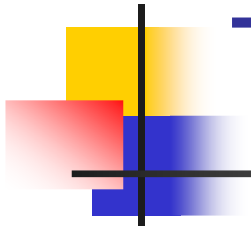
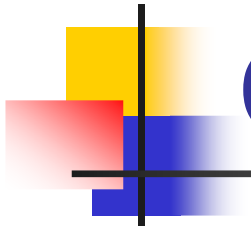
Introduction aux compilateurs

ESI – École nationale Supérieure en Informatique



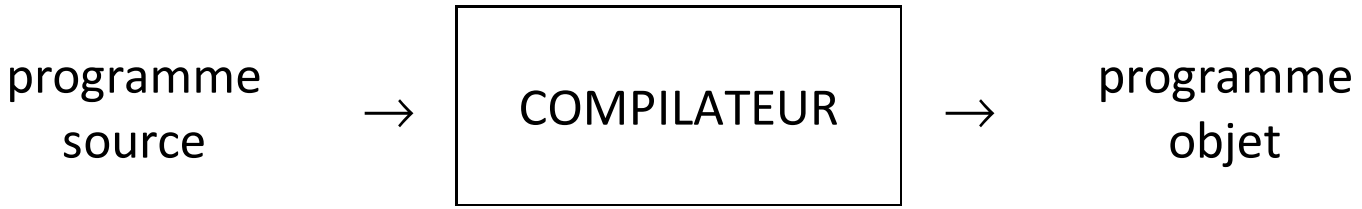
Traduction

- Les programmes qui convertissent un programme de l'utilisateur écrit en un langage quelconque en un programme écrit dans un autre langage sont appelés des traducteurs.
- Le langage dans lequel on écrit le programme originel est le langage source, tandis que le langage après conversion est le langage cible (ou langage objet).



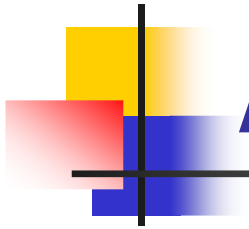
Compilateur

- langage de haut niveau (par exemple C++ ou LISP) et le langage objet est de bas niveau (langage machine)



Autres traducteurs - 1

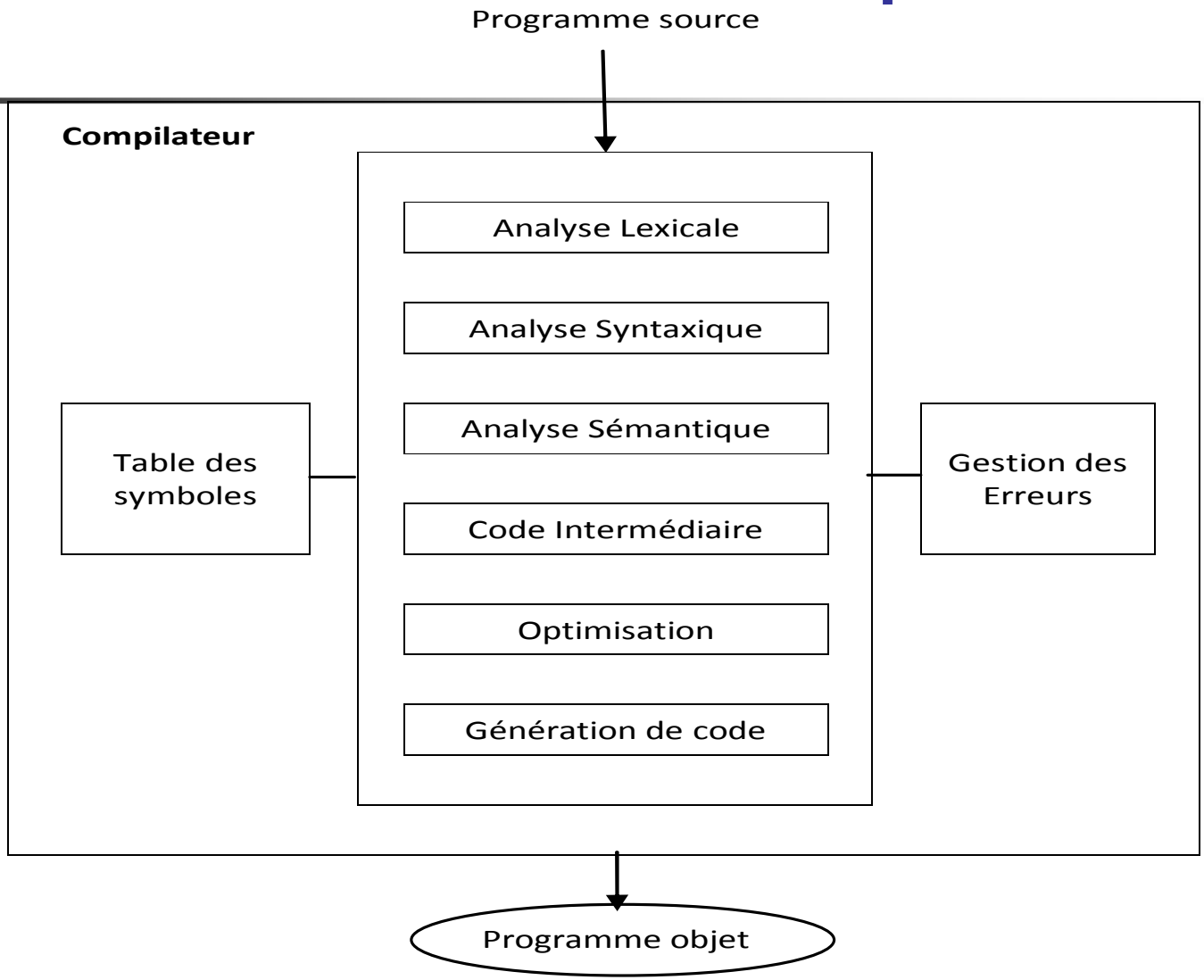
- Lorsque le langage source est essentiellement une représentation symbolique du langage d'une machine, le traducteur est appelé un **assembleur**.
- Le **pré-processeur** désigne un traducteur qui convertit un programme écrit dans un langage de haut niveau en un programme équivalent dans un autre langage de haut niveau.



