ED CHAINE DE PRODUCTION DE PROGRAMMEs

CORRIGE

Exercice 1 : Petit compilateur

Question A

Le résultat de l'analyse lexicale est le suivant:

A1 = A * 100 + 2 * B
? A1 = B : C = D
? A = 3
A B = 32
Lors de l'analyse de la phrase "A = 2 ! 3", on constate la présence d'un caractère qui ne fait partie d'aucun symbole de notre langage. Il s'agit donc d'une erreur lexicale.

2.2.2. Question B

Retrouver la structure syntaxique consiste à retrouver les règles de productions qui ont été utilisées pour construire la phrase.
Pour A1 = A * 100 + 2*B, on cherche à développer `<instruction>`, en utilisant l'une des règles possibles, qu'il faut essayer successivement, en cas de refus, jusqu'à trouver le bon choix. On développe donc

```
1 <instruction> ::= <affectation>
2 <affectation> ::= <identificateur> = <expression>
```

qui permet d'accepter "A1 =". Ensuite, on utilise successivement les règles

```
2 <expression> ::= <facteur> {+|-} <expression> | <facteur>
4 <facteur> ::= <terme> {*|/} <facteur> | <terme>
5 <terme> ::= <identificateur> | <nombre> | ( <expression> )
```

permettant d'accepter "A". Le retour sur la règle 4 définissant `<facteur>` conduit ensuite à accepter "*", et à développer les règles

```
6 <facteur> ::= <terme> {*|/} <facteur> | <terme>
7 <terme> ::= <identificateur> | <nombre> | ( <expression> )
```

qui conclut sur le nombre 100. Cette fois le retour sur la règle 6 définissant `<facteur>` conduit à la terminaison de cette règle, puis à la terminaison de la règle 4, permettant alors l'acceptation du "+". On développe alors les règles

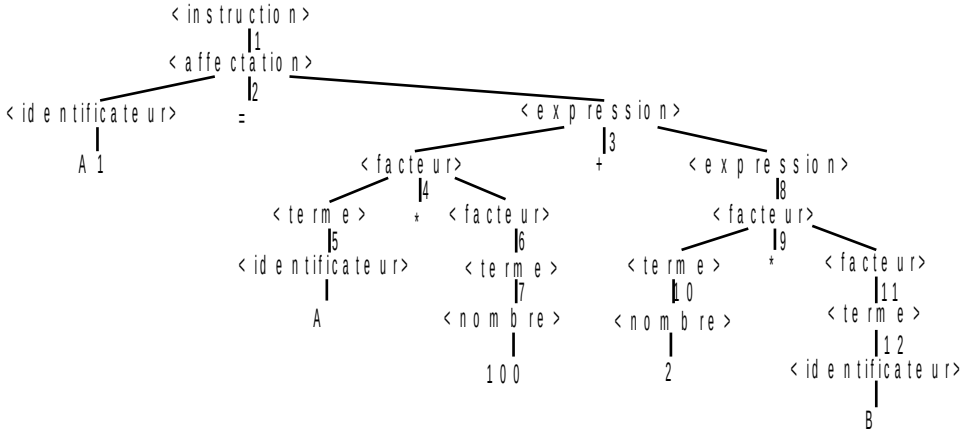
```
7 <expression> ::= <facteur> {+|-} <expression> | <facteur>
  <facteur> ::= <terme> {*|/} <facteur> | <terme>
10    <terme> ::= <identificateur> | <nombre> | ( <expression> )
```

permettant d'accepter "2". Le retour sur la règle 9 définissant `<facteur>` conduit ensuite à accepter "*", et à développer les règles

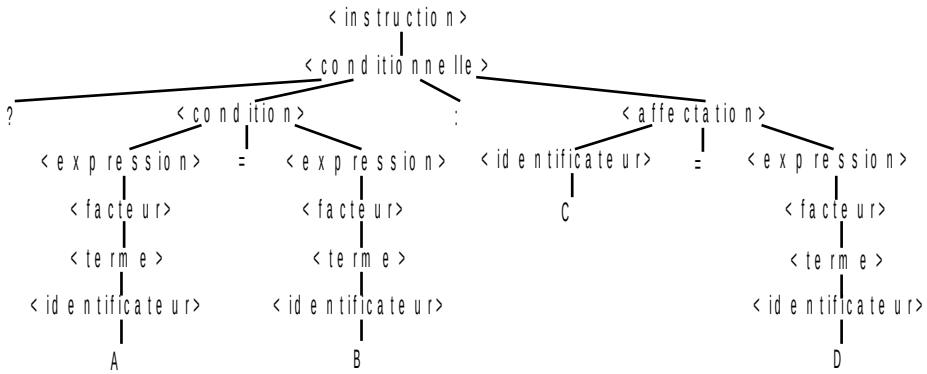
```
11  <facteur> ::= <terme> {*|/} <facteur> | <terme>
12    <terme> ::= <identificateur> | <nombre> | ( <expression> )
```

qui conclut sur l'identificateur B. Le retour sur la règle 11 conduit à la terminer, impliquant la terminaison de la règle 9. Le retour sur la règle 8 conduit également à la terminer, impliquant

