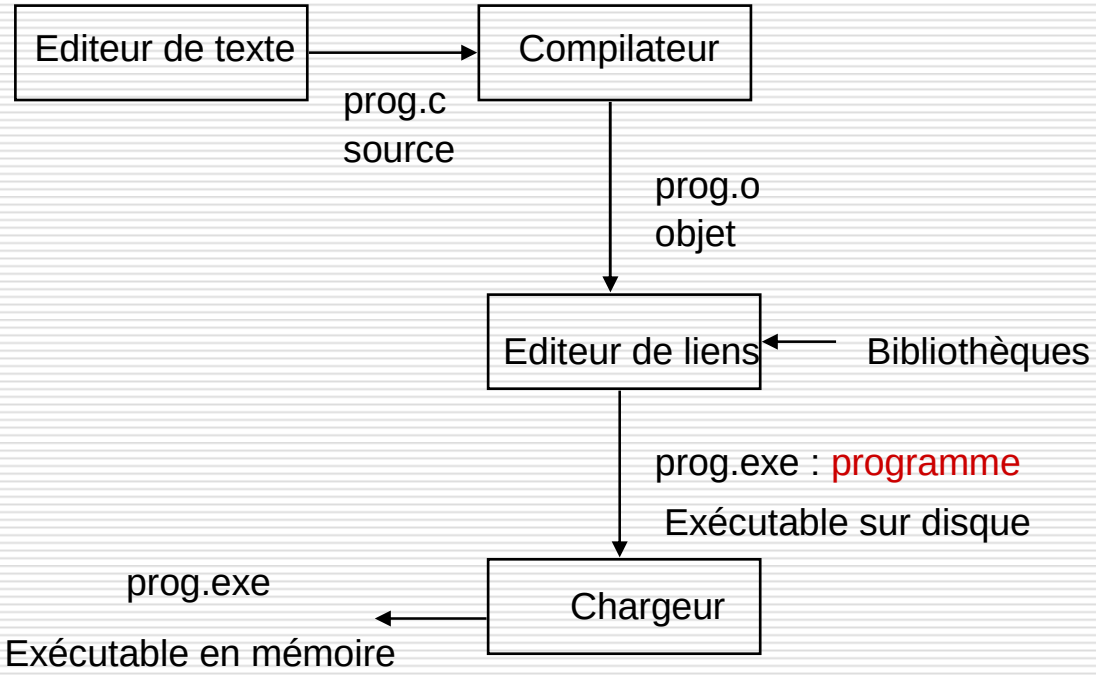
FONCTION D'EXECUTION
Du programme exécutable au processus

La chaîne de production de programmes :
compilation, éditions des liens et chargement

La chaîne de production de programmes

Cette chaîne désigne l'ensemble des étapes nécessaires à la construction d'un programme exécutable à partir d'un programme dit source :

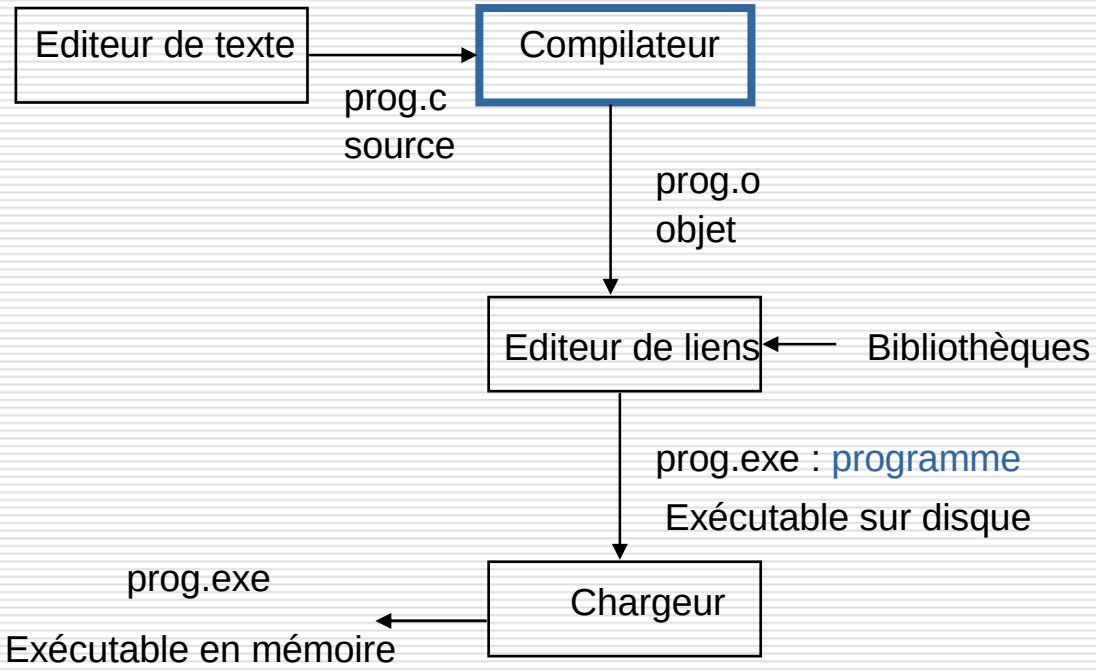
- Compilation
- Edition des liens
- Chargement



La chaîne de production de programmes : compilation

Cette chaîne désigne l'ensemble des étapes nécessaires à la construction d'un programme exécutable à partir d'un programme dit source :

- Compilation
- Edition des liens
- Chargement



Les niveaux de langage de programmation

Programme en langage haut niveau
(indépendant machine physique)

```
While (x > 0)
do
        y := y + 1;
        x := x - 1;
done;
```

COMPILATEUR

Programme en langage d'assemblage
(très proche du langage machine)

```
loop : add  R1, 1
       sub R2, 1
       jmpP loop
```

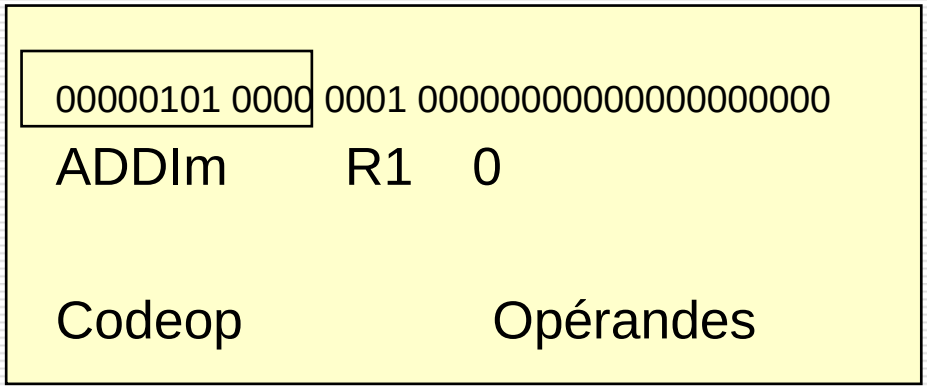
ASSEMBLEUR

Programme en binaire
(langage machine)

```
1000 : 00000110000100000000001
       00001110001000000000001
       01100000000000000001000
```

Les niveaux de langage de programmation

- Chaque processeur possède son propre jeu d'instructions machine (chaîne binaire). Seul es ces instructions sont exécutables par le processeur.
- Le langage d'assemblage est l'équivalent du langage machine. Chaque champ binaire de l'instruction machine est remplacé par un mnémonique alphanumérique.
- Le langage de haut niveau est indépendant de la machine physique. Pour pouvoir être exécuté, un programme en langage de haut niveau doit être traduit vers son équivalent machine : c'est le rôle du compilateur.



Le codage d'un problème ...