Autres traducteurs - 2

- Un **interprète** (ou interpréteur) est un traducteur qui ne génère pas un exécutable comme un compilateur mais qui exécute les instructions du programme source l'une après l'autre avec les données correspondantes.
- Les langages LISP (LISt Processing) ou BASIC sont des langages interprétés.

Structure d'un compilateur



Analyse lexicale - 1

- Le flot de caractères formant le programme source est lu de gauche à droite et groupé en unités lexicales, qui sont des suites de caractères ayant une signification collective.
- Les identificateurs, les mots-clés, les constantes et opérateurs sont des unités lexicales.
- Les commentaires et les blancs séparant les caractères formant les unités lexicales sont éliminés au cours de cette phase.

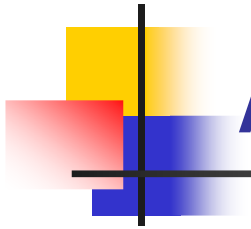
Analyse lexicale - 2

- Soit l'instruction Pascal suivante :

  IF (5 = MAX) THEN GOTO L1 ;

- Les unités lexicales identifiées sont :

IF	**(**	**5**	**=**	**MAX**	**)**	**THEN**	**GOTO**	**L1**	**;**



Analyse lexicale - 3

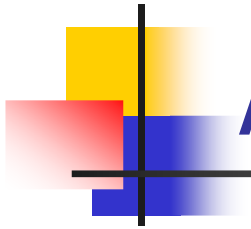
- L'analyseur lexical associe à chaque unité lexicale un code qui spécifie son type et une valeur (un pointeur vers la table des symboles).
- Après analyse de l'instruction précédente on peut avoir la sortie suivante:

  code-if (TS[341] = TS[729]) code-then code-goto TS[554]

Analyse lexicale - 4

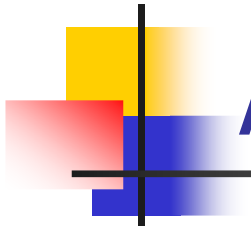
- Table des symboles :

341	5 ; Constante
554	L1 ;
729	MAX ; Identificateur
	.
	.



Analyse syntaxique - 1

- Cette phase consiste à regrouper les unités lexicales du programme source en structures grammaticales (i.e. vérifie si un programme est correctement écrit selon la grammaire qui spécifie la structure syntaxique du langage).
- En général, cette analyse est représentée par un arbre.



