La chaîne de production de programme

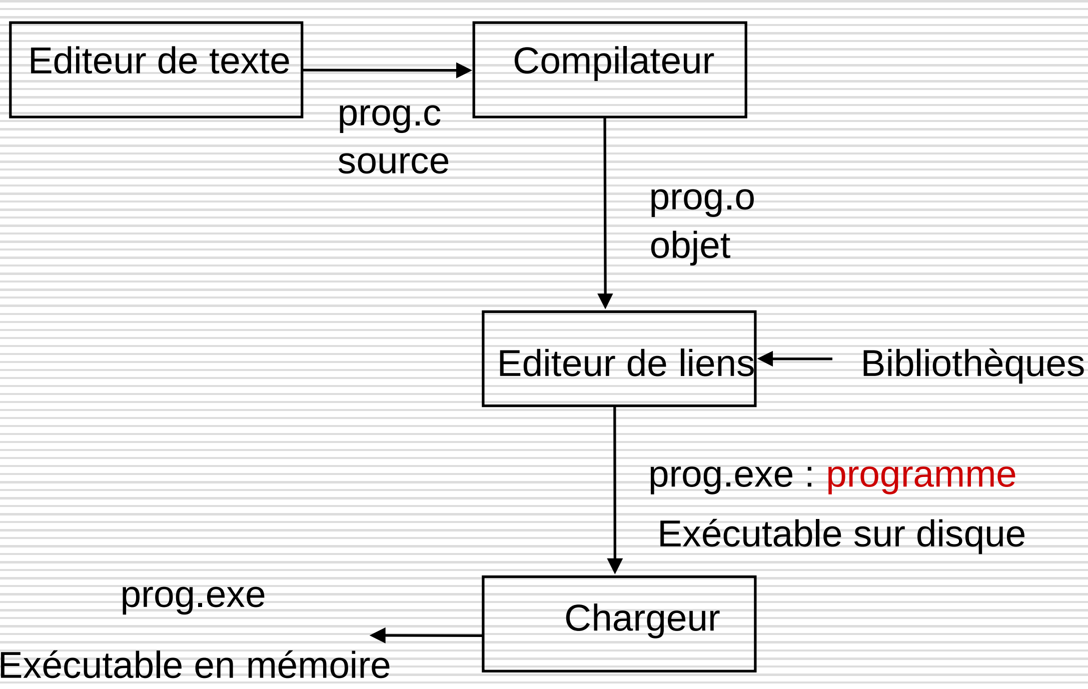
Du programme source au processus :