Programme en langage de haut niveau
instructions de haut niveau

Compilateur

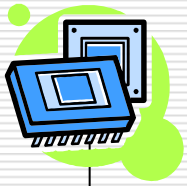
Niveau utilisateur

Machine physique "matérielle"

Programme en langage d'assemblage

Assembleur

processeur



Bus

01101110111110011
01111010001011100
10111101110111111
00111011110111011
00111111000111101

Mémoire centrale

Programme à
exécuter : instructions machine
et valeurs en binaire

traduction

ROLE DU COMPILATEUR

- Un compilateur traduit un **programme source** écrit en langage de haut niveau en un **programme objet** en langage de bas niveau.
- Le compilateur est lui-même un programme important et volumineux.

- Le travail du compilateur se divise en plusieurs phases :
  - **analyse lexicale** (recherche des mots-clés)
  - **analyse syntaxique** (vérification de la syntaxe)
  - **analyse sémantique** (vérification de la sémantique)
  - **génération du code objet**

Structure d'un langage de haut niveau

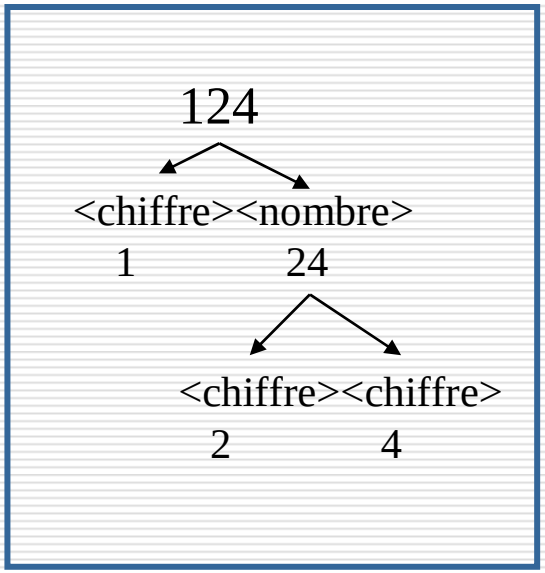
- Un langage de haut niveau s'appuie sur
  - un **alphabet** : symboles élémentaires disponibles (caractères, chiffres, ponctuations)
  - des **identificateurs** : groupe de symboles de l'alphabet (A1)
  - des **phrases ou instructions** : séquences de noms et de symboles formés selon la syntaxe du langage (A1 = 3;)
- Un programme est une suite de phrases du langage, respectant la syntaxe du langage.

Structure d'un langage de haut niveau

- Il faut exprimer la syntaxe du langage. On utilise pour cela la **notation de BACKUS-NAUR (BNF)**

- < objet du langage > **::=** <objet du langage> | symbole
  - | représente une alternative
  - < > entoure les objets du langage

- <nombre> ::= <chiffre> | <chiffre><nombre>
- <chiffre> ::= 0 | 1| 2 | 3 | 4| 5 | 6 | 7 | 8 | 9



EXEMPLE

```
PROGRAM Z
    INT A
    INT B
DEBUT
    A := 5
    B := A / 2
FIN
```

<programme> ::= PROGRAM <identificateur><corps de programme>

<corps de programme> ::= <suite de declarations> DEBUT <suite d'affectations> FIN

<suite de declarations> ::= <declaration> | <declaration><suite de declarations>

<declaration> ::= INT <identificateur>

<suite d'affectations> ::= <affectation> | <affectation><suite d'affectations>

<affectation> ::= <identificateur > := <terme> | <terme> <operateur><terme>

<terme> ::= <entier> | <identificateur>

<operateur> ::= + | - | * | /

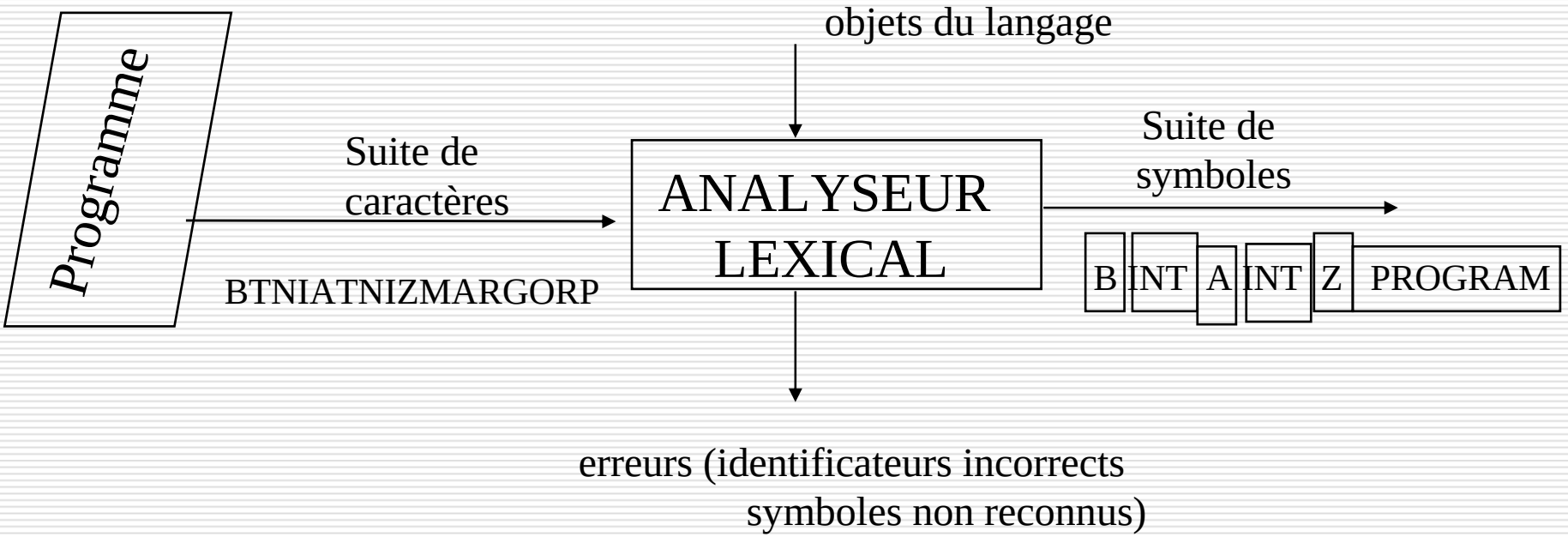
<identificateur> ::= <lettre>|<lettre><chiffre>

<entier> ::= <chiffre> | <chiffre><entier>

<lettre> ::= A |B| C| D | E....| X | Y| Z

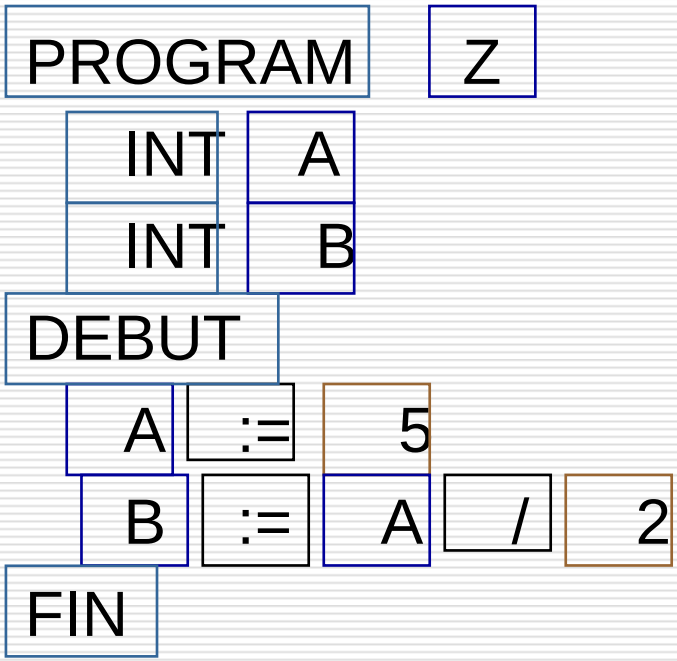
<chiffre> ::= 0 |1 | 2 | 3 | 4...| 9

ANALYSE LEXICALE



- Rôle de l'analyse lexicale
  - reconnaître dans la suite de caractères que constitue un programme les objets du langage
  - éliminer le "superflu" (espaces, commentaires)

