

Notes de cours

GEI 340 : ÉLÉMENTS DE COMPILATION

Chapitre 2

*Analyse lexicale, expressions régulières (ER),
automates à états finis (AEF)*

Ahmed KHOUMSI

PLAN

- Unités lexicales (terminaux) et lexèmes
- Expressions régulières
- Spécification des terminaux à l'aide des expressions régulières
- Reconnaissance des terminaux
- Automates à états finis
- Passage des expressions régulières aux automates à états finis
- Écriture et génération d'un analyseur lexical
- Exemples d'écriture d'un scanner en C

Unités lexicales et lexèmes

Tâches essentielles de l'analyseur lexical = parcourt le programme source pour

- Élimination des caractères inutiles pour la suite de la compilation (commentaires, espaces, tabulation, EOL, ...)
- Regroupement de caractères consécutifs pour constituer des unités lexicales
- Transmettre les unités lexicales et leurs attributs à l'analyseur syntaxique

Le regroupement des caractères consécutifs est tel que chaque chaîne de caractères constituée a une signification collective

Lexème = la chaîne de caractères

Exemple : *position := 50 + vitesse*temps*

Les 7 lexèmes constitués sont :

«position» «:=» «50» «+» «vitesse» «*» « temps»

Unités lexicales et lexèmes (suite)

Unité lexicale = type d'un lexème

Soit par exemple un langage dans lequel sont utilisés :

- des nombres
- des identificateurs
- des opérateurs
- des mots clés

On peut alors définir 4 unités lexicales : **nb**, **id**, **op** et **mc**

Exemple : *position := 50 + vitesse*temps*

- «position», «vitesse» et «temps» sont des lexèmes de l'unité lexicale **id**
- «*», «+» et «:=» sont des lexèmes de l'unité lexicale **op**
- « 50 » est un lexème de l'unité lexicale **nb**

Unités lexicales et lexèmes (suite)

Attributs des unités lexicales

Lorsque le scanner reconnaît une unité lexicale, il la transmet au parser avec plusieurs informations (attributs) qui s'y rattachent

Exemple : *position* := 50 + *vitesse***temps*

Le scanner transmet au parser :

1. le mot **id** (qui identifie l'unité lexicale reconnue)
et le lexème «*position*» (ou bien un pointeur sur ce lexème)
- le mot **op** (qui identifie l'unité lexicale reconnue)
et le mot aff (qui identifie le type de l'opération, qui est une affectation)
- le mot **nb** (qui identifie l'unité lexicale)
et la valeur 50

....

En tout, le scanner transmet :

(**id**, «*position*») (**op**, aff) (**nb**, 50) (**op**, add)
(**id**, «*vitesse*») (**op**, mult) (**id**, «*temps*»)

Unités lexicales et lexèmes (suite)

Attributs (suite)

Il y a plusieurs manières de définir des unités lexicales et leurs attributs

Exemple :

Au lieu de définir l'unité lexicale **op** qui peut avoir des attributs tels que aff, add et mult, on peut aussi directement définir les unités lexicales **aff**, **add** et **mult** qui sont transmises sans attribut.

Remarque : D'autres attributs seront déterminés dans les phases suivantes

Exemple : À un identificateur, on peut associer l'attribut *type* qui sera utilisé par l'analyseur sémantique pour effectuer le contrôle de type

Unités lexicales et lexèmes (suite)

Table des symboles

Utilisée pour stocker les unités lexicales (avec leurs attributs) constituant un programme source.

Le stockage commence lors de la phase d'analyse lexicale et il est complété par les analyseurs syntaxique et sémantique

Exemple :

Pour le mot position, le scanner stocke les informations :

- id
- «*position*»

D'autres informations (attributs) seront ajoutées par la suite, par exemple :

- le type de l'identificateur : entier, réel, chaîne de caractères ...
- la portée : globale, locale (à une procédure) ...