- Compilation
- Éditions des liens et chargement
- L’utilitaire Make

La chaîne de production de programmes

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

- La compilation
- L'édition des liens
- Le chargement

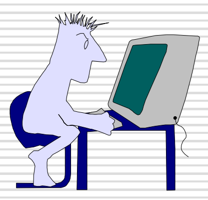


La compilation

Les niveaux de langage de programmation

le cnam

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 : 0001 0000 0001 000000000001
       0010 0000 0010 000000000001
       0110 00000000000000001000
```

Du langage de haut niveau au binaire

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.

Programme en langage de haut niveau
instructions de haut niveau

Compilateur

Niveau utilisateur

Machine physique "matérielle"

Programme en langage d'assemblage

Assembleur

processeur

0110111011111 0011
01111010001011100
10111101110111111
00111011110111011
00111111000111101

Mémoire centrale

Bus

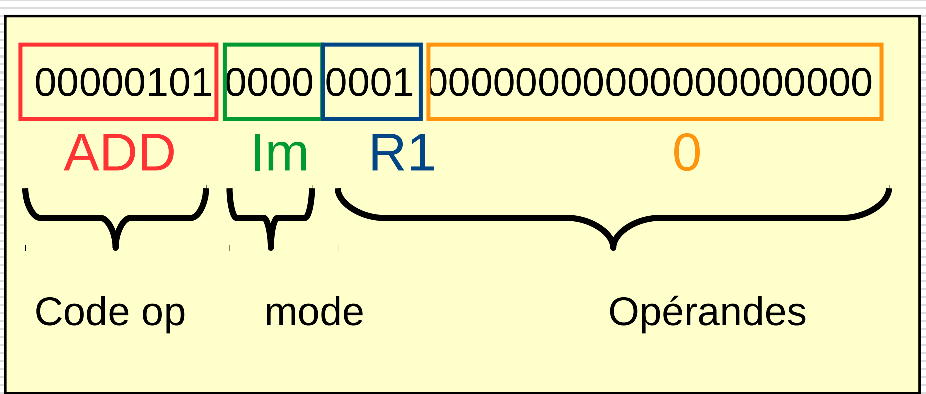
Programme à
exécuter : instructions machine
et valeurs en binaire

Du langage d'assemblage au langage binaire

le cnam

- Chaque processeur possède son propre jeu d'instructions machine (chaîne binaire). Seul ces instructions sont exécutables par le processeur (ADD, SUB, LOAD, STORE, JUMP, etc...)

- Le langage d'assemblage est l'équivalent du langage machine. Chaque champ binaire de l'instruction machine est remplacé par un mnémonique alphanumérique.



Rôle 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 :

  (1) analyse lexicale (recherche des mots-clés)

  (2) analyse syntaxique (vérification de la syntaxe)

  (3) analyse sémantique (vérification de la sémantique)

  (4) 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.

Formalisation de la syntaxe du langage

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

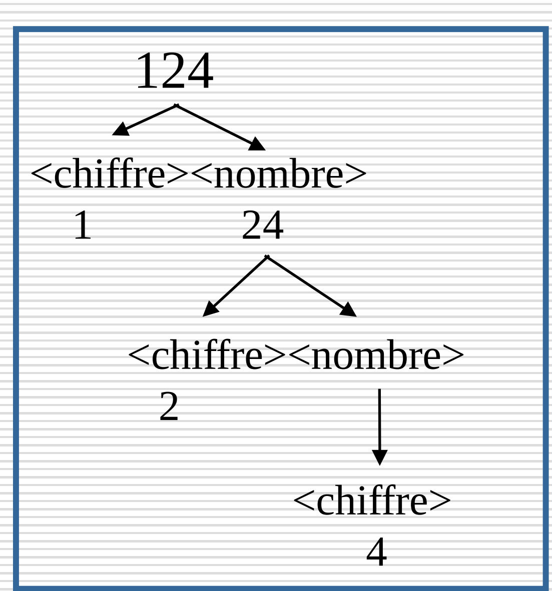
- < objet du langage > **::=** <objet du langage> | symbole

  - | représente une alternative
  - < > entoure les objets du langage

Exemple :

<nombre> ::= <chiffre> | <chiffre><nombre>

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