l'ANALYSE LEXICALE : Exemple



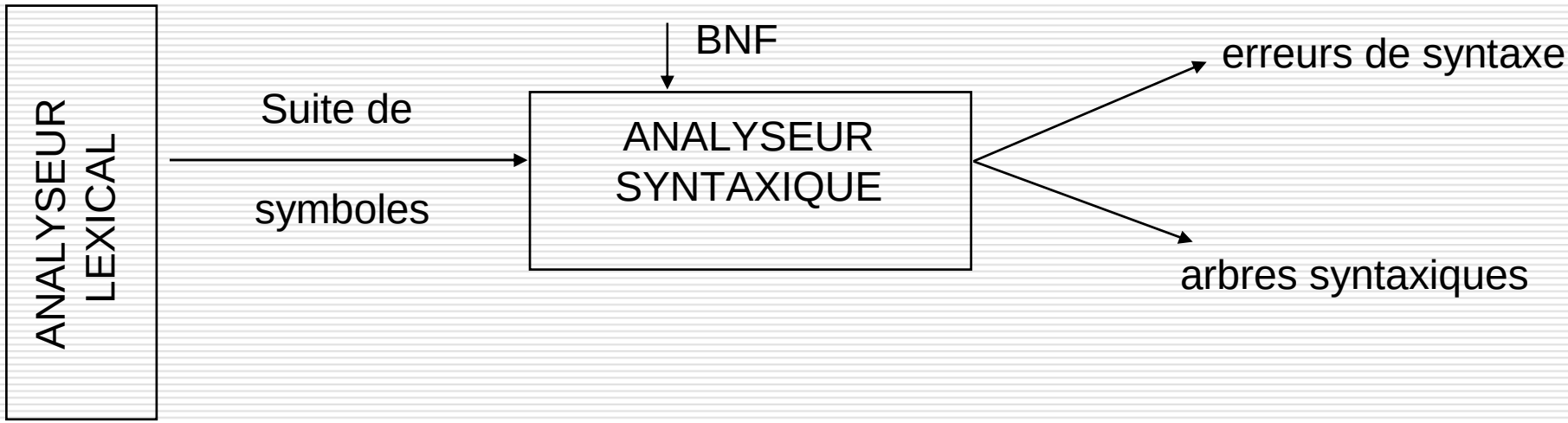
entier

Mot clé

identificateur

opérateur

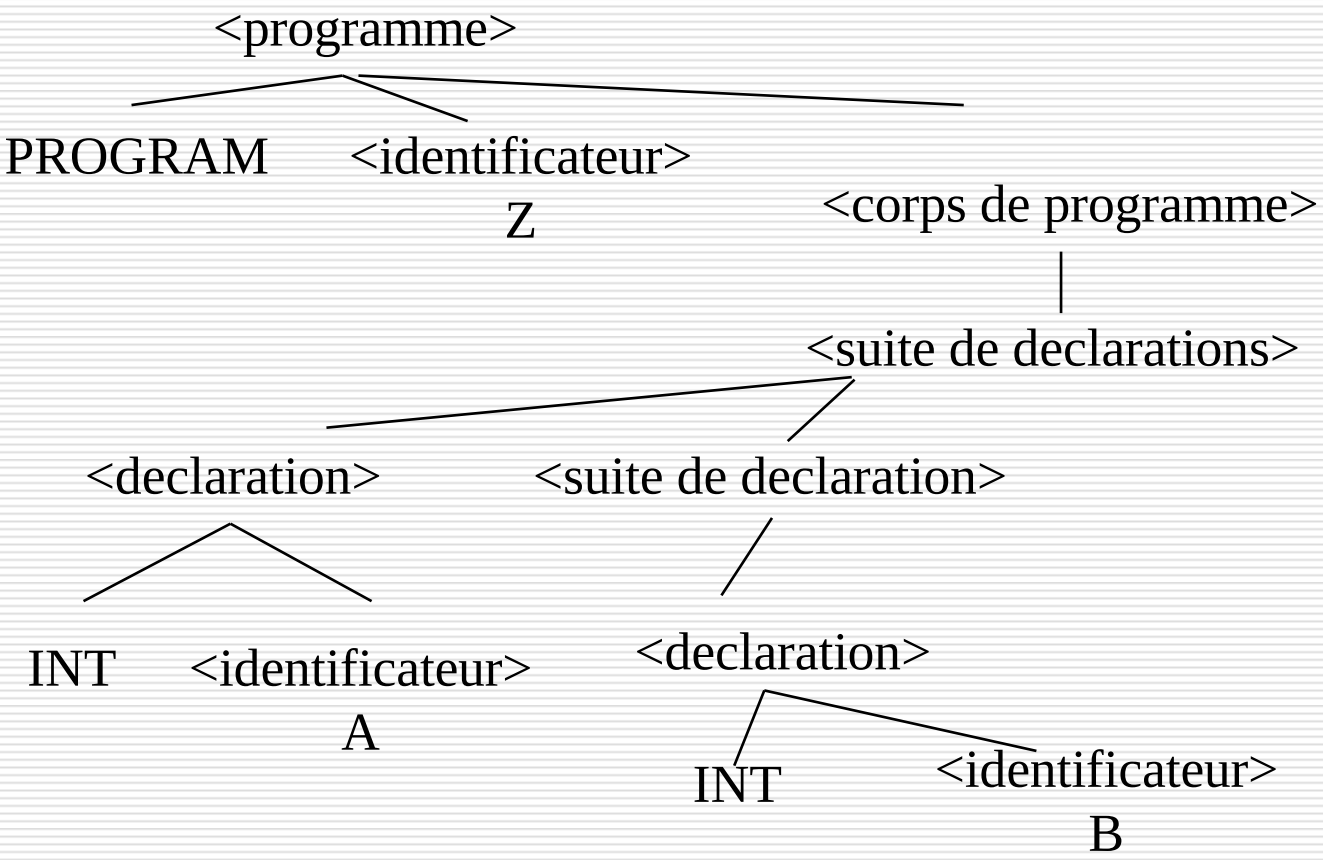
ANALYSE SYNTAXIQUE



- Rôle de l'analyse syntaxique
  - reconnaître si la suite de symboles issue de l'analyse lexicale respecte la syntaxe du langage
  - construction à partir des BNF de **l'arbre syntaxique** correspondant au programme analysé