Unités lexicales et lexèmes (suite)

Implantation de la table des symboles

Elle peut être implantée, par exemple, comme un tableau de structures où :

- le premier champ d'une structure identifie l'unité lexicale
- les autres champs dépendent du premier champ

Exemple 1 : identificateur

- Deux champs : **id** et un pointeur sur le lexème
- Deux champs (qui restent vides pour l'instant) : le type et la portée

Exemple 2 : nombre

- Deux champs : **nb** et la valeur du nombre

Exemple 3 : opérateur

- Deux champs : **op** et le type d'opérateur (add, mult, ...)
- Deux champs (qui restent vides pour l'instant) :
les types des opérandes et du résultat

Remarque : Des champs peuvent être ajoutés dans les phases suivantes, en fonction du besoin (conception incrémentale)

Expressions régulières

Alphabet : Ensemble fini de caractères utilisés, noté Σ

Langage : Ensemble de chaînes de caractères d'un alphabet donné Σ , noté L_Σ

Exemple : - alphabet = {a, b}
- langage = { ϵ , a, bba, abba, abbbba, ...} (où ϵ dénote la chaîne vide)

Expressions régulières : Elles permettent de définir formellement tout langage appartenant à la classe des langages dits réguliers

Soit r une expression régulière :

- le langage défini par r est alors désigné par $L(r)$
- on dira que r *dénote* $L(r)$

Expressions régulières (suite)

Voici les règles qui sont respectées par les expressions régulières

Règle 1 : ϵ est une expression régulière qui dénote $\{\epsilon\}$

Règle 2 : Si a appartient à Σ , alors a est une expression régulière qui dénote $\{a\}$

Règle 3 : Soient : - r et s deux expressions régulières
- $L(r)$ et $L(s)$ les langages dénotés par r et s

Alors :

- (r) est une expression régulière qui dénote $L(r)$
- $r|s$ est une expression régulière qui dénote le langage $L(r) \cup L(s)$
(c-à-d. l'union des deux langages $L(r)$ et $L(s)$)
- rs est une expression régulière qui dénote le langage $L(r) L(s)$
(c-à-d. la concaténation des deux langages $L(r)$ et $L(s)$)
- r^* est une expression régulière qui dénote le langage $L(r)^*$
(c-à-d. toutes les chaînes de $L(r)$ répétées un nombre quelconque (positif ou nul) de fois)

Expressions régulières (suite)

Un langage régulier est un langage dénoté par une expression régulière

Exemples d'expressions régulières :

Soit $\Sigma = \{0,1,2,3,4,5,6,8,9\}$

Exemple 1 : $(0|1|2|3|4|5|6|7|8|9)$ est une expression régulière dénotant l'ensemble des nombres entiers à un chiffre

Exemple 2 : $(0|1|2|3|4|5|6|7|8|9)(0|1|2|3|4|5|6|7|8|9)$ est une expression régulière dénotant l'ensemble des nombres entiers à deux chiffres
Si on pose $r = (0|1|2|3|4|5|6|7|8|9)$, on peut aussi écrire rr ou $(r)(r)$

Exemple 3 : $r r^*$ dénote l'ensemble des nombres entiers

Exemple 4 : $r^* = (r r^* | \varepsilon)$

Expressions régulières (suite)

Exemple d'expression régulière avec $\Sigma = \{a, b\}$

$(a|b)^*$, qui est aussi équivalente à $(a^*b^*)^*$, dénote l'ensemble des chaînes constituées de a et b (à titre d'exercice, vous pourriez démontrer l'égalité ci-dessus)

D'autres opérateurs ont été définis pour représenter les expressions régulières :

- r^+ signifie r écrite un nombre > 0 de fois

Autrement dit : $r^+ = r r^*$

exemple : $L((ab)^+) = \{ab, abab, ababab, \dots\}$

- $r^?$ signifie r écrite 0 ou une fois

Autrement dit : $r^? = (r|\varepsilon)$

Ce qui implique :