Un exemple de syntaxe de langage

Formalisation de la syntaxe par la notation BNF

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

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

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

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

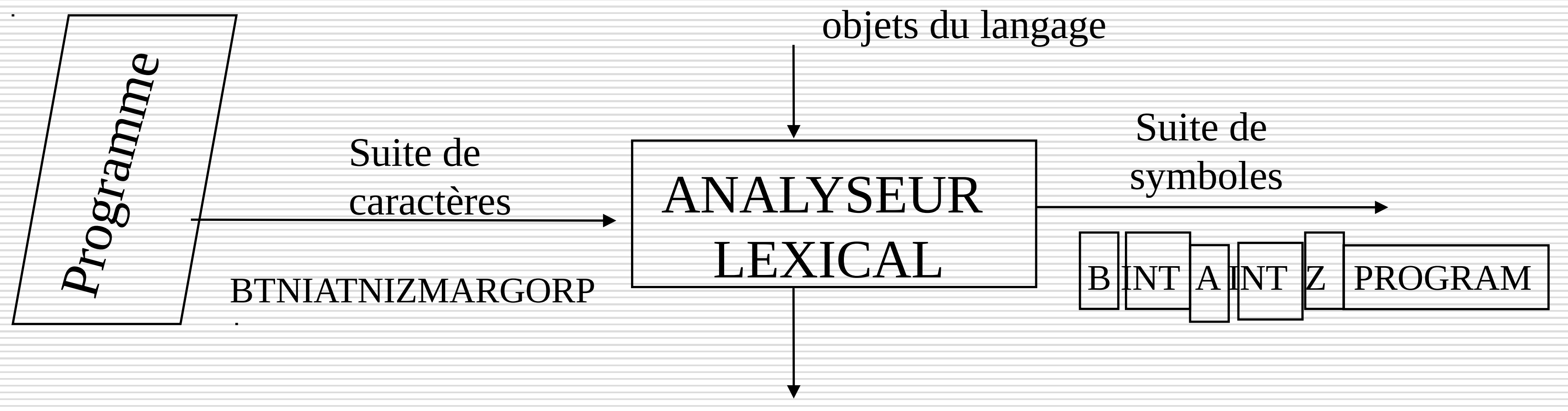
<entier> ::= <chiffre> | <chiffre><entier>

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

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

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

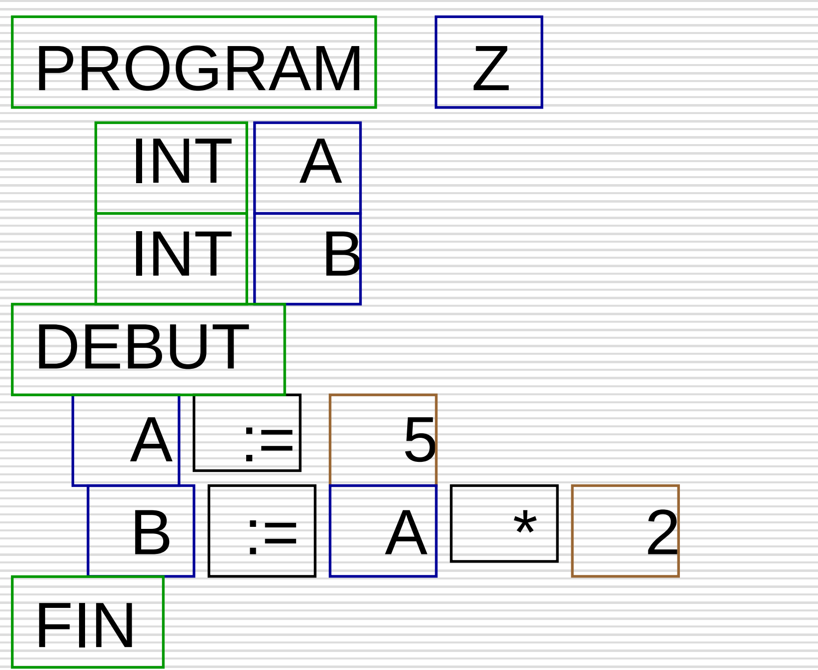
Analyse lexicale



Rôle de l'analyse lexicale

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

Analyse lexicale : un exemple



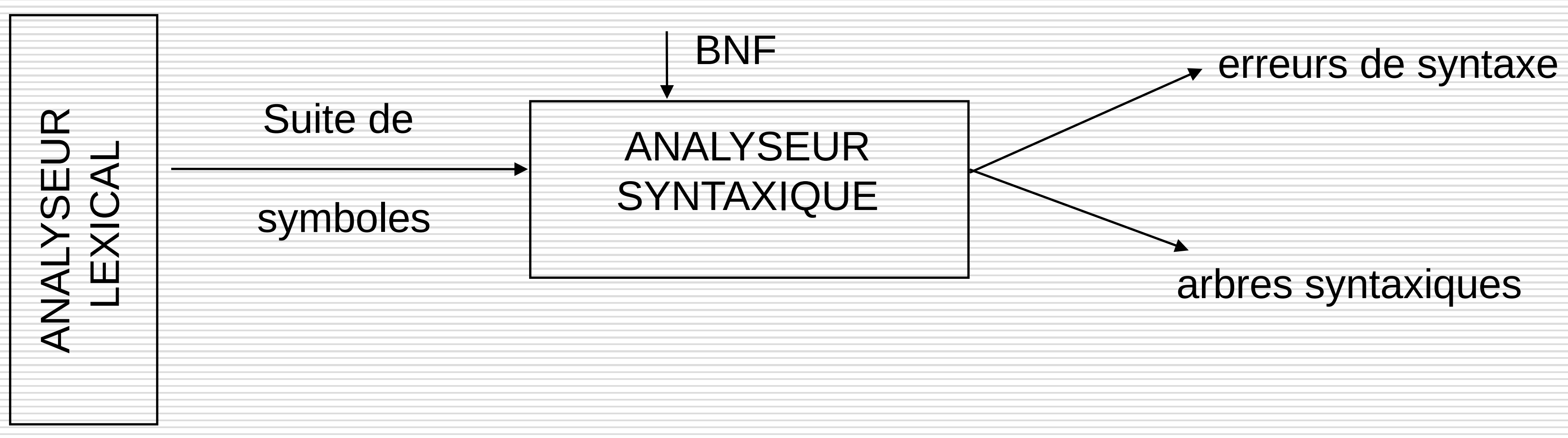
entier

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 de **l'arbre syntaxique** correspondant au programme analysé

Arbre Syntaxique : Exemple

<programme>

PROGRAM <identificateur> <corps de programme>

Z

<suite de declarations>

<declaration> <suite de declaration>

INT <identificateur> <declaration>

A

INT <identificateur>

B

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

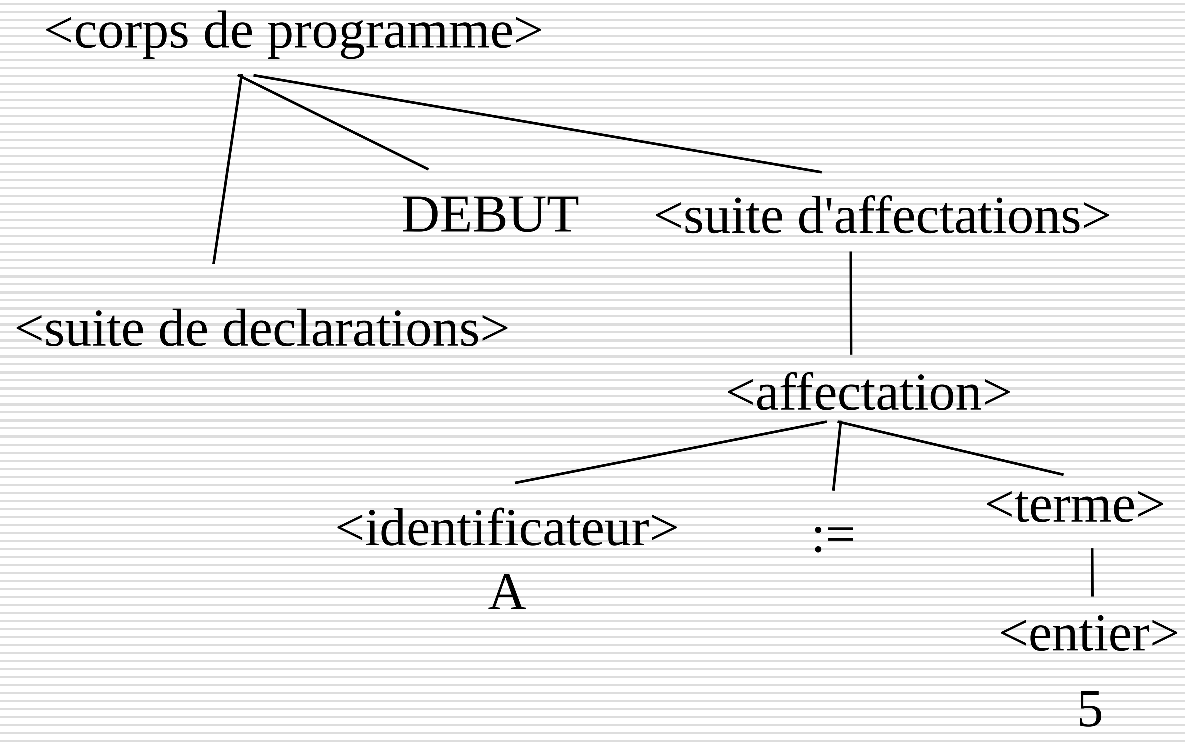
<programme> ::= PROGRAM <identicateur><corps de programme>

<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




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

<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>
  - <affectation>
    A := 5
  - <suite d'affectations>
    - <affectation>
      - <identificateur>
        B
      - :=
      - <terme>
        - <identificateur>
          A
      - <operateur>
        *
      - <terme>
        - <entier>
          2
- FIN

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

<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 sémantique

Rôle de l'analyse sémantique :

Contrôler la signification des différentes phrases du langage

A partir de la liste des objets manipulés, le compilateur connaît 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 sémantique : exemples