Arbre Syntaxique : Exemple

PROGRAM Z
INT A
INT B

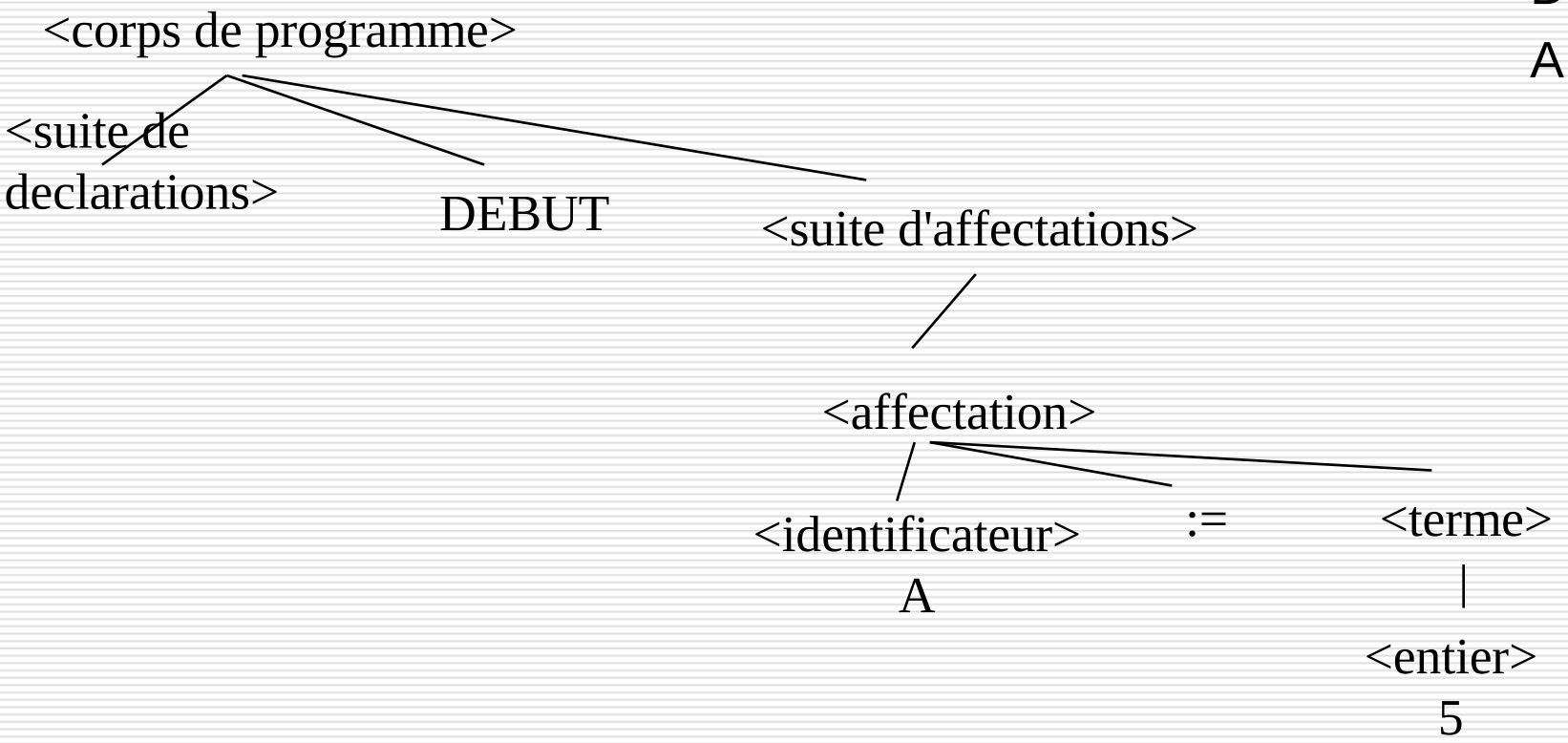


<programme> ::= PROGRAM <identicateur><corps de programme>
<corps de programme> ::= <suite de declarations> DEBUT <suite d'affectations> FIN
<suite de declarations> ::= <declaration> | <declaration><suite de declarations>
<declaration> ::= INT <identificateur>

Arbre Syntaxique : Exemple

DEBUT

A := 5



<corps de programme> ::= <suite de declarations> DEBUT <suite d'affectations> FIN

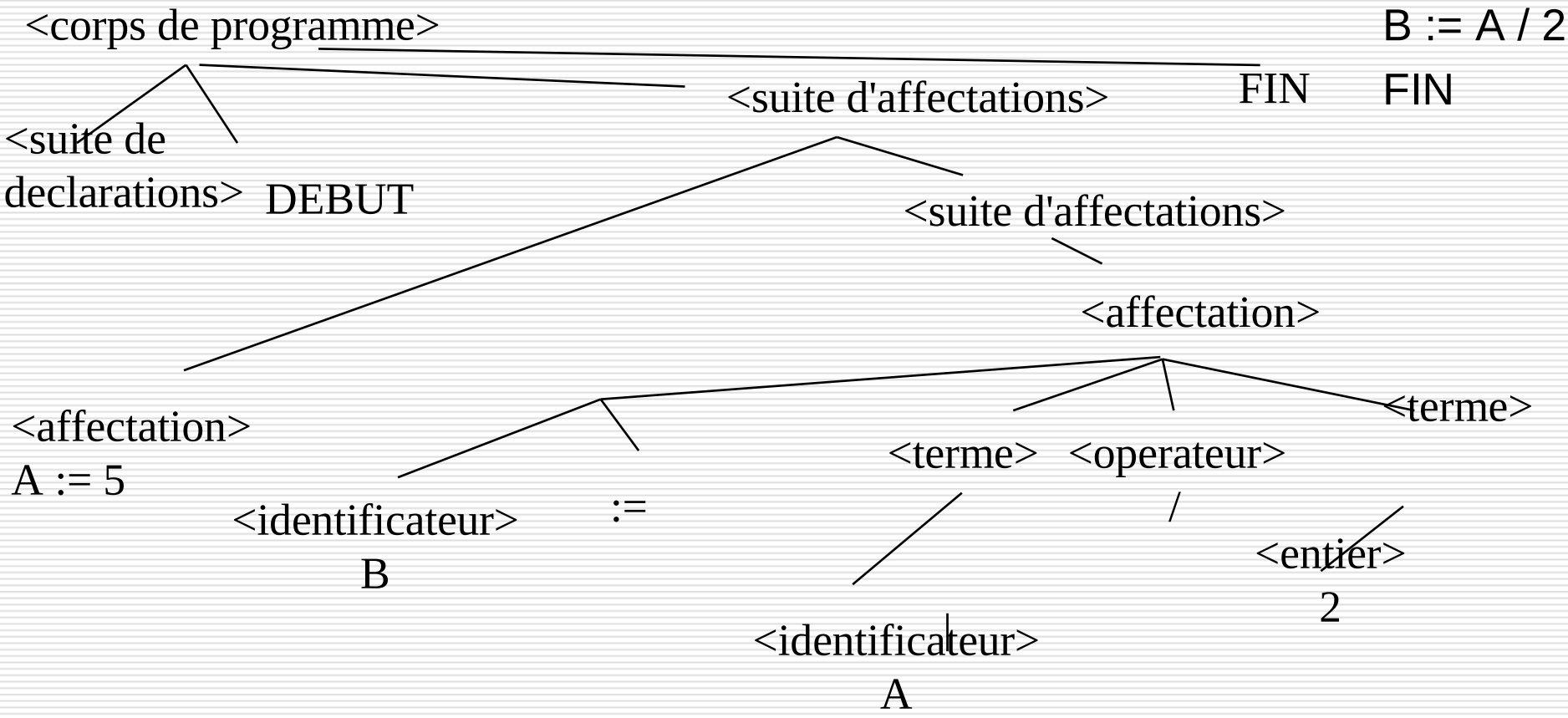
<suite d'affectations> ::= <affectation> | <affectation><suite d'affectations>

<affectation> ::= <identificateur > := <terme> | <terme> <operateur><terme>

<terme> ::= <entier> | <identificateur>

<operateur> ::= + | - | * | /

Arbre Syntaxique : Exemple



<corps de programme> ::= <suite de declarations> DEBUT <suite d'affectations> FIN

<suite d'affectations> ::= <affectation> | <affectation><suite d'affectations>

<affectation> ::= <identificateur > := <terme> | <terme> <operateur><terme>

<terme> ::= <entier> | <identificateur>

<operateur> ::= + | - | * | /

ANALYSE SEMANTIQUE

- Rôle de l'analyse sémantique
  - trouver le sens et la signification des différentes phrases du langage
    - quels sont les objets manipulés et leurs propriétés ?
      - type, durée de vie, taille, adresse
    - contrôler la cohérence dans l'utilisation des objets :
      - erreur de type, absence de déclarations, déclarations multiples, déclarations inutiles, expressions incohérentes

ANALYSE SEMANTIQUE

- exemple : signalisation d'une erreur sémantique

```
float i;                          -- i, indice de parcours de tableau
    réel
for (i=1 to 5)
loop
      tab(i)=valeur;
end loop
```