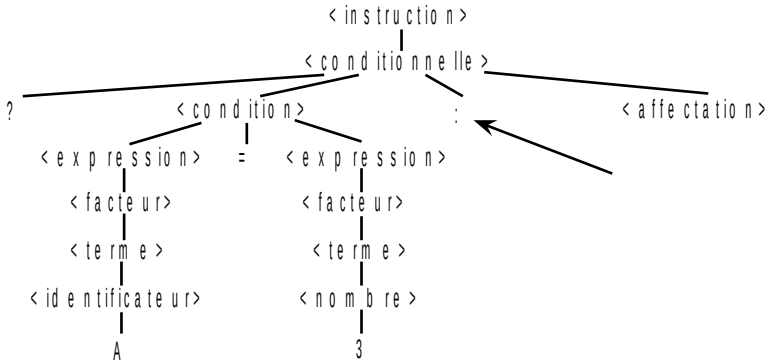
la terminaison des règles 3 puis 2 puis 1, et ensuite à l'acceptation de `K_fin_ligne`, et l'acceptation finale de la phrase. La structure syntaxique est donc:



Pour ? A1 = B : C = D, nous obtenons

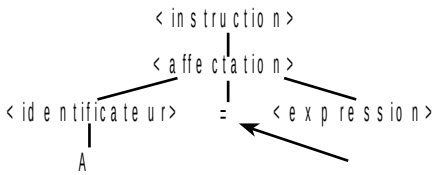


Pour la phrase ?A=3, l'analyseur syntaxique commence à construire la structure suivante



Lorsqu'il vérifie que le symbole est ':', il trouve en fait `K_fin_ligne`. Comme il n'a pas d'autre choix possible, il indique une erreur syntaxique.

Pour la phrase AB=32, rappelons d'abord que l'analyseur lexical l'a transformée en suite de symboles codés -1, -2, -287, 32, -297. Ceci montre bien la présence de deux identificateurs qui se suivent. L'analyseur syntaxique commence à construire la structure suivante



Lorsqu'il vérifie que le symbole est '=', il trouve en fait l'identificateur 'B'. Comme il n'a pas d'autre choix possible, il indique une erreur syntaxique.

La phrase A = 2 ! 3 étant erronée lexicalement, il est inutile d'en faire l'analyse syntaxique. De fait, le caractère '!' ne correspondant à aucun symbole, ne peut être codé.
Notons que dans certains cas, pour permettre de trouver le maximum d'erreurs le plus vite possible, l'analyseur lexical ignorera ce caractère après avoir signalé l'erreur. Il transmettra à l'analyseur syntaxique la suite codée, comme s'il n'existait pas.

Exercice 2 : Compilation

A.

```
<addition> ::=  <identificateur> :=  <identificateur>  +  <valeur
  entière> ;  |  <identificateur> :=  <identificateur>  +  <valeur
  réelle> ;
```

<multiplication>::=<identificateur>:=<identificateur>*<identificateur>;

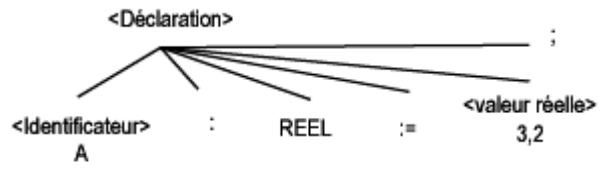
B.

```
PROGRAM A3 /
A :REEL := 3,2;
B :ENTIER:=3,2;
A :=A + 5,1;
B :=B *6 1 ,
FIN
```

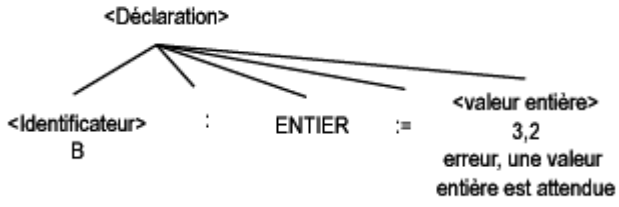
Une erreur lexicale est levée à la lecture du symbole / qui ne fait pas partie des symboles admis dans le langage.

C.

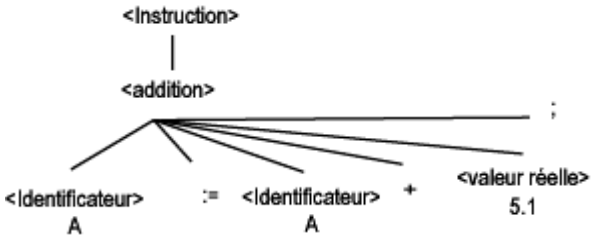
PHRASE 1 A : REEL := 3,2 ;