```
float i;
for (i=1 ; i< 5 ; i++)
{
 tab[i]=5;
}
```

Indice de parcours de boucle réel

```
int a ;
int b ;
a = 5
b = a/2 ;
```

Résultat réel affecté à un entier

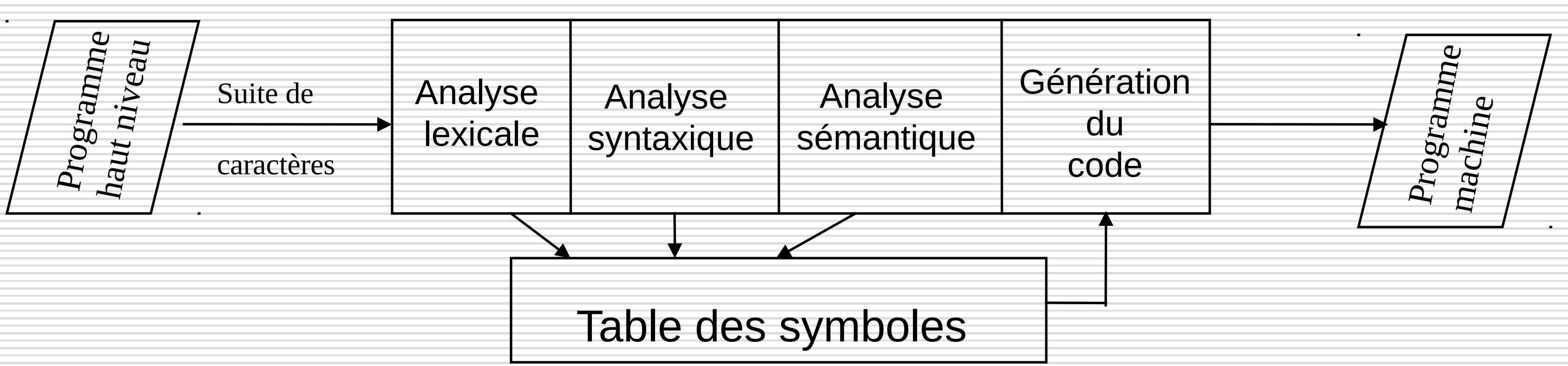
Table des symboles

- 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, d’analyse syntaxique et d’analyse sémantique.

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

Génération 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

$(0)_H$

$(4)_H$

$(8)_H$ 00000000 0000 0001 $(5)_H$

$(C)_H$ 00000001 0001 0001 $(0)_H$

$(10)_H$ 00000101 0000 0001 $(2)_H$

$(14)_H$ 00000001 0001 0001 $(4)_H$

Génération du code

Programme en langage de haut niveau → Programme en assembleur → Programme en langage machine

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

→