Analyse syntaxique - 2

- Considérons la grammaire suivante des expressions arithmétiques et soit à analyser la phrase suivante : A + B * C :

$$E \rightarrow E + T \mid T$$

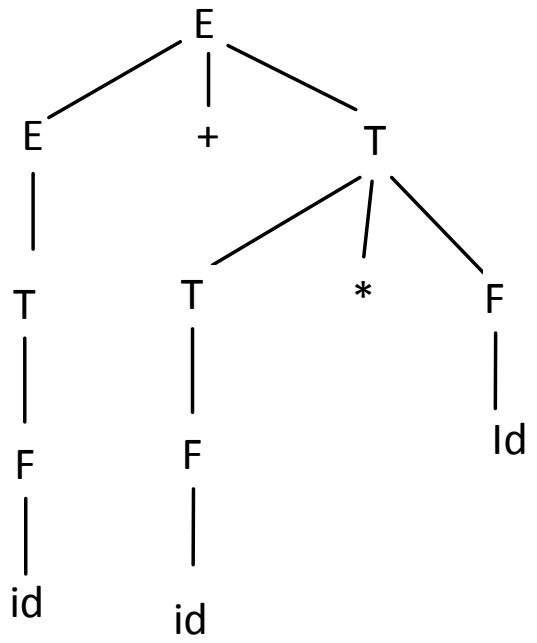
$$T \rightarrow T * F \mid F$$

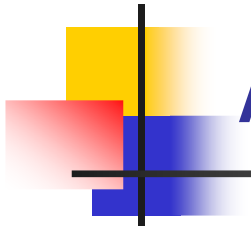
$$F \rightarrow (E) \mid id$$

Analyse syntaxique - 3

A + B * C → Analyse lexicale → id1 + id2 * id3

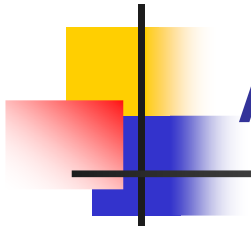
- arbre syntaxique





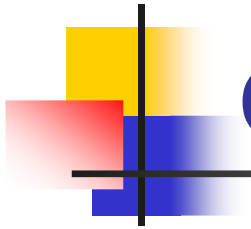
Analyse Sémantique - 1

- Dans cette phase des traitements (portions de code) dits également actions sémantiques sont insérées à des endroits précis de la grammaire pour répondre à des spécificités du langage considéré.



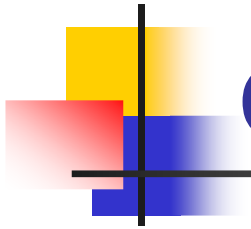
Analyse Sémantique - 2

- Par exemple, cette phase vérifie que les opérandes de chaque opérateur sont conformes aux spécifications du langage source.
- Beaucoup de définitions de langages de programmation exigent que le compilateur signale une erreur chaque fois qu'un nombre réel est employé pour indicer un tableau.



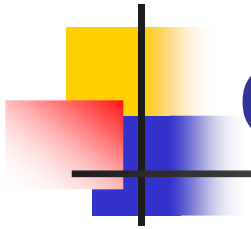
Génération de code intermédiaire - 1

- A l'issue de l'analyse syntaxique et de l'analyse sémantique, certains compilateurs construisent explicitement une représentation intermédiaire du programme source.
- Cette représentation doit avoir deux propriétés importantes: elle doit être facile à produire et facile à traduire en langage cible.



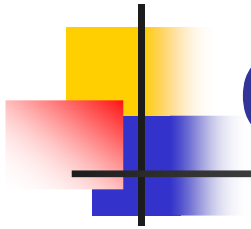
Génération de code intermédiaire - 2

- Plusieurs formes intermédiaires sont possibles.
- La forme intermédiaire ‘appelée code à trois adresses’ est largement utilisée. Dans cette forme, chaque instruction a au plus trois opérandes.



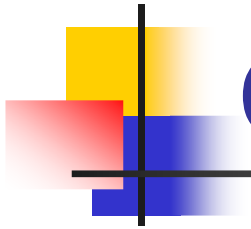
Génération de code intermédiaire - 3

- L'expression arithmétique (id1 + id2 * id3) peut être traduite en code à trois adresses de la façon suivante (temp1 et temp2 sont des temporaires):
- **temp1 := id2 * id3**
- **temp2 := id1 + temp1**



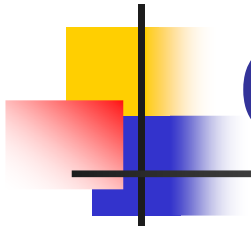
Optimisation - 1

- Cette phase tente d'améliorer le code intermédiaire pour réduire le temps d'exécution ou l'occupation mémoire.