- $L(r)$ est l'intersection de $L(r^?)$ et $L(r^+)$

- $L(r^*)$ est l'union de $L(r^?)$ et $L(r^+)$

exemple : $L((ab)^?) = \{\varepsilon, ab\}$

Spécification des unités lexicales

Les expressions régulières permettent de spécifier des unités lexicales

Exemple 1 : Σ = ensemble des chiffres et des lettres

On définit un identificateur par une chaîne de caractères dont :

- le premier caractère est une lettre
- les autres caractères peuvent être des chiffres ou des lettres

Formellement :

- $\Sigma = L((0|1|2|3|4|5|6|7|8|9|A|B|C|\dots|Z|a|b|c|\dots|z))$
- lettre = $(a|b|c|\dots|A|B|C|\dots|Z)$
- chiffre = $(0|1|2|\dots|9)$

On a alors :

- identificateur : **id** = $\text{lettre}(\text{lettre}|\text{chiffre})^*$
- nombre : **nb** = $\text{chiffre}(\text{chiffre})^* = \text{chiffre}^+$
- mot clé : **mc** = $(\text{«if»} | \text{«then»} | \text{«else»})$

Spécification des unités lexicales (suite)

Exemple 2 :

- chiffre = $(0|1|2|\dots|9)$
- chiffres = $\text{chiffre}(\text{chiffre})^* = \text{chiffre}^+$
- fraction = $(\text{.chiffres})?$
- exposant = $(E(+|-)?\text{chiffres})?$
- **nb** = $(\text{chiffres})(\text{fraction})(\text{exposant})$
- lettre = $A|B|\dots|Z|a|b|\dots|z$
- **id** = $\text{lettre}(\text{lettre}|\text{chiffre})^*$
- **si** = «if»
- **alors** = «then»
- **sinon** = «else»
- **oper** = $< | <= | = | <> | > | >= | :=$
- **separ** = $(| \backslash t | \backslash n)$
- **espace** = separ^+

Expressions régulières soulignées sont reconnues comme unités lexicales

Remarque : l'unité lexicale **espace** n'est pas forcément transmise par le scanner au parser (choix de conception)

Spécification des unités lexicales (suite)

Exemple 2 (suite) : La reconnaissance des expressions régulières comme unités lexicales peut être représentée par le tableau suivant

Expression régulière	unité lexicale	attribut(s) (assignés par le scanner)
espace	-	-
«if»	si	-
«then»	alors	-
«else»	sinon	-
id	id	pointeur vers lexème
nb	nb	valeur
« < »	oper	PPQ
« <= »	oper	PPE
« = »	oper	EGA
« <> »	oper	DIF
« > »	oper	PGQ
« >= »	oper	PGE
« := »	oper	AFF

Reconnaissance des unités lexicales

Nous travaillerons sur l'exemple 2 précédent pour montrer comment le scanner reconnaît les unités lexicales.

Nous allons voir dans l'ordre :

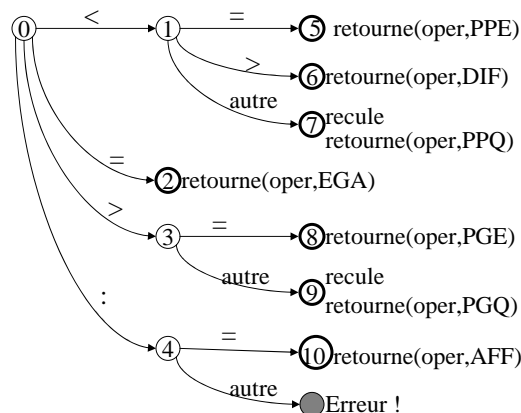
- reconnaissance d'un opérateur < <= = <> > >= :=
- reconnaissance d'un identificateur ou d'un mot clé (if, then, else)
- reconnaissance d'un nombre