```
debut :  load  Im R1 5
         store D  R1 A
         mul   Im R1 2
         store D  R1 B
```

→

$(0)_H$

$(4)_H$

$(8)_H$ 00000000 0000 0001 $(5)_H$

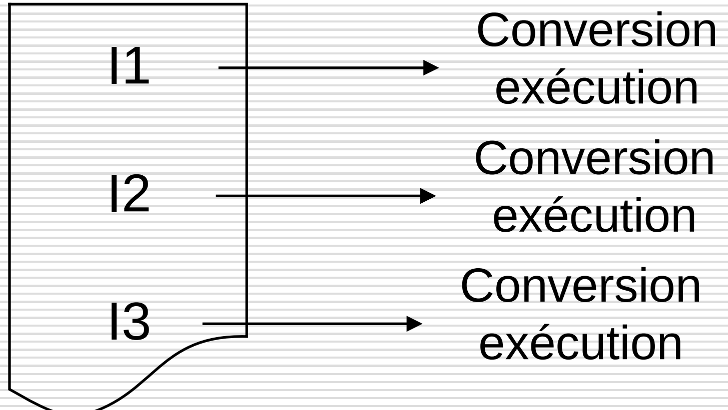
$(C)_H$ 00000001 0001 0001 $(0)_H$

$(10)_H$ 00000101 0000 0001 $(2)_H$

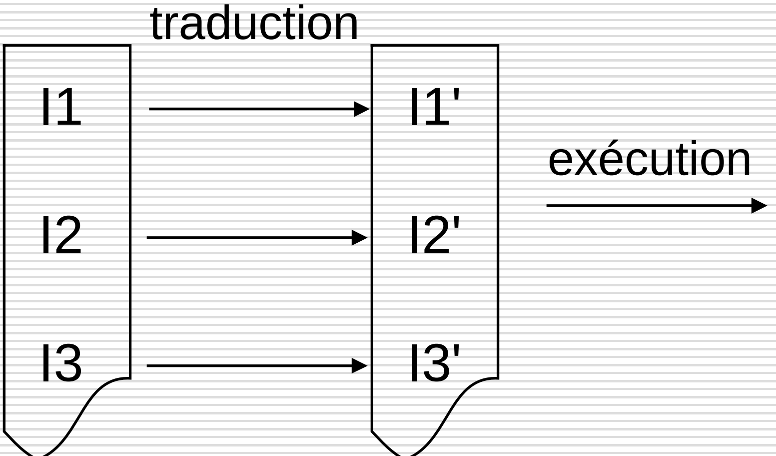
$(14)_H$ 00000001 0001 0001 $(4)_H$

Compilateur VS Interpréteur

- **Interprétation** : conversion et exécution de chaque instruction les unes derrière les autres

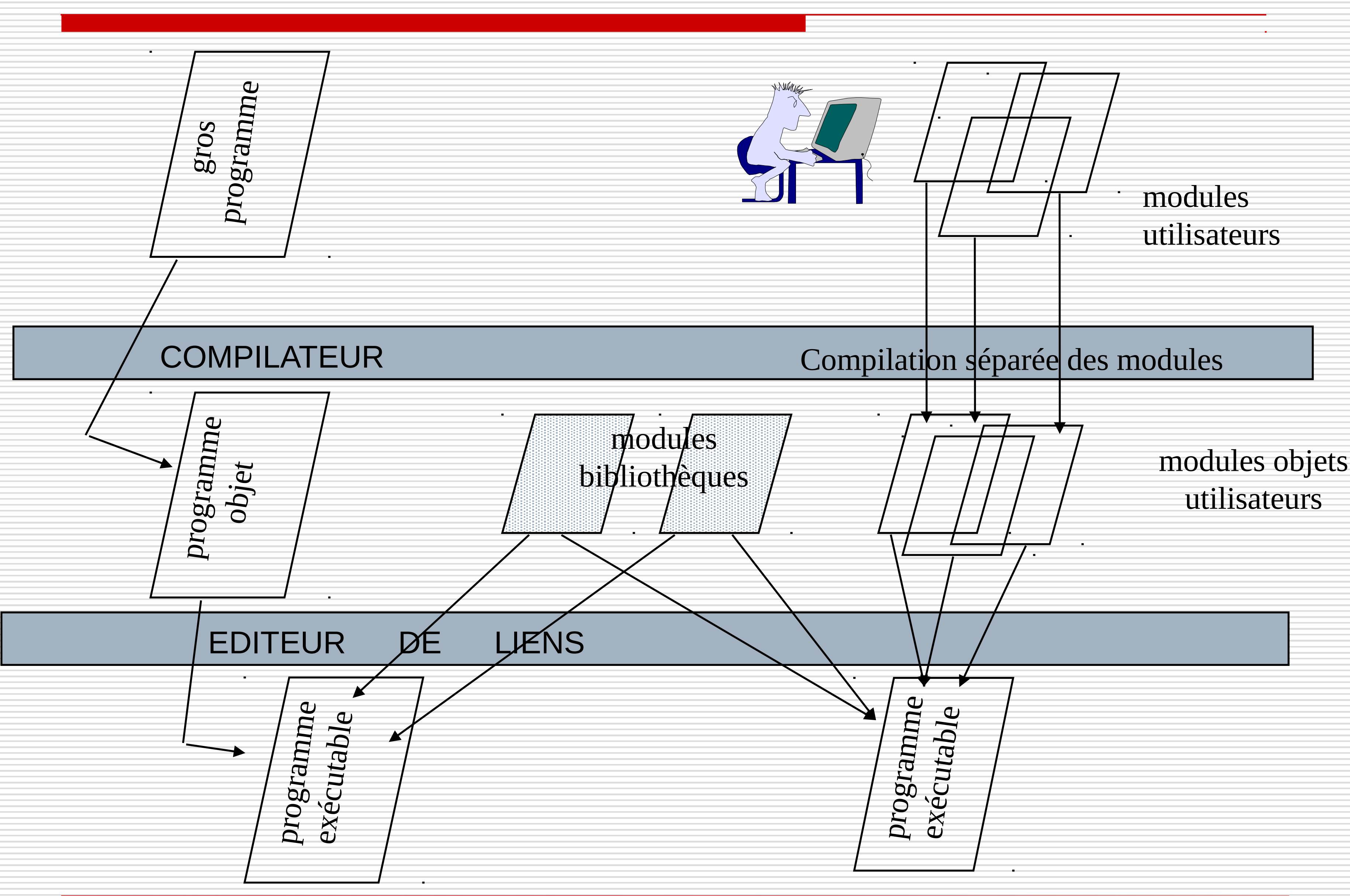


- **Compilation** : traduction de toutes les instructions puis exécution de la traduction



Edition de liens et chargement

Le développement d'un "gros programme"



Rôle de l'éditeur de liens

Un **éditeur de liens** est un logiciel qui permet de combiner plusieurs modules objets obtenus par compilation séparée pour construire un seul programme exécutable. Il combine deux sortes de modules objets :

- Les modules objets utilisateur
- Les 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.

- Elles sont regroupées par thème (mathématiques, graphiques, fonctions systèmes)
- Elles sont prédéfinies et usuelles : le programmeur n'a pas à réécrire le code; il utilise la fonction fournie (par exemple, SQRT(), Line()…)

API sqrt (entier x) Cos (entier x) Sin (entier x)	interface
Code fonction sqrt(x) Code fonction sin(x) Code fonction cos(x)	corps

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**)

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

Objet interne ou privé

Objet public ou exporté

Objet importé ou externe