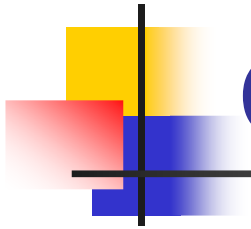
Optimisation - 2

- Déplacement de code invariant :
  - le but est d'extraire du corps de la boucle les parties du code qui sont des invariants (nécessitent une seule exécution). Le nombre total d'instructions exécutées est diminué.
- Elimination du code inutilisé :
  - le compilateur sait reconnaître les parties du code qui ne sont pas utilisées (la raison peut être une erreur de programmation).



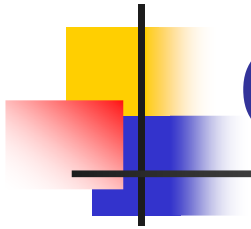
Optimisation - 3

- Elimination des sous-expressions communes :
  - dans le cas où la valeur d'une expression est recalculée en plusieurs endroits du programme, l'optimiseur gère le résultat de cette expression et le réutilise plutôt que de refaire le calcul.



Génération de code - 1

- Le but de cette phase est de convertir le code intermédiaire optimisé en une séquence d'instructions machine.



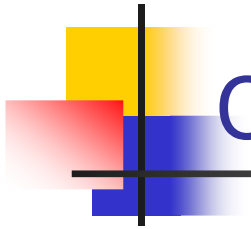
Génération de code - 2

- **Exemple** :
- L'instruction du code intermédiaire T:=A+B est convertie en instructions machine suivantes :

LOAD A

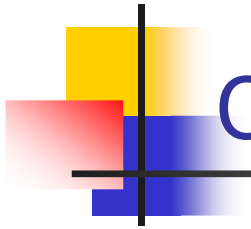
ADD B

STORE T



Outils d'aide à l'écriture de compilateurs -1

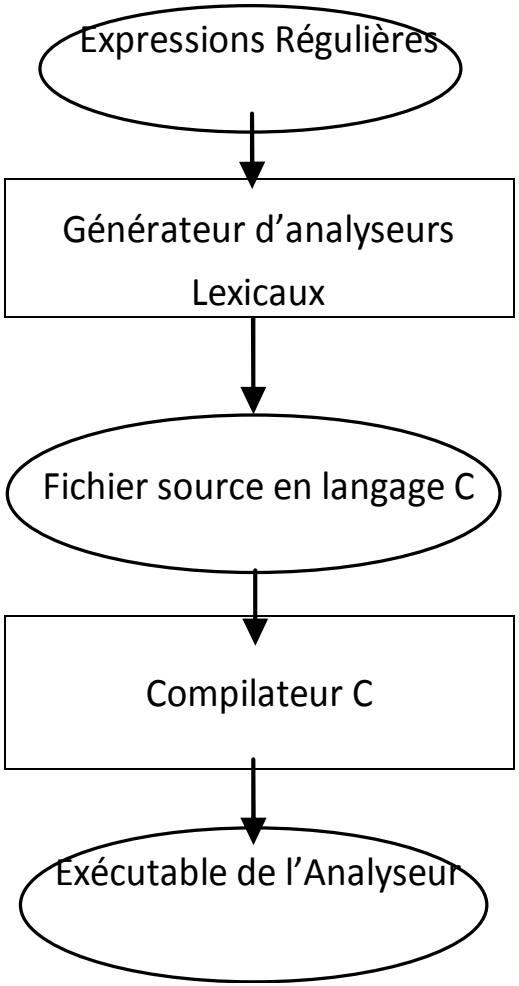
- Des outils spécialisés ont été mis au point pour faciliter l'implantation de certaines phases d'un compilateur.
- Les outils qui rencontrent le plus grand succès sont ceux qui cachent les détails de l'algorithme de construction.