Reconnaissance d'un opérateur

- Lecture d'un caractère c
- si c est égal à $<$ alors
 - on lit le caractère suivant c'
 - si c' est égal à $=$ alors on a reconnu l'opérateur $<=$
 - si c' est égal à $>$ alors on a reconnu l'opérateur $<>$
 - sinon - on recule (pour que c' soit lu à la prochaine lecture)
 - on a reconnu l'opérateur $<$
- si c est égal à $=$ alors on a reconnu l'opérateur $=$
- si c est égal à $>$ alors
 - on lit le caractère suivant c'
 - si c' est égal à $=$ alors on a reconnu l'opérateur $>=$
 - sinon - on recule (pour que c' soit lu à la prochaine lecture)
 - on a reconnu l'opérateur $>$
- si c est égal à $:$ alors
 - on lit le caractère suivant c'
 - si c' est égal à $=$ alors on a reconnu l'opérateur $:=$
 - sinon il y a erreur

Reconnaissance d'un opérateur (suite)

La reconnaissance d'un opérateur peut être représentée par un automate à états finis (AEF) AEF_OP :



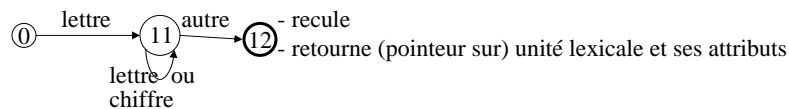
- les ronds représentent des états
- les flèches d'un état à un autre représentent des transitions d'état

Reconnaissance d'un opérateur (suite)

- Une transition \xrightarrow{c} représente la lecture d'un caractère c
Cette lecture fait passer d'un état à un autre
- Un état \bigcirc signifie qu'aucune unité lexicale n'a été reconnue
- Un état \bigcirc signifie qu'une unité lexicale a été reconnue
Dans ce cas, le scanner doit retourner l'unité lexicale reconnue avec les attributs correspondants.
Dans certains cas, le scanner doit reculer son pointeur de lecture avant de retourner l'unité lexicale reconnue
- Un état \bullet signifie qu'une erreur a été détectée
Un message d'erreur doit normalement être envoyé

Reconnaissance d'un identificateur

Elle peut être représentée par l'AEF_ID:

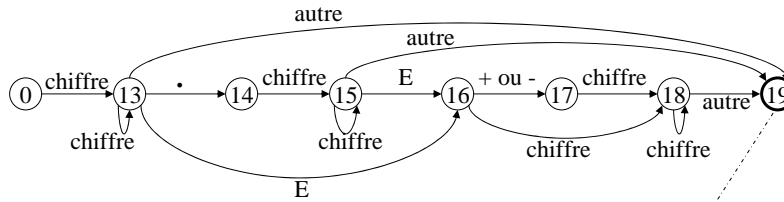


Lorsqu'un identificateur est reconnu par le scanner, alors :

- le scanner recule son pointeur de lecture
- si l'unité lexicale reconnue est un mot clé (if, then, else) alors
le scanner retourne l'unité lexicale reconnue **si, alors** ou **sinon**
sinon le scanner retourne (un pointeur sur) :
 - l'unité lexicale **id**
 - (un pointeur sur) le lexème

Reconnaissance d'un nombre

Elle peut être représentée par l'AEF_NB:



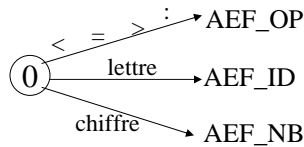
- recule
- retourne (pointeur sur) unité lexicale et ses attributs

Lorsqu'un nombre est reconnu par le scanner, alors :

- le scanner recule son pointeur de lecture
- il retourne pointeur sur) : - l'unité lexicale nb
- (un pointeur sur) la valeur

Reconnaissance des unités lexicales (suite)

Pour récapituler, la reconnaissance des unités lexicales peut être représentée par l'AEF schématisé comme suit :



Remarque : pour rendre l'AEF complètement spécifié, on ajoute à tout état une transition autre (si elle n'existe pas) qui mène à un état ERREUR.