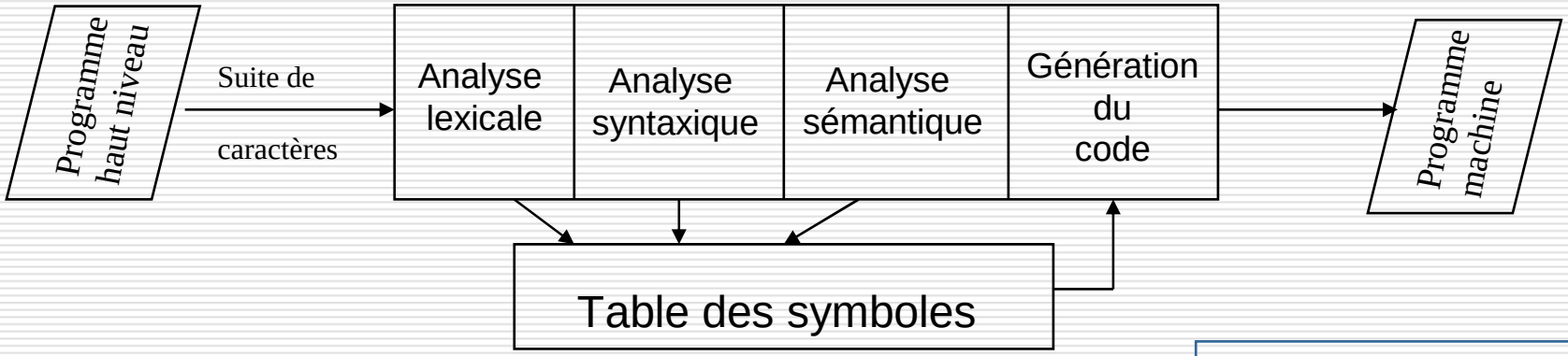
- int A
  int B
  A := 5
  B := A/2 résultat réel affecté à un entier

Table des symboles ou dictionnaire des variables

- Le compilateur manipule une table des symboles qui contient toutes les informations sur les propriétés des objets du programme
- La table est construite durant les 3 phases d'analyse lexicale, analyse syntaxique et analyse sémantique.

nom	type	taille	adresse
A	entier	4 octets	$(0)_H$
B	entier	4 octets	$(4)_H$

GENERATION DU CODE



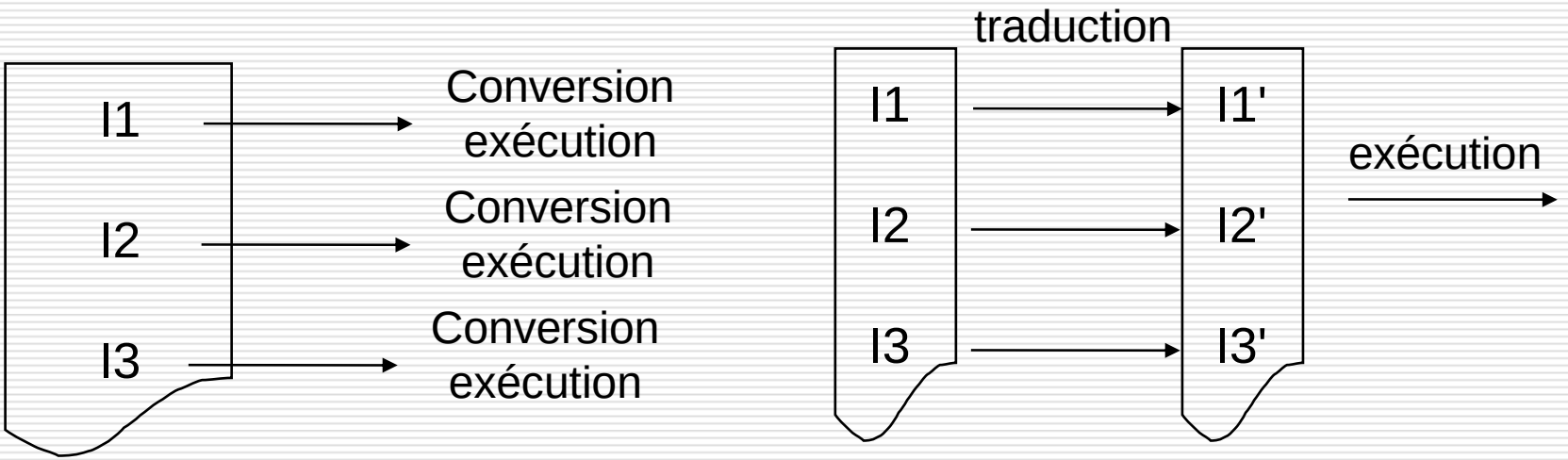
- La génération de code est l'étape ultime de la compilation.
- Elle consiste à produire le code machine équivalent du code en langage haut niveau. Ce code machine est qualifié de relogeable car les adresses dans ce code sont calculées à partir de 0

$(0)_H$
$(4)_H$
$(8)_H$ 00000 000 0001 $(4)_H$
$(C)_H$ 00000 000 0001 $(4)_H$
$(10)_H$ 00001 001 0001 $(0)_H$

$(14)_H$ 00000 000 0001 $(2)_H$
$(18)_H$ 00001 001 0001 $(4)_H$
$(1C)_H$ 00000 000 0001 $(6)_H$
$(20)_H$ 00001 000 0001 $(8)_H$

COMPILATEUR / INTERPRETEUR

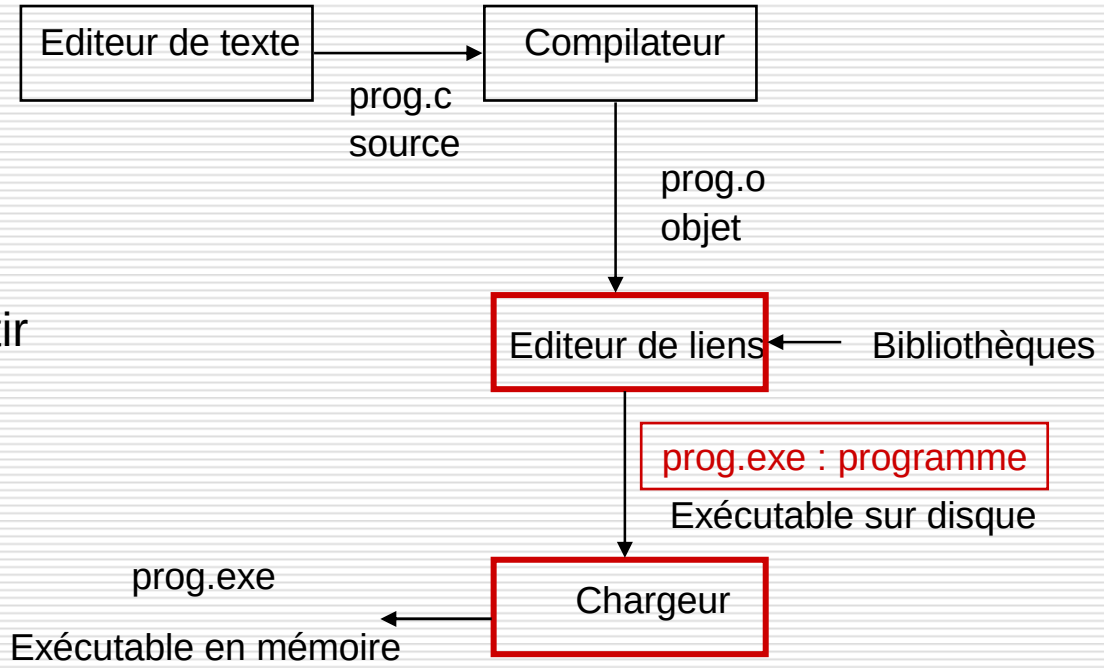
- Interprétation : conversion et exécution de chaque instruction les unes derrière les autres
- Compilation : traduction de toutes les instructions puis exécution de la traduction



La chaîne de production de programmes : éditions des liens et chargement

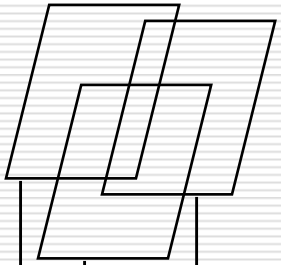
Cette chaîne désigne l'ensemble des étapes nécessaires à la construction d'un programme exécutable à partir d'un programme dit source :