```
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
```

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

Table des symboles

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

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

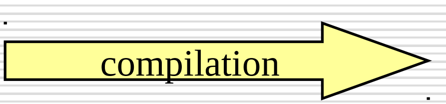
Code objet
Principal.o
40 octets

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



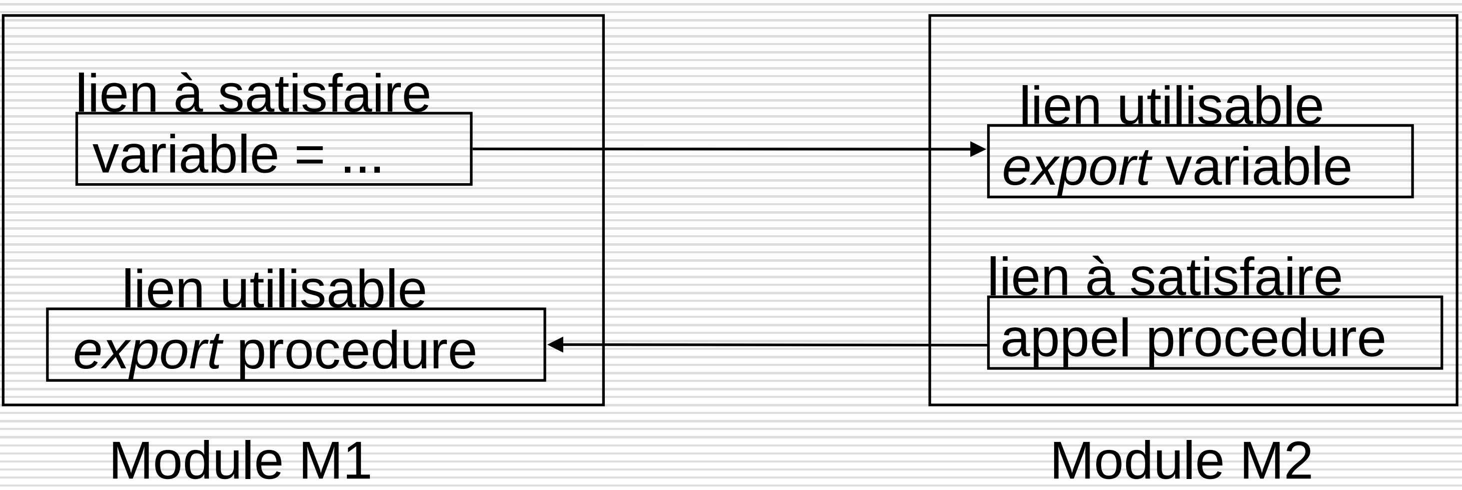
<LU EMPILER 44 >
<LU DEPILER 60>

Code objet
pile.o
76 octets

Table des symboles

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

Rôle de l'éditeur de liens



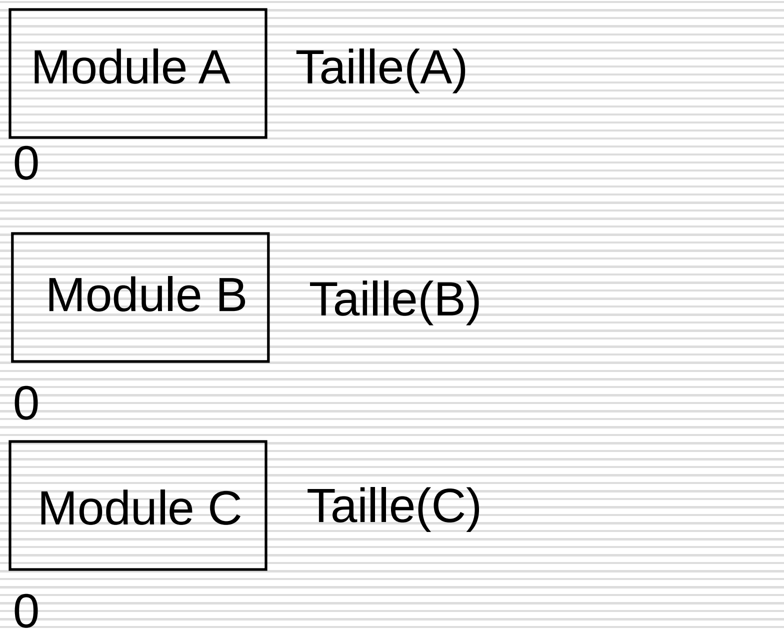
- L'éditeur de liens doit construire le programme exécutable final à partir des modules objet entrant dans sa composition.

- 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*

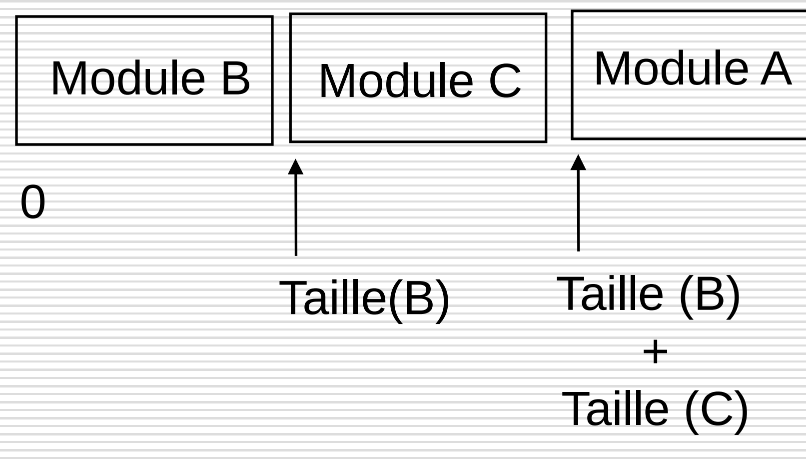
Construction de la carte d'implantation

Détermination des adresses d'implantation de chaque module utilisateur du programme en les plaçant les uns derrière les autres

Compilation : modules relogeables



Carte d'implantation



Construction de la table des liens

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

Nom de lien	Adresse

(1) **Pour** chaque lien <nom de lien> dans chaque module **faire**

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
si (le lien est un lien à satisfaire) **alors**
associer à <nom de lien> <adresse indefinie>

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)

Edition des liens : exemple

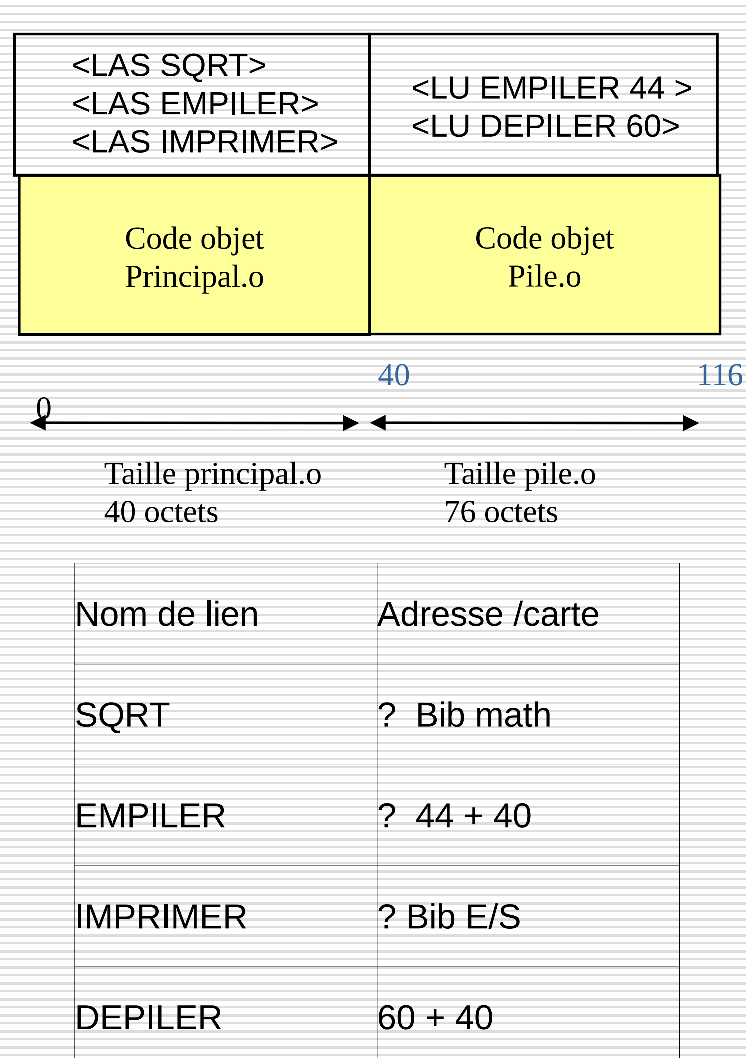
(1) Recenser l'ensemble des liens existants et leur associer leur adresse dans la carte d'implantation

(1) **Pour** chaque lien <nom de lien> dans chaque module **faire**

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
si (le lien est un lien à satisfaire) **alors**
associer à <nom de lien> <adresse indefinie>

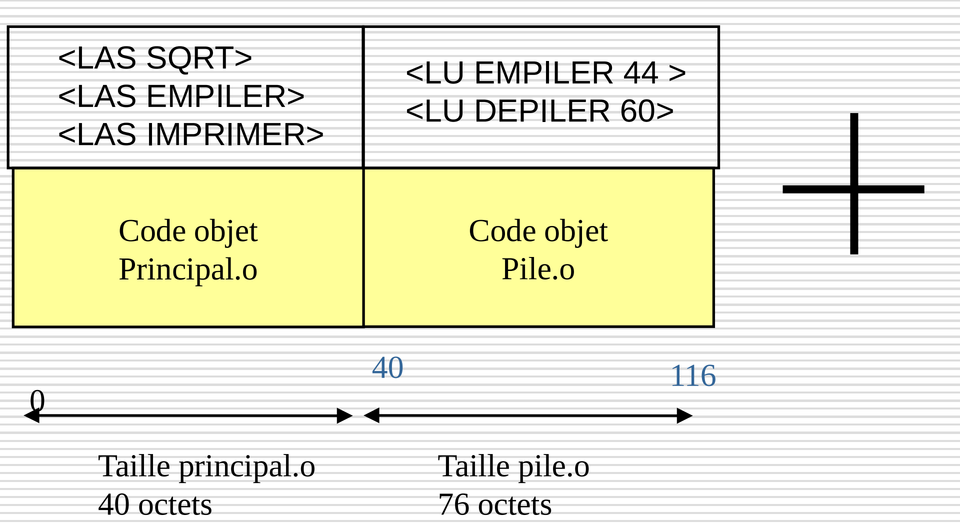