Reconnaissance des unités lexicales (suite)

Selon le choix de conception, les unités lexicales reconnues peuvent être déposées (avec leurs attributs) dans une table des symboles :
une seule fois ou à chaque fois qu'on les rencontre
Ce traitement peut être intégré à la fonction qui retourne l'unité lexicale

Table des symboles pour identificateurs

- La sauvegarde dans une table des symboles est particulièrement pratique pour les identificateurs.
- Les éléments de la table doivent alors avoir une structure permettant de stocker les différents types d'identificateurs. Par exemple :

- Pointeur vers lexème
 - Type
 - Portée
 - ...} (ces champs ne sont pas utilisés par le scanner)
(d'autres champs peuvent être ajoutés dans les phases suivantes)
- Lorsqu'un identificateur est reconnu, alors un pointeur vers l'élément de la table des symboles peut être retourné

Automates à états finis (AEF)

Comme nous l'avons vu sur un exemple, un AEF peut être utilisé pour reconnaître des unités lexicales spécifiées par des expressions régulières

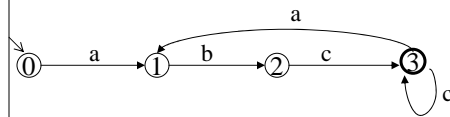
Formellement, un AEF est défini par :

- un alphabet fini : Σ
- un ensemble fini d'états : Q
- un ensemble de transitions : δ
chaque élément de δ est un triplet (q, σ, r) , où :
 - q et r sont des éléments de Q
 - σ est un élément de Σ
- un état initial q_0 (qui appartient à Q)
- un ensemble d'états finaux (accepteurs) : F (qui est inclus dans Q)

Exemple d'un AEF

$\Sigma = \{a, b, c\}$, $Q = \{0, 1, 2, 3\}$, $q_0 = 0$, $F = \{3\}$ $\delta = \{(0, a, 1), (1, b, 2), (2, c, 3), (3, a, 1), (3, c, 3)\}$

L'AEF peut être représenté graphiquement des deux manières suivantes :



Où : est l'état initial
 est un état final

événement \ état de départ	a	b	c
0	1		
1		2	
2			3
3	1		3

Où : l'état initial est sur la première ligne
 l'état final est encadré

Automates à états finis (suite)

Dans le cas de l'analyse lexicale, une transition représente la lecture du caractère a

L'AEF de la page précédente signifie :

- initialement, on est dans l'état 0
- à partir de l'état 0, on peut lire a, et on passe alors à l'état 1
- à partir de l'état 1, on peut lire b, et on passe alors à l'état 2
- à partir de l'état 2, on peut lire c, et on passe alors à l'état 3
- à partir de l'état 3, on peut lire :
 - a, et on passe alors à l'état 1
 - c, et on reste dans l'état 3

Chaînes de caractères reconnues (acceptées)

- Ce sont les chaînes de caractères qui mènent de l'état initial à un état final
- Pour notre exemple, l'AEF accepte les chaînes définies par l'expression régulière $abc((abc)^*|c)^*$

Passage d'une expression régulière à un AEF

Il est démontré que la passage est toujours possible dans les deux sens
(expression régulière \longleftrightarrow AEF)

Ce passage nous permettra de :

- spécifier les unités lexicales par des AEFs
- de reconnaître les unités lexicales en utilisant les AEFs qui les spécifient

On a vu qu'une expression peut être construite récursivement à l'aide des règles :

Règle 1. ϵ est une expression régulière

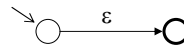
Règle 2. c est une expression régulière (où c est un caractère)

Règle 3. Soient $r1$ et $r2$ deux expressions régulières

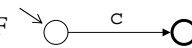
alors : $\left. \begin{array}{l} (r1)(r2) \\ (r1)|(r2) \\ (r1)^* \end{array} \right\}$ sont des expressions régulières

Règles de passage d'une expression régulière à un AEF

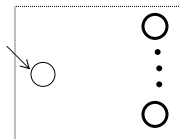
Règle 1. Pour l'expression ϵ , on obtient l'AEF
(on dit que la transition se fait *spontanément*)



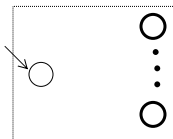
Règle 2. Pour l'expression c , on obtient l'AEF
(où c est un caractère)



Règle 3. On considère deux expressions régulières $r1$ et $r2$ représentées par des AEFs $A1$ et $A2$