- Compilation
- Edition des liens
- Chargement



Le Développement d'un "gros programme"

gros programme



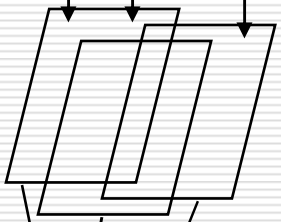
modules utilisateurs

COMPILATEUR

Compilation séparée des modules

programme objet

modules bibliothèques



modules objets utilisateurs

EDITEUR DE LIENS

programme exécutable

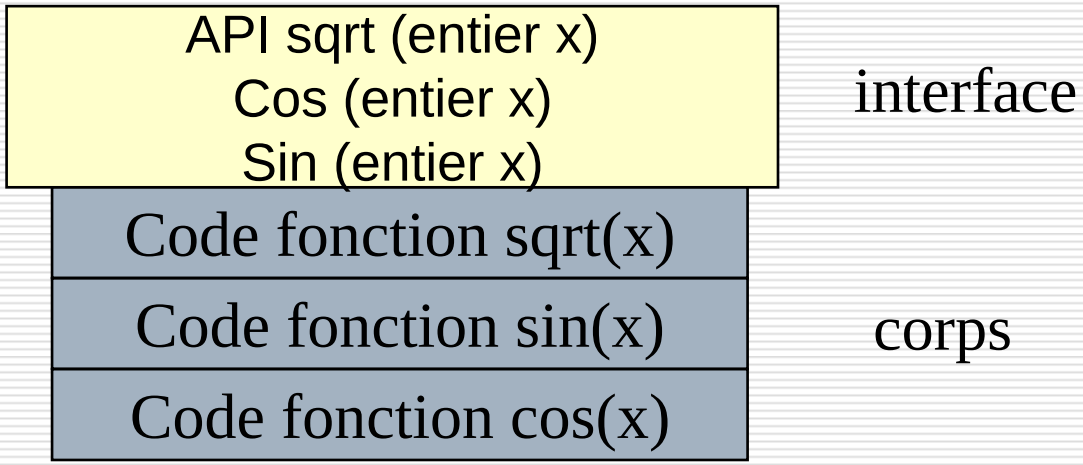
programme exécutable

ROLE DE L'EDITEUR DE LIENS

- Un éditeur de liens est un logiciel qui permet de combiner plusieurs modules objet obtenus par compilation séparée pour construire un seul programme exécutable
  - modules objets utilisateur
  - modules objets prédéfinis dans des bibliothèques
    - fonctions interfaces des appels systèmes
    - fonctions mathématiques
    - fonctions graphiques
    - etc...

Notion de bibliothèques (librairie)

- Une bibliothèque logicielle est un ensemble de fonctions compilées regroupées dans un fichier.
  - Regroupées par thème (mathématiques, graphiques, fonctions systèmes)
  - Prédéfinies et usuelles : le programmeur n'a pas à réécrire le code; il utilise la fonction fournie (par exemple, SQRT(), Line()…)



Exemple

```
PROGRAM Z
    INT A
    INT B
    INT C
DEBUT
    A := 5
    B := A / 2
    C := SQRT(B)
    EMPILER(C)
    IMPRIMER (C)
FIN
```

Module principal utilisateur

```
Module Gestion_Pile
INT Pile[10];
INT haut := 0;
Export EMPILER, DEPILER

procedure EMPILER(x)
debut
    Pile(haut):=x;
    haut := haut + 1;
fin
procedure DEPILER(x)
debut
    haut :=haut - 1;
    x := Pile(haut);
fin
```

Module Pile utilisateur

Code fonction cos(X)
Code fonction sin(X)
Code fonction SQRT(X)

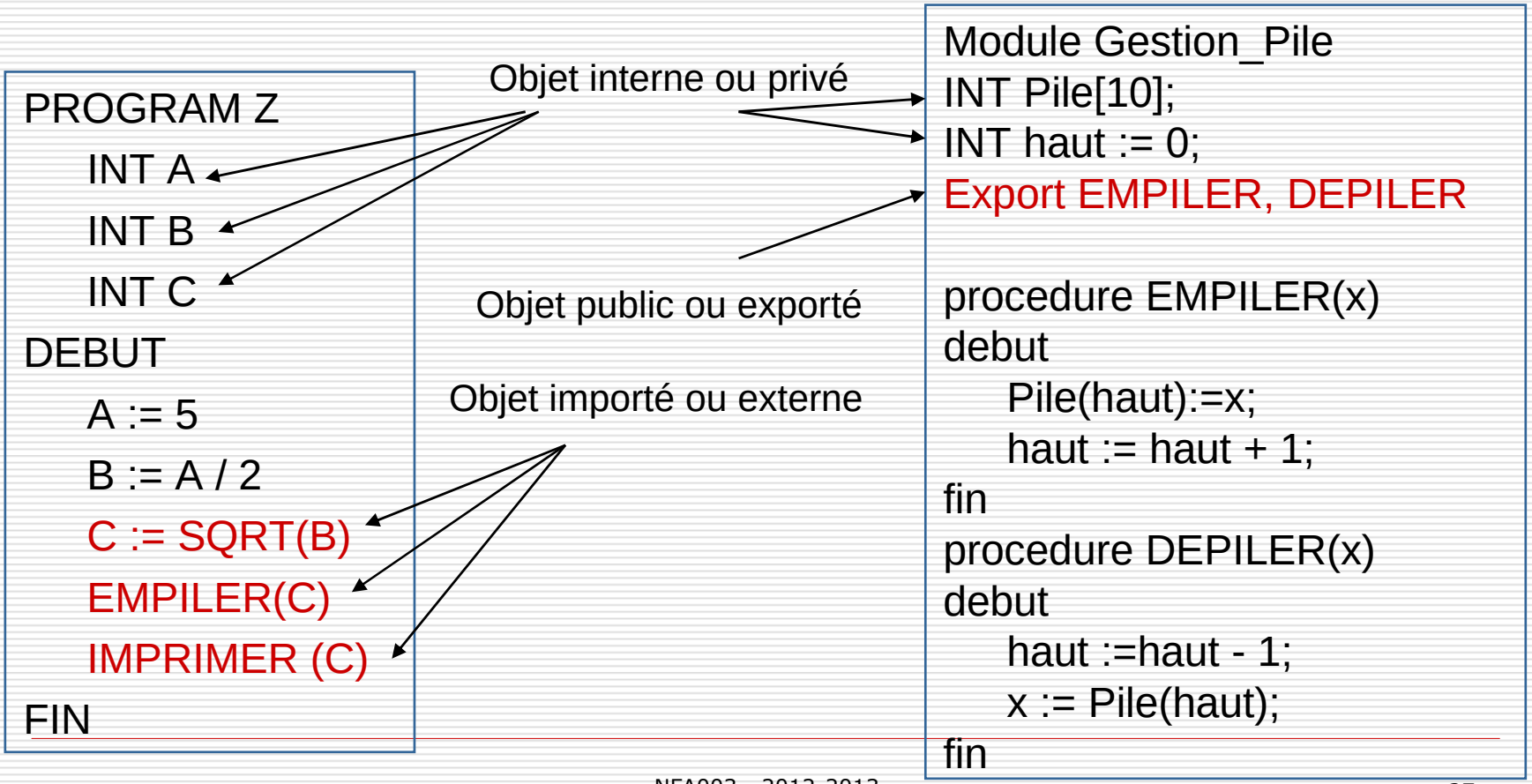
Bibliothèque mathématique

Code fonction IMPRIMER(X)
Code fonction LIRE(X)

Bibliothèque système entrées-sorties

Les types d'objets des modules

- Un module comprend trois catégories d'objets :
  - objet interne au module, inaccessible de l'extérieur
  - objet interne au module mais accessible de l'extérieur (**objet exporté ou public**)
  - objet n'appartenant pas au module, mais utilisé par le module (**objet importé ou externe**)



NOTION DE LIENS

- Le compilateur recense dans chaque module les objets privés, les objets exportés et les objets importés.