sinon

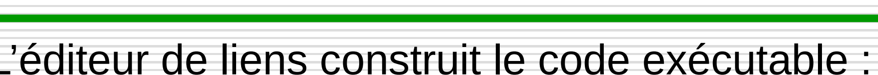
si (le lien est un lien utilisable) et l'entrée existante dans la table est <adresse indefinie> **alors**
associer à <nom de lien> son adresse selon la carte
(résolution LAS / LU)



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

Construction de l'exécutable final





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

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

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

(1) Il translate les adresses des objets dans les modules de la valeur de l'adresse d'implantation du module dans la carte.

Bibliothèque et chargement

Mémoire centrale

Mémoire centrale	
Code SQRT() Code Cos(X) Code Sin (X)	Bibliothèque mathématique
Code Imprimer(X) Code Lire (X)	Bibliothèque Entrées-sorties
Code exécutable Appel à SQRT () Appel à Imprimer ()	Adr_imp

A) Chargement du programme en MC

A) 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.

Un outil pour la construction de programme : Make

Modification de modules
Reconstruire le
programme exécutable

Fichier makefile

COMPILATEUR

modules
bibliothèques

MAKE

EDITEUR DE LIENS

programme
exécutable

L'outil Make

- Le **make** est un outil qui permet de construire un programme en n'exécutant que les opérations de compilation et éditions de liens nécessaires

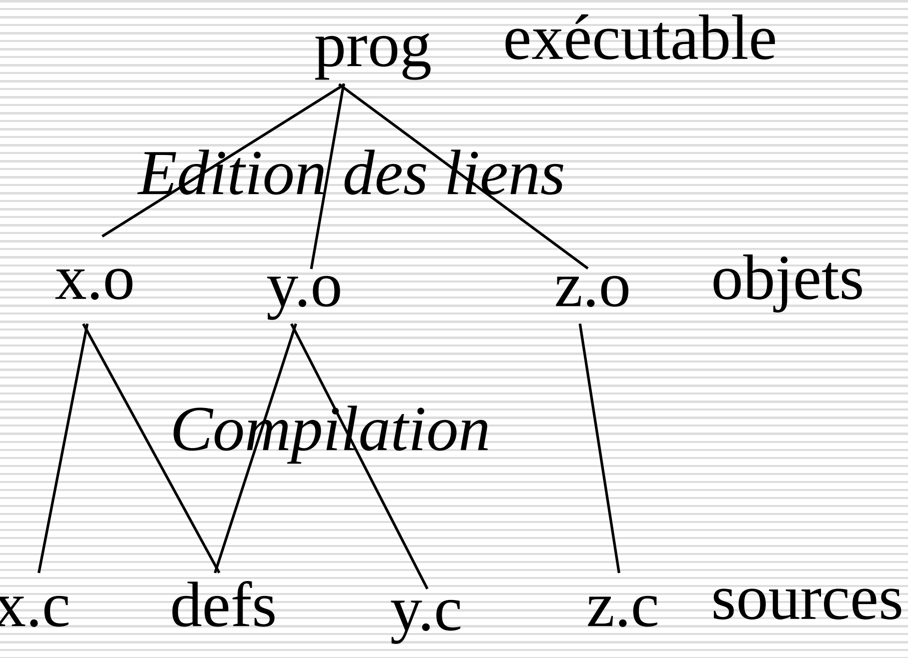
- Le make utilise trois sources d'informations
  - *un fichier de description : le **Makefile***
  - *les noms et les dates de dernières modifications des fichiers*
  - *des règles implicites liées aux suffixes des noms de fichiers*

Le fichier Makefile

Le fichier makefile décrit

- les dépendances existantes entre les modules intervenant dans la construction d'un exécutable (Graphe de dépendance)
- les opérations à lancer pour construire l'exécutable

Le fichier Makefile



Makefile