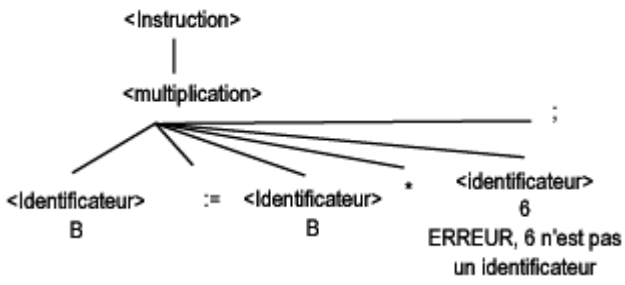
PHRASE 2 B : ENTIER := 3,2 ;



PHRASE 3 A := A + 5,1;



PHRASE 4 B := B + 61,



Exercice 3 : Étude du chargeur et de l'éditeur de liens

Question A

Question A.1

Le module objet doit se présenter sous la forme suivante:

```
<    entête                 ESSAI     7                 >
<    lien_utilisable        INCR      2                 >
<    lien_à_satisfaire      SUITE                       >
<    code_absolu            0         25                >
<    code_absolu            1         ?                 >
<    code_à_translater      2         10000             >
<    code_absolu            3         30010             >
<    code_absolu            4         20001             >
<    code_à_translater      5         40001             >
<    code_avec_l_a_s        6         1         50000   >
<    fin_module                                         >
```

Question A.2

Lors de l'édition de liens, le module étant placé à l'emplacement 123, il faut ajouter cette valeur aux adresses internes. Comme il est de taille 7, le module placé derrière lui sera placé à l'emplacement relatif 130. Comme SUITE est un lien utilisable défini comme emplacement relatif 8 de ce module, il est donc à l'emplacement relatif 138 du programme. Les enregistrements produits par l'éditeur de liens pour ce module sont donc les suivants:

```
<    code_absolu            123         25              >
<    code_absolu            124         ?               >
<    code_à_translater      125       10123             >
<    code_absolu            126       30010             >
<    code_absolu            127       20001             >
<    code_à_translater      128       40124             >
<    code_à_translater      129       50138             >
```

Question A.3

Les emplacements occupés par ce module après chargement sont définis comme suit:

```
1165        25
1166         ?
1167     11165
1168     30010
1169     20001
1170     41166
1171     51180
```

Le programme étant en 1042, le module commence en 1042+123 = 1165. Par ailleurs, le chargeur ajoute 1042 aux emplacements contenant un code à translater. L'emplacement associé à B a un contenu indéterminé, puisqu'il n'est pas initialisé.

2.6.2. Question B

La structure de `t_enregistrement` convient évidemment pour les modules objets, puisqu'elle permet d'une part la définition des liens à satisfaire et des liens utilisables, d'autre part la translation de code et les liaisons vers d'autres modules par les enregistrements `code_avec_l_a_s`. Notons que les liens à satisfaire étant repérés dans ce cas par leur numéro d'ordre, l'enregistrement `lien_à_satisfaire` correspondant au $n^{ième}$ lien à satisfaire doit se trouver avant les enregistrements `code_avec_l_a_s` mentionnant ce numéro n.

Pour un module exécutable, la structure convient aussi, mais certains enregistrements ne devront pas se trouver dans un tel module. En effet, un module exécutable ne peut contenir que les enregistrements `entête`, `code_absolu`, `code_à_translater`, `adresse_lancement`, ou `fin_module`. On peut admettre aussi les enregistrements `lien_utilisable`, mais les enregistrements `lien_à_satisfaire` et `code_avec_l_a_s` sont interdits.

Exercice 4 : Exemple d'édition de liens

L'adresse d'implantation d'un module est 0 pour le premier, et pour les autres, le premier emplacement laissé libre par le module qui le précède, c'est-à-dire, la somme entre son adresse d'implantation et sa taille:

```
PROGRAMME        0
ETIQUETTE      332
LECTURE        460
IMPRESSION    1300
EDITION       1512
```

La taille totale du programme est la somme des tailles de tous les modules, ou encore l'adresse du premier emplacement laissé libre par le dernier module, c'est-à-dire, la somme entre son adresse d'implantation et sa taille, soit 2154.

La table des liens contient tous les identificateurs présents dans les modules, en tant que lien utilisable ou lien à satisfaire, ainsi que éventuellement les noms des modules. De plus, la table associe à tout identificateur rencontré comme lien utilisable dans un module, leur adresse dans le programme, obtenue en ajoutant à leur adresse dans ce module l'adresse d'implantation du module. Enfin aux noms de modules, la table associe l'adresse d'implantation du module.

```
*  PROGRAMME        0
   OUVRIR         460
   LIRE           800
   FERMER        1192
   EDITER        1512
*  ETIQUETTE      332
   NOM            332
   SOCIETE        364
   ADRESSE        396
   CODEPOST       428
   VILLE          433
*  LECTURE        460
*  IMPRESSION    1300
   IMPRIMER      1300
*  EDITION       1512
```

Les lignes marquées par un “*” ne sont pas toujours dans la table, suivant que l'on utilise ou non la table pour mémoriser les adresses d'implantations des modules.

L'adresse de lancement est la valeur trouvée dans l'un des modules augmentée de l'adresse d'implantation de ce module: 133 + 0 = 133**.**