Pour chaque objet rencontré, selon sa catégorie :

- si l'objet est interne et privé, il associe une adresse à l'objet dans la table des symboles
- si l'objet est interne et exporté, il lui associe une adresse à l'objet dans la table des symboles et publie cette adresse sous forme d'un lien utilisable <LU, nom_objet, adresse dans le module>.
- si l'objet est externe (importé), il ne connaît pas l'adresse à l'objet. Il demande à obtenir cette adresse sous forme d'un lien à satisfaire <LAS, nom_objet, adresse_inconnue>.

Exemple

```
PROGRAM Z
    INT A
    INT B
    INT C
DEBUT
    A := 5
    B := A / 2
    C := SQRT(B)
    EMPILER(C)
    IMPRIMER (C)
FIN
```

Module principal utilisateur

compilation

<LAS SQRT>
<LAS EMPILER>
<LAS IMPRIMER>

Code objet
Principal.o
40 octets

Nom	type	taille	adress
A	entier	4	(0)
B	entier	4	(4)
C	entier	4	(8)
SQRT			?
EMPILER			?
IMPRIMER			?

Exemple

```
Module Gestion_Pile
INT Pile[10];
INT haut := 0;
Export EMPILER, DEPILER

procedure EMPILER(x)
debut
     Pile(haut):=x;
     haut := haut + 1;
fin
procedure DEPILER(x)
debut
     haut :=haut - 1;
     x := Pile(haut);
fin
```

Module Pile utilisateur

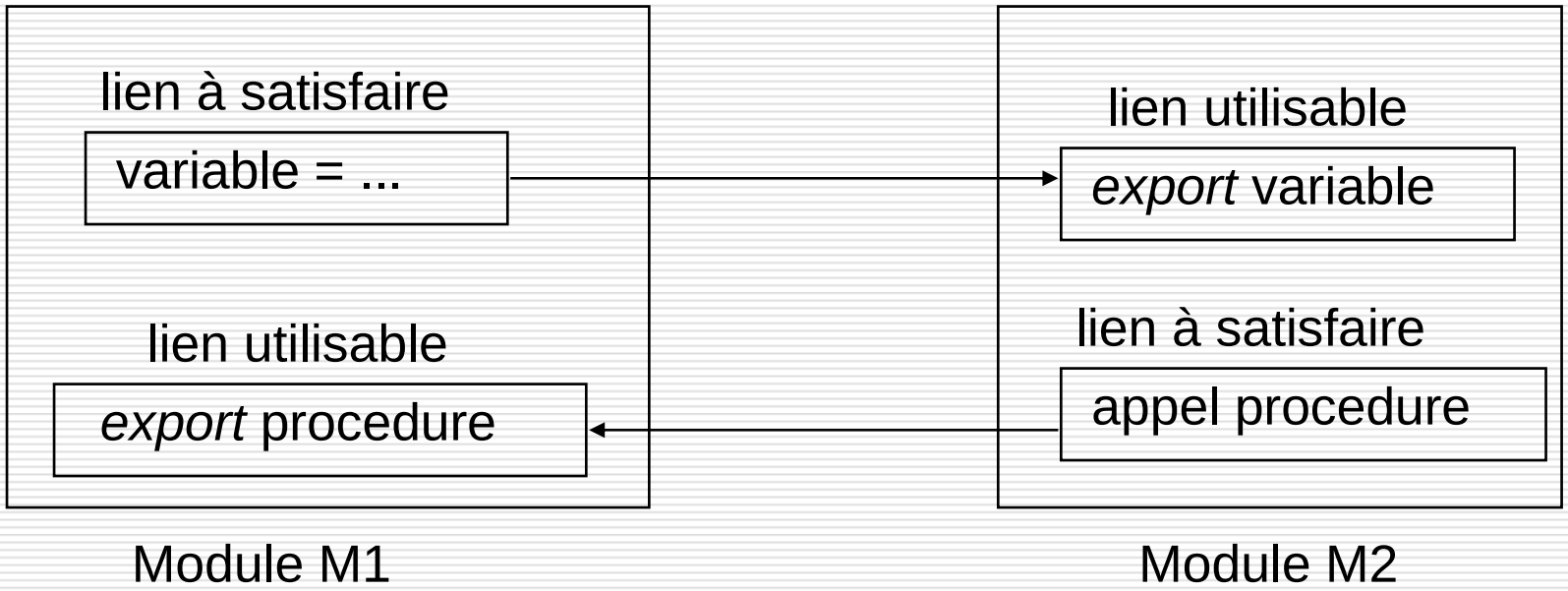
compilation

<LU EMPILER 44 >
<LU DEPILER 60>

Code objet
pile.o
76 octets

nom	type	taille	adresse
Pile	entier	40	(0)
haut	entier	4	(40)
EMPILER		16	(44)
DEPILER		16	(60)

ROLE DE L'EDITEUR DE LIENS



☞ L'éditeur de liens doit construire le programme exécutable final à partir des modules objet entrant dans sa composition.

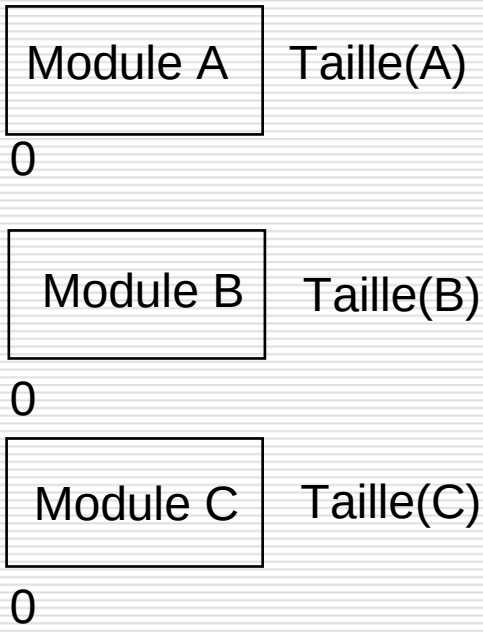
☞ Il procède en trois étapes :

1. *Construction de la carte d'implantation du programme*
2. *Construction de la table des liens utilisables*
3. *Construction du programme exécutable final*

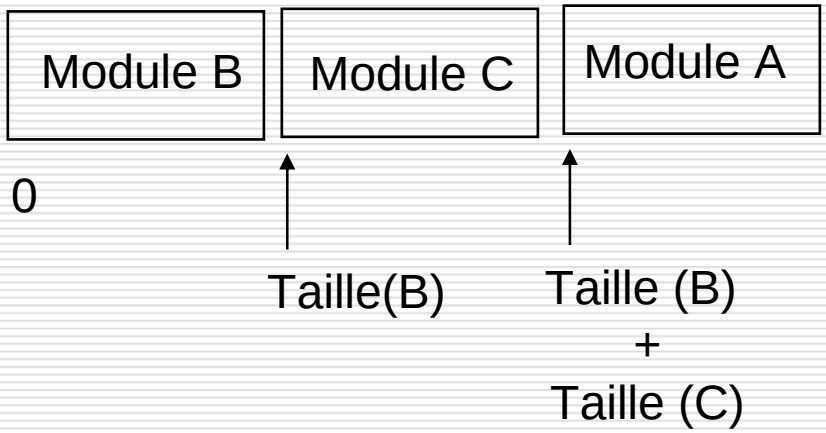
Construction de la carte d'implantation

- Détermination des adresses d'implantation de chaque module utilisateur du programme, placés les uns derrière les autres

Compilation : modules relogeables



Carte d'implantation



Construction de la table des liens

- Recenser l'ensemble des liens existants et leur associer leur adresse dans la carte d'implantation

Pour chaque lien <nom de lien> dans chaque module

Si (<nom de lien> n'est pas dans la table)

Nom de lien	Adresse

Alors

créer une entrée <nom de lien>
si (le lien est un lien utilisable) alors associer à <nom de lien> son adresse selon la carte fsi
si (le lien est un lien à satisfaire) alors associer à <nom de lien> <adresse indefinie> fsi

sinon

si (le lien est un lien utilisable) et l'entrée existante dans la table est <adresse indefinie>

alors
associer à <nom de lien> son adresse selon la carte (résolution LAS / LU)
fsi
Fsi

fin pour

Edition des liens : exemple

Pour chaque lien <nom de lien> dans chaque module

Si (<nom de lien> n'est pas dans la table)