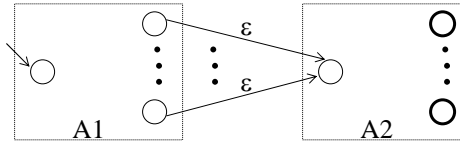
A1



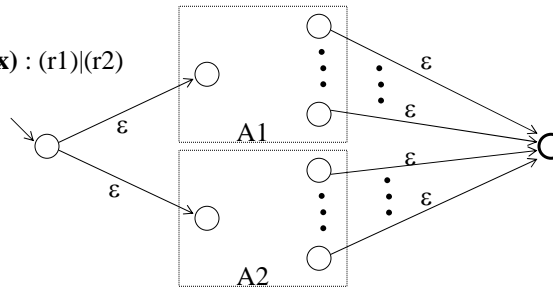
A2

Règle 3 (de passage d'une expression régulière à un AEF, suite)

Concaténation : $(r_1)(r_2)$

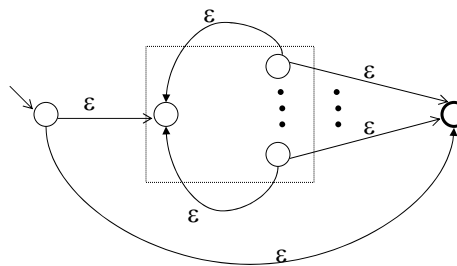


Réunion (choix) : $(r_1)|(r_2)$



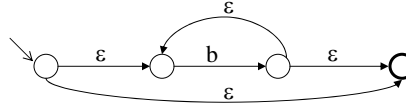
Règle 3 (de passage d'une expression régulière à un AEF, suite)


Fermeture de Kleene (itération) : $(r)^*$



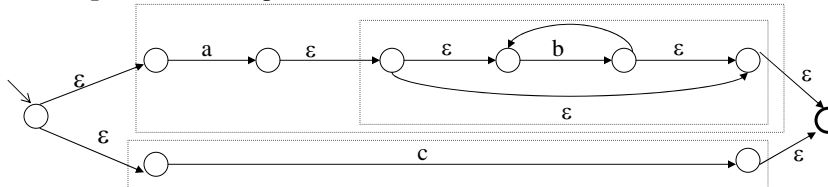
Exemples de passage d'une expression régulière à un AEF

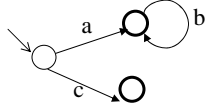
Exemple 1 : AEF acceptant b^*



Cet AEF peut aussi se simplifier en 

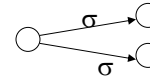
Exemple 2 : AEF acceptant $ab^*|c$



Cet AEF peut aussi se simplifier en 

Transformations des AEFs obtenus

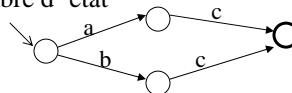
- Les AEFs obtenus sont en général non déterministes, c-à-d. que la lecture d'un même caractère peut mener à plusieurs états



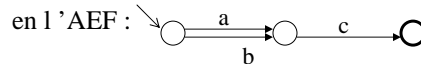
- Les AEFs obtenus contiennent en général des transitions spontanées ϵ

- Les AEFs obtenus peuvent aussi être non minimaux, c-à-d. qu'on peut diminuer leur nombre d'état

Exemple : l'AEF



peut être transformé



Pour un AEF obtenu, il s'agit alors de :

- supprimer les ϵ
 - le rendre déterministe
 - le rendre minimal
- } pour avoir un scanner plus optimal et plus efficace

Réalisation d'un scanner

L'AEF spécifiant les différentes unités lexicales acceptées est en fait un modèle du scanner, c-à-d. le scanner est un programme qui code l'AEF

La réalisation d'un scanner consiste donc à :

1. Écrire une expression régulière pour chaque unité lexicale
Soit donc r_1, r_2, \dots, r_n
2. Construire un AEF qui reconnaît $r = (r_1 \mid r_2 \mid \dots \mid r_n)$
3. Transformer l'AEF obtenu en :
 - lui supprimant les ϵ ,
 - le rendant déterministe
 - le rendant minimal
4. Réaliser le programme qui code l'AEF obtenu