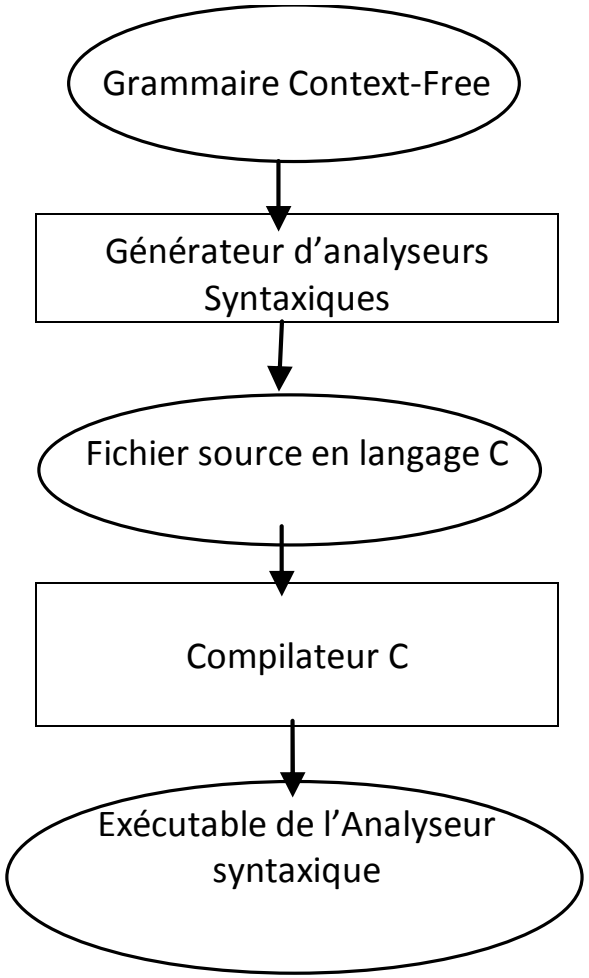
Outils d'aide à l'écriture de compilateurs -2

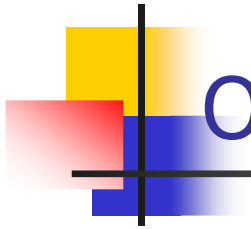
- générateurs d'analyseurs lexicaux;
- générateurs d'analyseurs syntaxiques;
- générateurs de compilateurs (compiler-compiler, compiler-generator,…)

Outils d'aide à l'écriture de compilateurs -3



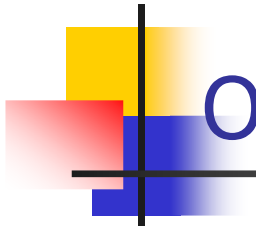
Outils d'aide à l'écriture de compilateurs -4





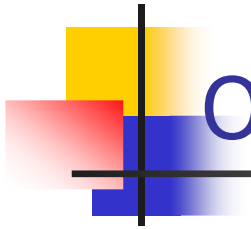
Outils d'aide à l'écriture de compilateurs -5

- Parmi la multitude de générateurs existants, deux outils que sont LEX et YACC constituent des standards de fait.
- Ces outils ont fait leur apparition sous Unix mais des versions sous MS Windows sont également largement répandues.



Outils d'aide à l'écriture de compilateurs -6

- **LEX** est un générateur d'analyseurs lexicaux. A partir d'expressions régulières, LEX génère un code source écrit en langage C qui contient structures de données et algorithmes relatifs à l'analyseur lexical.
- **YACC** (Yet Another Compiler-Compiler) est un générateur de compilateurs qui se base sur une analyse syntaxique LALR(1)

