > Pour le calcul de la longueur, chaque mot est unique dans la table de hachage, donc comparaison par pointeurs possible

Chaque token du nouveau texte apparait dans le résultat :

- // soit parce qu'il appartient à la PLSC,
- // soit parce qu'il a été inséré ou remplace un mot.

- // Apparaissent aussi les mots supprimés ou remplaces
- // provenant du texte de référence