Alors

créer une entrée <nom de lien>
si (le lien est un lien utilisable) alors associer à <nom de lien> son adresse selon la carte fsi
si (le lien est un lien à satisfaire) alors associer à <nom de lien> <adresse indefinie> fsi

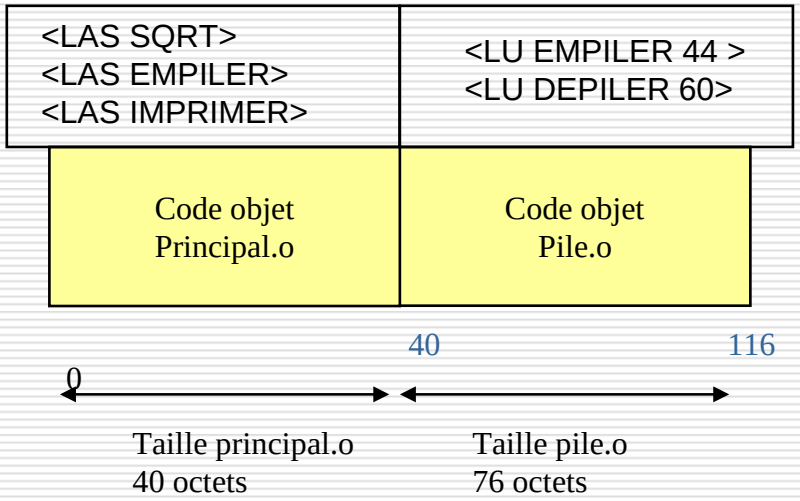
sinon

si (le lien est un lien utilisable) et l'entrée existante dans la table est <adresse indefinie>

alors
associer à <nom de lien> son adresse selon la carte (résolution LAS / LU)
fsi

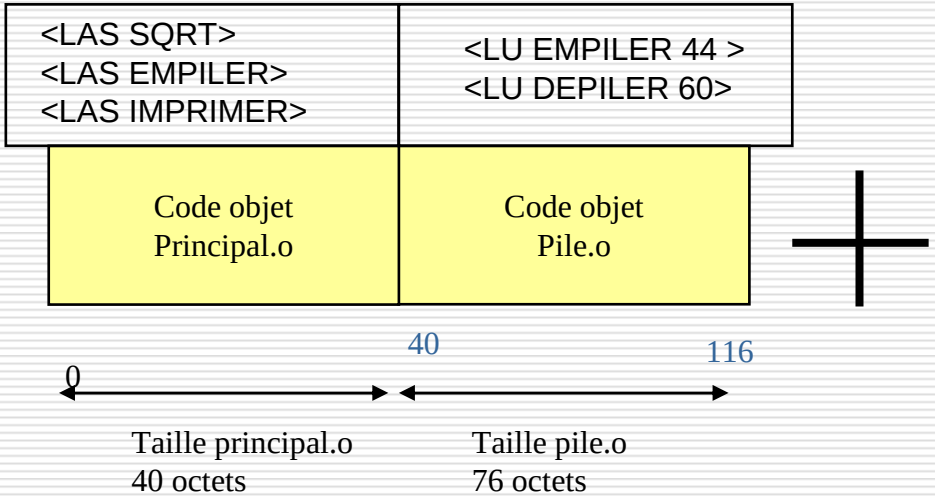
Fsi

fin pour



Nom de lien	Adresse /carte
SQRT	? Bib math
EMPILER	? 44 + 40
IMPRIMER	? Bib E/S
DEPILER	60 + 40

Construction du programme exécutable final

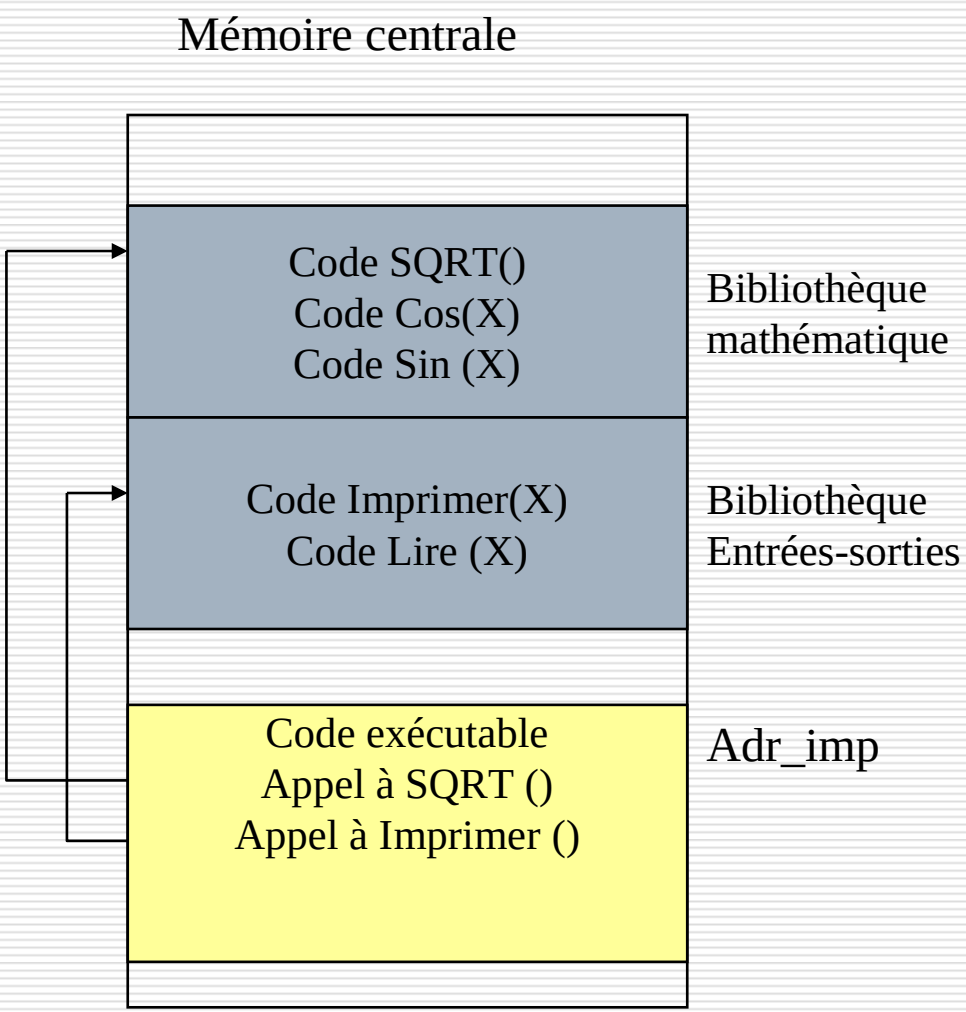


Nom de lien	Adresse /carte
SQRT	Bib math
EMPILER	44 + 40
IMPRIMER	Bib E/S
DEPILER	60 + 40

- L'éditeur de liens construit le code exécutable :
  - 1. Il remplace les occurrences des objets figurant dans les LAS par leur adresse dans la table des liens
  - 2. Il translate les adresses des objets dans les modules de la valeur de l'adresse d'implantation du module dans la carte.

Code exécutable
Appel à SQRT () ?
Appel à Imprimer () ?

Construction du programme exécutable final
Bibliothèque et chargement



- Chargement du programme en mémoire centrale
- L'outil chargeur place le programme exécutable en mémoire centrale ainsi que les bibliothèques qui lui sont utiles.
  - 1. Il résout les liens vers les bibliothèques;
  - 2. Il translate les adresses des objets dans le programme exécutable de la valeur de l'adresse d'implantation en mémoire centrale Adr_imp.

FIN
Questions ?
Cours suivant : processus et ordonnancement