- L'étape 1 est faite manuellement
- Les étapes 2 à 4 peuvent être effectuées :
 - soit manuellement,
 - soit automatiquement, par exemple à l'aide de l'outil LEX

Réalisation manuelle d'un scanner

On considère l'exemple 2 des pages 14 et 15

Les unités lexicales à reconnaître sont :

- **oper** (opérateur) $< \leq = <> > \geq :=$
- **id** (identificateur) mot constitué d'une lettre suivie par lettres et chiffres
- **mc** (mot clé) **si alors sinon**
- **nb** (nombre) constitué de chiffres

- Les blancs, tabulations et retour à la ligne sont ignorés
- On simplifie le problème en supposant que les nombres sont des entiers, donc sans point décimal ni exposant.

Les états 14 à 18 (voir page 21) sont donc supprimés

- L'étape 1 est illustrée dans les pages 14 et 15
- L'étape 2 est illustrée dans les pages 16 à 23
- L'étape 3 est inutile
- **Faisons l'étape 4**

Réalisation manuelle d'un scanner (étape 4)

```

Unlex Scanner()
{
    état = 0;
    do {
        switch(état) {
            case 0 : /* traitement de case 0 */ break;
            case 1 : /* traitement de case 1 */ break;
            case 2 : return(EGA);
            case 3 : /* traitement de case 3 */ break;
            case 4 : /* traitement de case 4 */ break;
            case 5 : return(PPE);
            case 6 : return(DIF);
            case 7 : recule(1); return(PPQ);
            case 8 : return(PGE);
            case 9 : recule(1); return(PGQ);
            case 10 : return(AFF);
            case 11 : /* traitement de case 11 */ break;
            case 12 : /* traitement de case 12 */ break;
            case 13 : /* traitement de case 13 */ break;
            case 19 : /* traitement de case 19 */ break;
        } /* fin de switch */
    } while (car != EOF)
    return(0);
}

```

Réalisation manuelle d'un scanner (étape 4, suite)

```

Case 0 : car = lit_caractère();
if (car == ' ' || car == '\t' || car == '\n') {
    état = 0;
}
else if (car == '<') état = 1;
else if (car == '=') état = 2;
else if (car == '>') état = 3;
else if (car == '=') état = 4;
else if (isletter(car)) {
    chaîne = car; état = 11;
}
else if (isdigit(car)) {
    val = car - '0'; état = 13;
}
else erreur();

break;

Case 1 : car = lit_caractère();
if (car == '=') état = 5;
else état = 9;

break;

```

Réalisation manuelle d'un scanner (étape 5, suite)

```
Case 3 : car = lit_caractère();
        if (car == ' ') état = 8;
        else état = 9;
break
```

```
Case 4 : car = lit_caractère();
        if (car == ' ') état = 10;
        else erreur;
```

```
Case 11 : car = lit_caractère();
         if (isletter(car) || isdigit(car)) {
             chaîne = chaîne + car;
             état = 11; /* inutile */
         }
         else état = 12;
break;
```

Réalisation manuelle d'un scanner (étape 4, suite)

```
Case 12 : recule(1);
        /* éventuellement traitement table symboles */
        if iskeyword(chaîne) return(UnilexKW());
        else {
            /* retourne (pointeur sur) structure
               - (pointeur sur) lexème
            */
        }
}
```

```
Case 13 : car = lit_caractère(0);
        if isdigit(car) {
            val = val*10 + car - '0';
            état = 13; /* inutile */
        }
        else état = 19
break
```

```
Case 19 : recule(1);
        /* retourne (pointeur sur)
           - nb
           - (pointeur sur) valeur
        */
```

Réalisation manuelle d'un scanner (étape 4, suite)

```
UnilexKW() {  
    switch(chaine) {  
        case « if » : return(si);  
        case « then » : return(alors);  
        case « else » : return(sinon);  
    }  
}
```