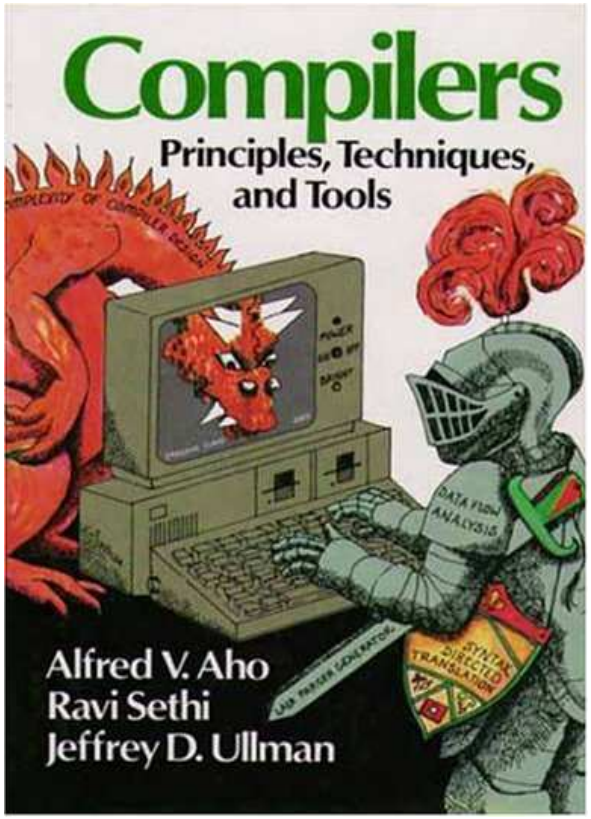


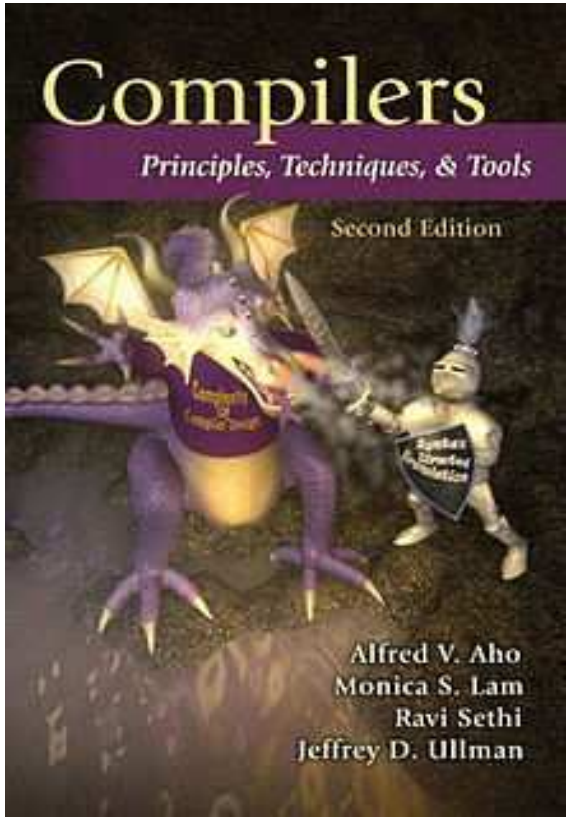
Outils d'aide à l'écriture de compilateurs -7

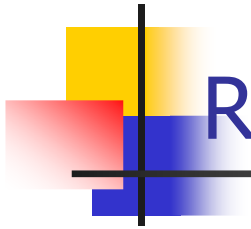
- **JLEX**
- **JAVACC**
- **SABLECC**
- **ANTLR**
- **COCO/R**

Références

- The Dragon Book :





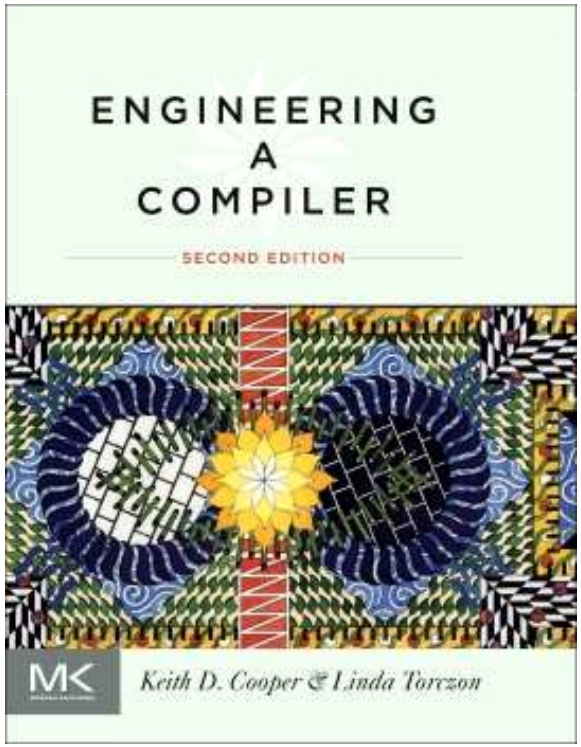


Références

- The Dragon Book :
  - Aho, A.V., & Ullman, J.D. (1972). *The theory of parsing, translation, and compiling*. Prentice-Hall, Inc.
  - Aho, A.V., & Ullman, J.D. (1977). *Principles of compiler design* (Vol. 21). Reading, Mass.: Addison-Wesley.
  - Aho, A.V., Sethi, R., & Ullman, J.D. (1986) *Compilers: Principles, Techniques, and Tools*, Addison-Wesley.
  - Aho, A.V., Lam, M.S., Sethi, R. & Ullman. J.D (2006), *Compilers. Principles, Techniques, and Tools*. Pearson Education, Inc.

Références

- Cooper, K., & Torczon, L. (2011). *Engineering a compiler*. Elsevier.



Références

- Ait-Aoudia, S, (2014). *Compilation cours et exercices corrigés*, Office des Publications Universitaires.