```
prog : x.o y.o z.o
    gcc x.o y.o z.o -o prog
x.o : defs x.c
    gcc -c x.c
y.o : defs y.c
    gcc -c y.c
z.o : z.c
    gcc -c z.c
```

Le Makefile traduit le graphe de dépendance du programme
une entrée est de la forme :

```
fichier cible : dépendances
  <tab>  commande pour construire le fichier cible
```

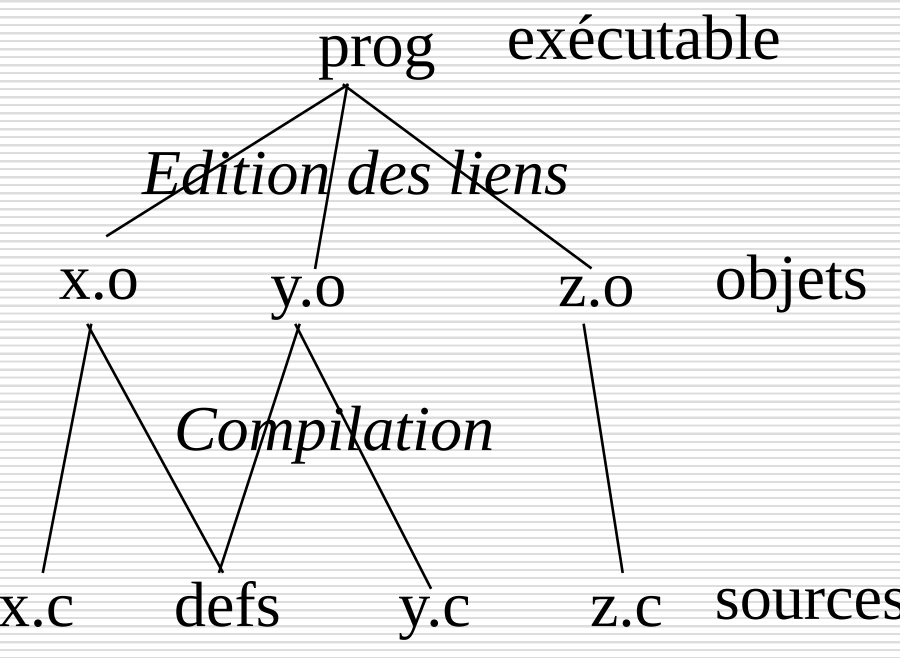
Fonctionnement du Make

L'outil Make utilise le fichier Makefile et les dates de dernières modifications des fichiers pour déterminer si un fichier est à jour

- un fichier est à jour si
  - le fichier existe
  - sa date de dernière modification est supérieure ou égale aux dates de dernière modification de tous les fichiers dont il dépend.

- Si un fichier n'est pas à jour, la commande associée à ses dépendances est exécutée

Fonctionnement du make



Makefile

```
prog : x.o y.o z.o
    gcc x.o y.o z.o -o prog
x.o : defs x.c
    gcc -c x.c
y.o : defs y.c
    gcc -c y.c
z.o : z.c
    gcc -c z.c
```

Modification de z.c : reconstruire le fichier objet `z.o` puis l'exécutable `prog`

Fonctionnement du make

prog exécutable

Edition des liens

x.o y.o z.o objets

Compilation

x.c defs y.c z.c sources

Makefile

```
prog : x.o y.o z.o
      gcc x.o y.o z.o -o prog
x.o : defs x.c
      gcc -c x.c
y.o : defs y.c
      gcc -c y.c
z.o : z.c
      gcc -c z.c
```

Modification de defs : reconstruire les fichiers objet `x.o, y.o` puis l'exécutable `prog`

Exercice

Soit la définition de la syntaxe suivante:

```
<instruction> ::= <identificateur> = <expression> ;
<expression> ::= <facteur> | <facteur> {+ | -} <expression>
<facteur> ::= <terme> | <terme> {*| /} <facteur>
<terme> ::= <identificateur> | <nombre> | (<expression>)
<identificateur ::= <lettre> | <lettre> <chiffre>
<nombre> ::= <chiffre> | <nombre>  <chiffre>
<lettre>::= A | B | C | … | Z
<chiffre>::= 0 | 1 | 2 | … | 9
```

Voici l'instruction que nous allons tenter d'analyser :

```
A1 = (23 * 5 + 2) + B2;
```

(1) Donnez le résultat du découpage effectué par l'analyse lexicale.
(2) Donnez l'arbre de la syntaxe.